Developers Guide

V 1.3

Prepared by:

Lionel Lim,
Darren Ong,
Yang Kai Ze,
Jerry Ho,
Michelle Toh

The purpose of this guide is to show a detailed outline of the functionality of the application BetterDuke and some of its key features.

Table of Contents

1. Introduction

a. Target user profile

Our target users are NUS students with poor time management. An average NUS student has about 4 to 6 modules each semester, with different workload required for each module. Since the modules taken each semester are different, they may have difficulty managing their tasks for each separate module. Furthermore, NUS students are more often than not always on the move in University travelling from one faculty to another for their lessons, this is why we utilise the convenience of their laptop to give them a solution to manage their never-ending tasks.

b. Value proposition

Given that our target users are NUS students, schedules are hectic with assignments, lessons and even co-curricular activities. This product would sort all these activities out and aid in their planning so that they are able to better manage their time. Additionally, tertiary students need their laptop for school work and since our software is a desktop application, it makes it easier for students to make use of this software with convenience and accessibility.

c. Purpose

Aid in planning their schedule. This can be done so by giving them an overview of their plans and deadlines.

2. Setting up

1.1 Prerequisite

1. JDK `11` or above
2. IntelliJ IDE

1.2 Setting up

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` (Max/Linux: `./gradlew processResources`). It should finish with `BUILD SUCCESSFUL` message.
   This will generate all resources required by the application and tests.

1.3 Verifying the setup

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

1.4 Configurations to do before writing code

1.4.1 Configuring the coding style

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
    a. 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
    b. For `Import Layout`: The order is `import static all other imports`, `import java.*`, `import javafx.*`, `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.

1.4.2 Updating documentation to match your fork

After forking the repo, links in the documentation will still point to the AY1920S1-CS2113T-W12-4/main repo. If you plan to develop this as a separate product (i.e. instead of contributing to the AY1920S1-CS2113T-W12-4/main) , you should replace the URL in the variable `repoURL` in DeveloperGuide.pdf and UserGuide.pdf with the URL of your fork.

1.4.3 Getting started with coding

When you are ready to start coding, we recommend that you get some sense of the overall design by reading about BetterDuke's architecture.

3. Design

3.1 Architecture

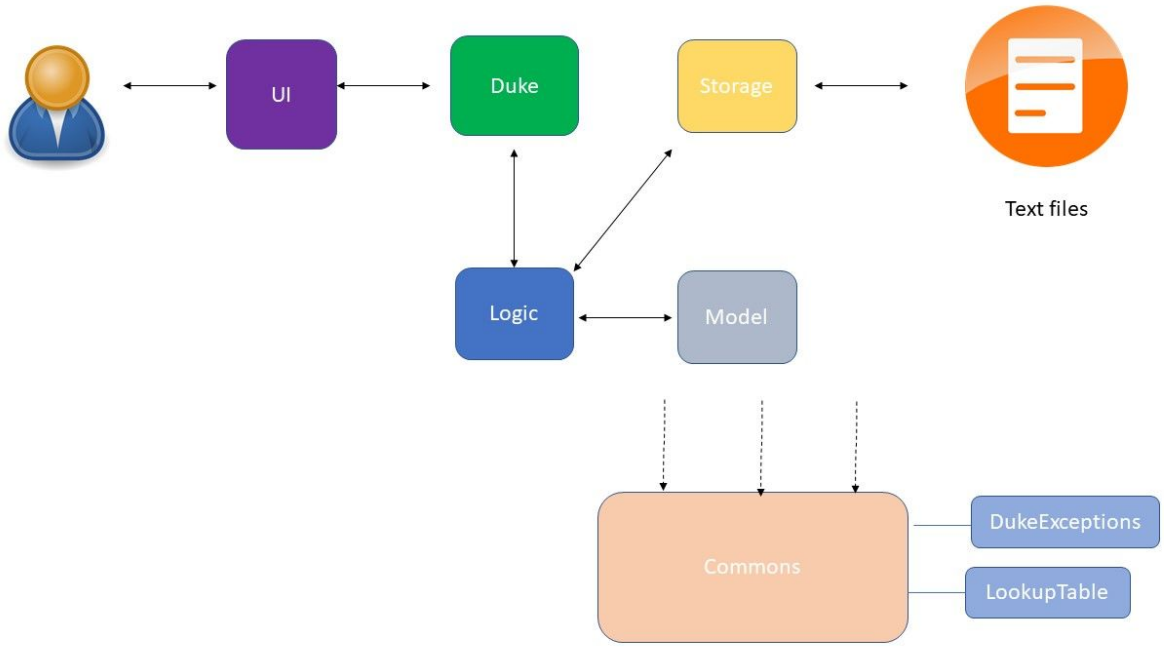


Figure 1.The Architecture Diagram given above explains the high-level design of the App. Given below is a quick overview of each component.

Main has 2 classes called Main and MainWindow. It is responsible for:

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

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

- `LookupTable` : Used by many classes to convert academic dates to standard date time format and vice versa.

The app consists of five `components`:

- `Logic : The command executor`
- `Model : Modifies and stores data of App in-memory`
- `Storage : Read data from and write data to files`
- `UI: The UI of the App`
- `DukeExceptions: Custom Duke exceptions that may be raised from contextual errors while executing commands in BetterDuke.`

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

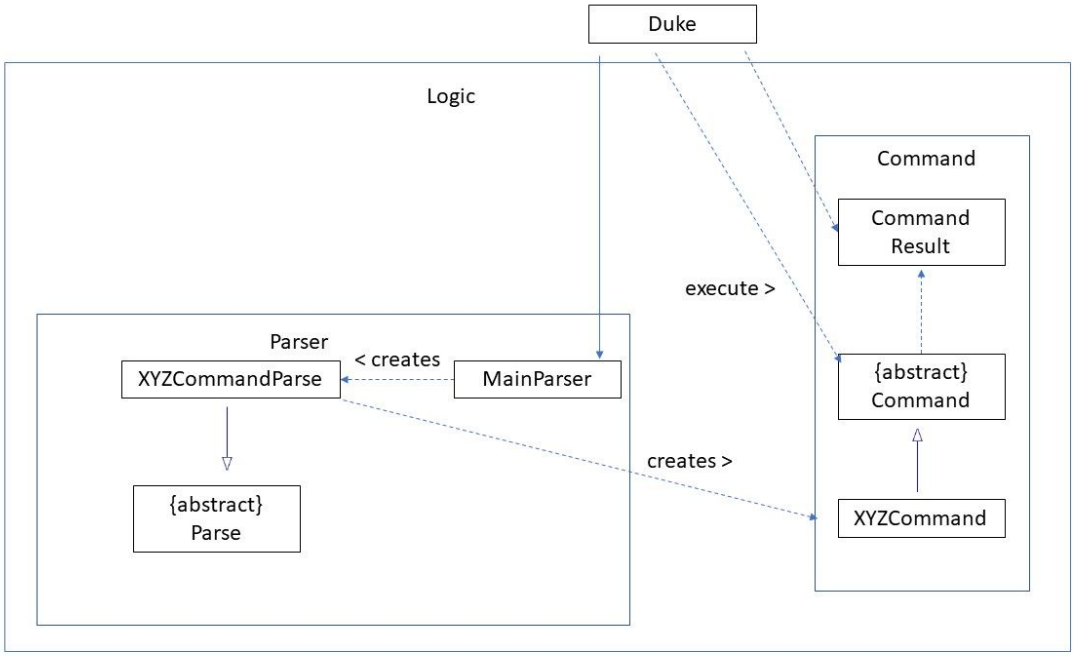


Figure 2. Class Diagram of Logic Component

Event- Driven design

How the architecture components interact with each other

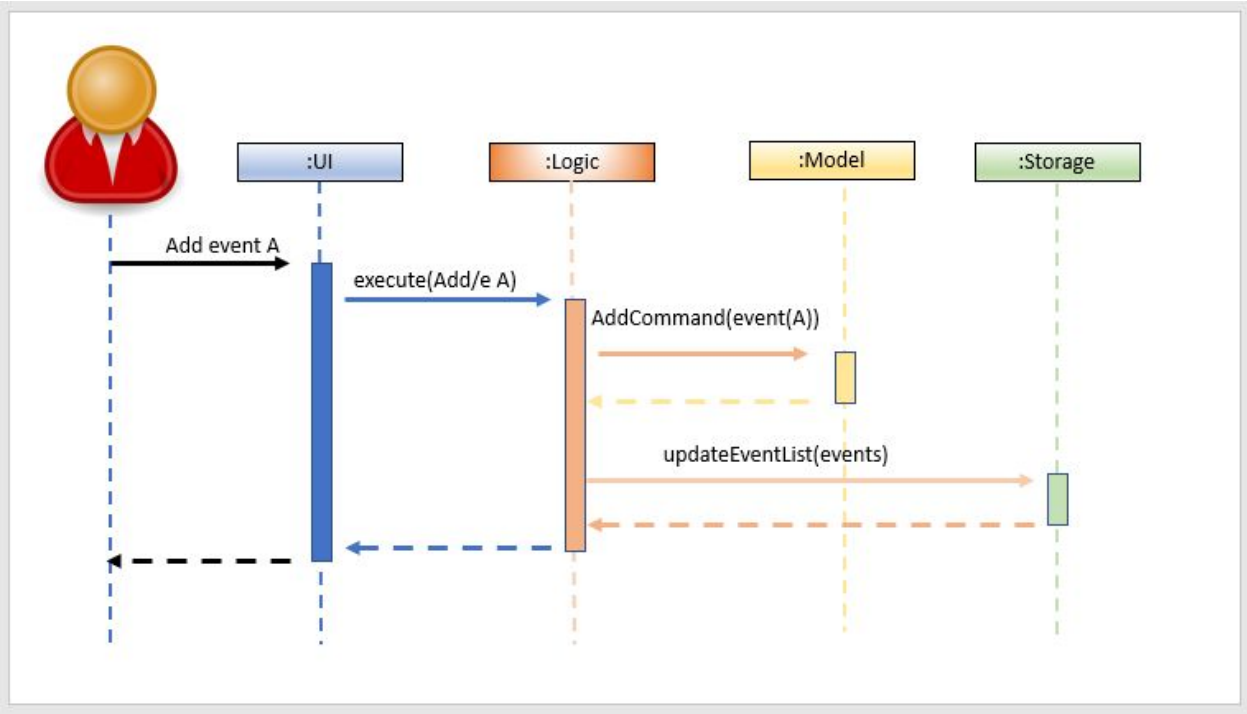
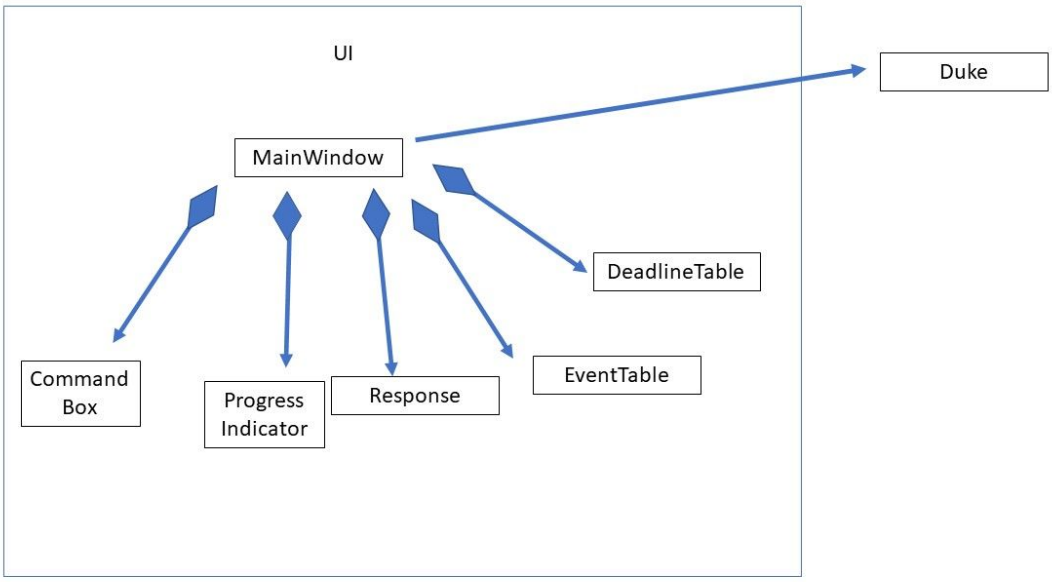


Figure 3: The *Sequence Diagram* below shows how the components interact with each other for the scenario where the user issues the command `Add/e A`.

3.2 UI component



The `UI` consists of `Main`, `MainWindow` and `AlertBox`. `MainWindow` makes use of `DeadlineView`, `DukeResponseView` and `ProgressController` which handles all displays and interactions with users.

The `UI` component uses `JavaFx UI` framework. The layout of these `UI` parts are defined in matching `.fxml` files that are in the `src/main/resources/view` folder. For example, the layout of `MainWindow` is specified in `Mainwindow.fxml`.

The UI component,

- Executes user commands using the Logic component.
- Listens for changes to Model data so that the UI can be updated with the modified data.

3.3 Logic component

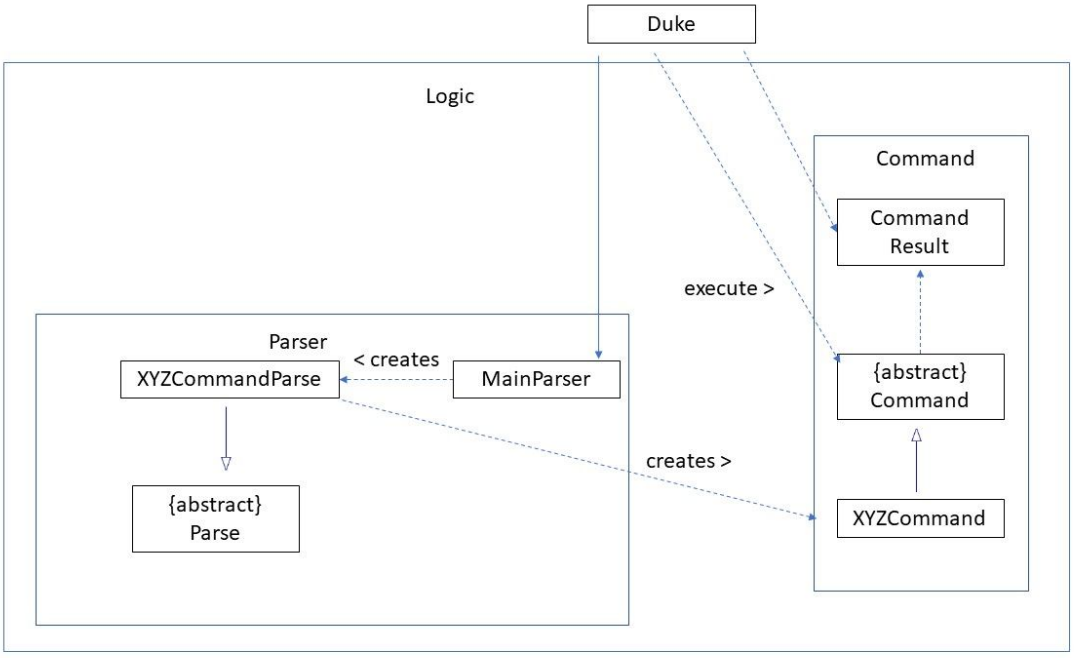


Figure 4. Structure of the Logic Component

The `Logic` component handles the parsing of user input and interacts with the Command objects.

- `Logic` uses the `MainParser` class to parse user command.
- This results in a `Command` object which is executed by `Duke`.
- The command execution can affect the `Model` (e.g. adding a event).
- The result of the command execution is encapsulated as a `CommandResult` object which is passed back to the `UI`.
- In addition, the `CommandResult` object can also instruct the `Ui` to perform certain actions, such as displaying help to the user.

3.4 Storage component

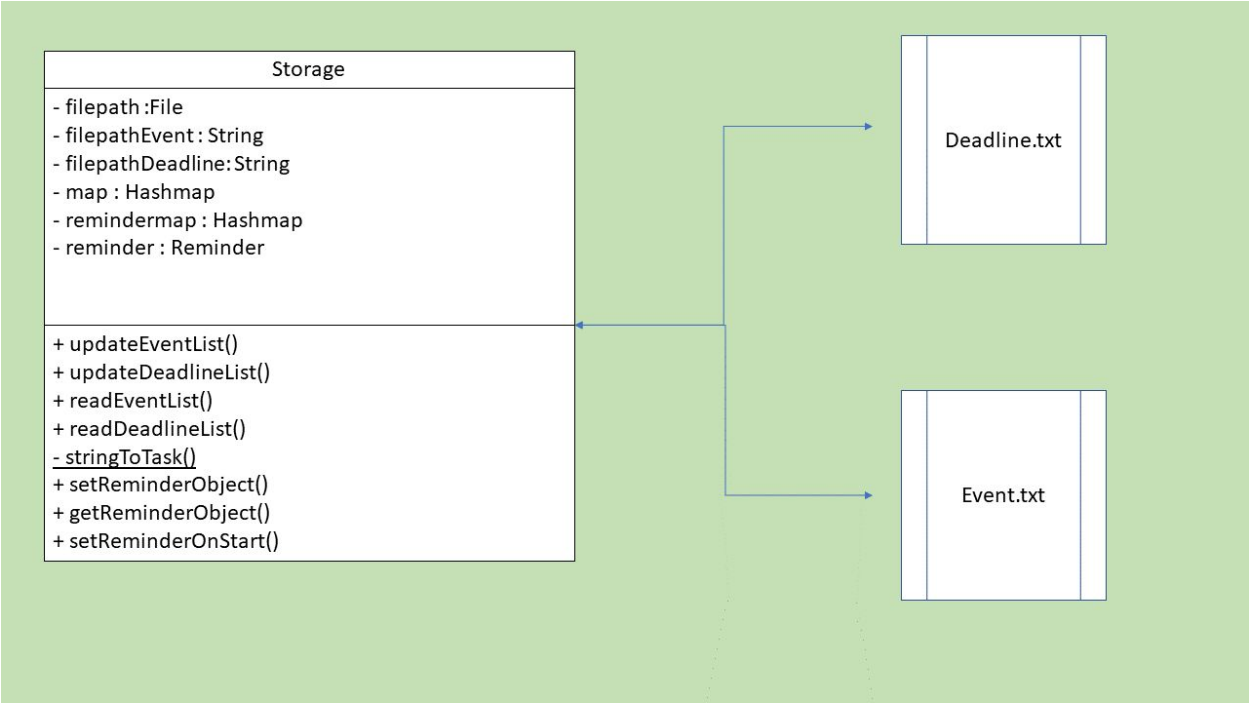


Figure 5: Class Diagram for Model component
Classes involved: Storage

The Storage class ensures data persistence upon termination of BetterDuke and is the intermediary between the external files and the Model component.

3.4.1. External files

Data in BetterDuke is saved and loaded from two main text (.txt) files.

- deadline..txt: Contains the current list of deadlines
- event.txt: Contains the current list of events

3.4.2. Storage attributes and methods (Loading and Saving)

Loading and saving to/from the above mentioned external files are vital for Storage to function. In order to do both, Storage has attributes of File type which records the paths to the local external files. There is also a temporary hashmap created to retrieve and store data from the Model component.

For loading from files, we need to read and write both external files mentioned above, so dedicated methods for deadline.txt and event.txt are present

3.5 Model component

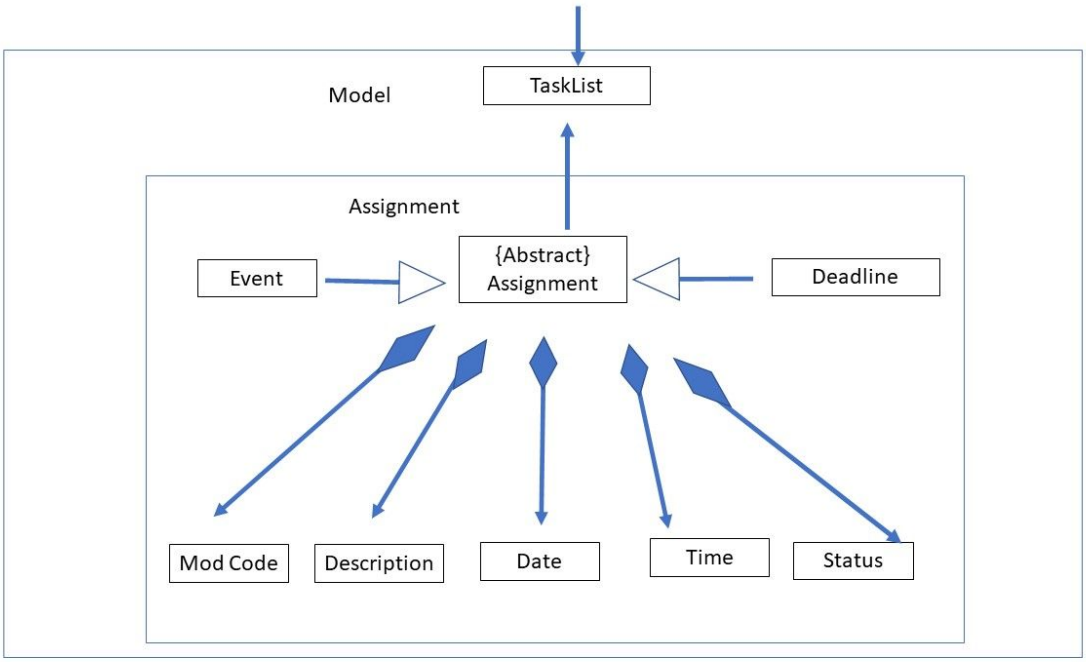


Figure 5. Structure of the Model Component

The `Model` component handles the in-application task data.

- Stores all data according to user's input.
- Data input by users will be parsed through Logic component to extract all relevant information as seen in figure 5.
- Main component that will pass information to all other components.

4. Implementation

4.1 Filter by Keyword feature

4.1.1 How the feature is implemented.
The filter mechanism is facilitated by FilterCommand, which extends the Command Abstract class.
The format of the command is as follows: show/filter “keyword”

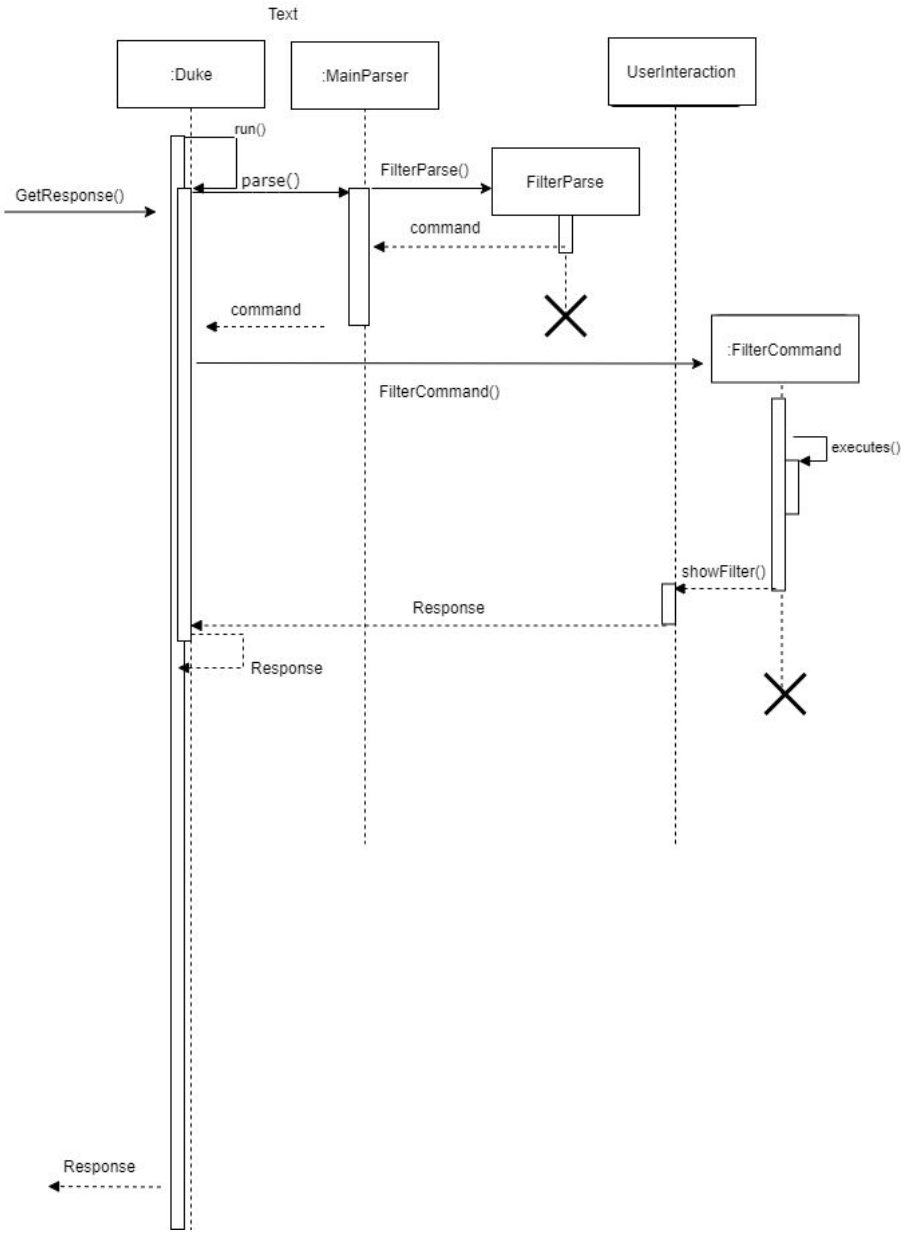
Below is a step by step sequence of what happens when a user enters a filter command.

1. The user enters a filter command using the following command line input `show/filter keyword`. After command has been parsed, FilterCommand is created.
2. FilterCommand#execute() then operates 2 For loops for both events list and deadline list.
3. The tasks are converted to string form using TaskList#toString() method to facilitate easy searching.
4. The keyword is then match to the events and deadlines using Java function .contain().
5. A String message is then returned back by calling Ui#showFilter() from FilterCommand for display by JavaFx graphic user interface.
6. If the keyword is not found in any of the task, an alertbox will be returned indicating a mismatched keyword.

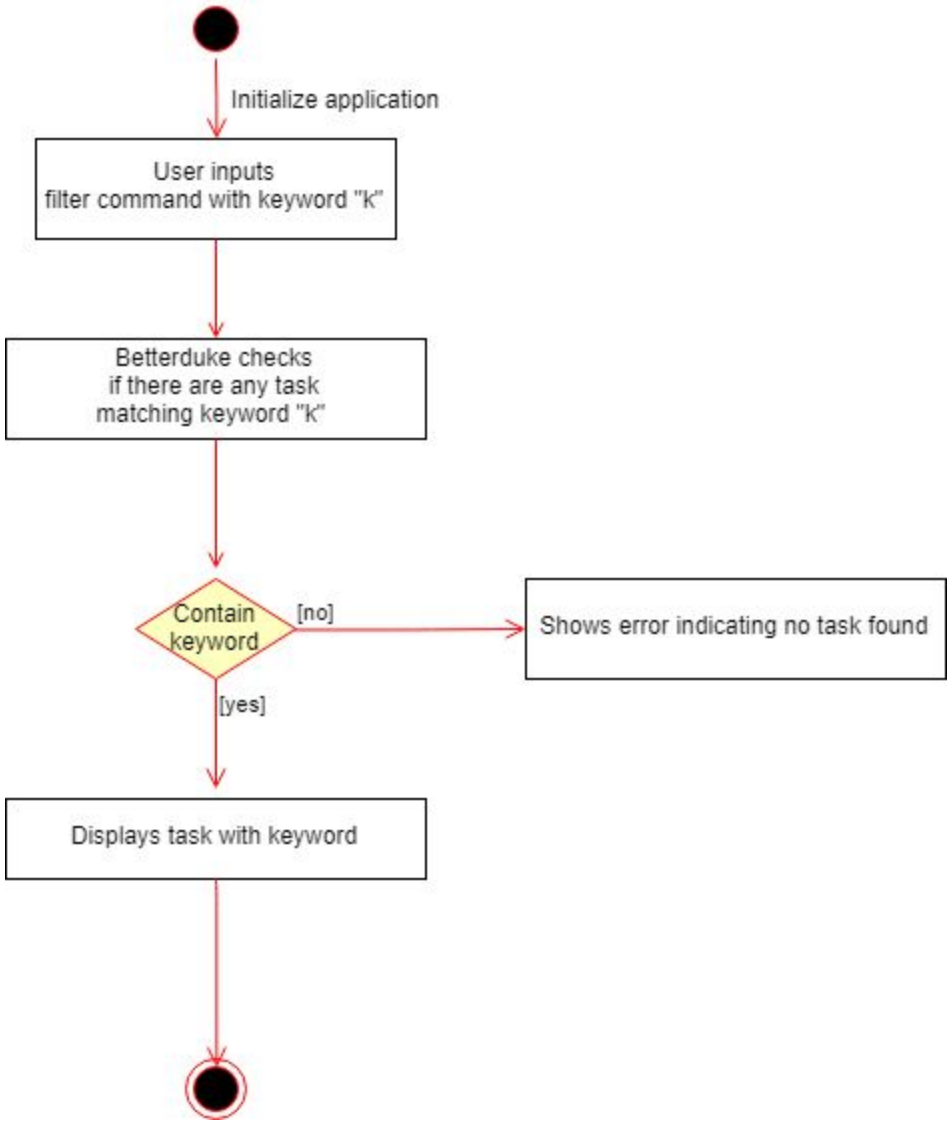
* This function matches both uppercase and lowercase matches of the keyword to the task in string format.

4.1.2 Why it is implemented that way.
It is implemented this way to allow user to enter a short command for easy usability. The user may enter the keyword in both uppercase and lowercase and both uppercase and lowercase instances of the keyword would be returned. As long as the keyword or phrase is a sub-string in the description field, the task is returned as a match.

The following sequence diagram shows how the filter operation works.



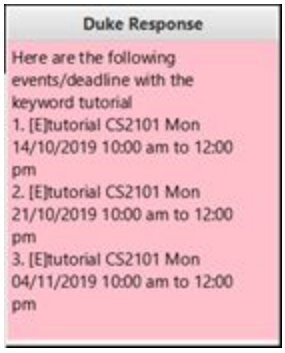
The following activity diagram summarizes what happens when a user executes a command:



4.1.3 How the feature works

Step 1: Input Command in the Command textbox and press submit

Step 2:System will display response with keyword “cs2113” in the response box



4.1.4 Design Considerations

Design Considerations

Aspect	Alternative 1 (Current Choice)	Alternative 2
1)Function to use for search function	Search using Java contains to match substring of task Pros: Efficient, results can be retrieved in O (1) time Cons: May return inconsequential results (filter 1 may return cs2101, 123, task1)	Search using Java regex to match specific word Pros: Results returned are more specific, only words are returned Cons: Inefficient, especially if there are many tasks, O(n) time

2)Data structure to support the undo/redo commands	Search by arraylist Pros: Easy to implement Cons: Inefficient, especially if there are many tasks, O(n) time	Search by HashMap Pros: Efficient, results can be retrieved in O(1) time Cons: Searching by Key may not return all instances of keyword Difficult and tedious to map by keyword

4.1.5 Future Optimisations/ Version 2.0

- Implementing filter feature using multiple keywords
    - Allows users to enter multiple keywords so as to better specific tasks that the user wants.
- Suggested list of keyword prompts
    - Shows a list of keywords common in all existing tasks allowing users to select keyword more easily

4.2 Remind feature

4.2.1 Implementation

The remind feature mechanism is facilitated by RemindCommand and Reminder. RemindCommand extends Command with an abstract execution method specific for the remind feature. It then calls on Reminder which implements the following core operations:

- Reminder#setReminderThread() - Runs a new thread using Timer and TimerTask for a reminder pop-up at set timing by user.
- Reminder#removeTimerTask() - Removes an existing reminder thread that was created previously.

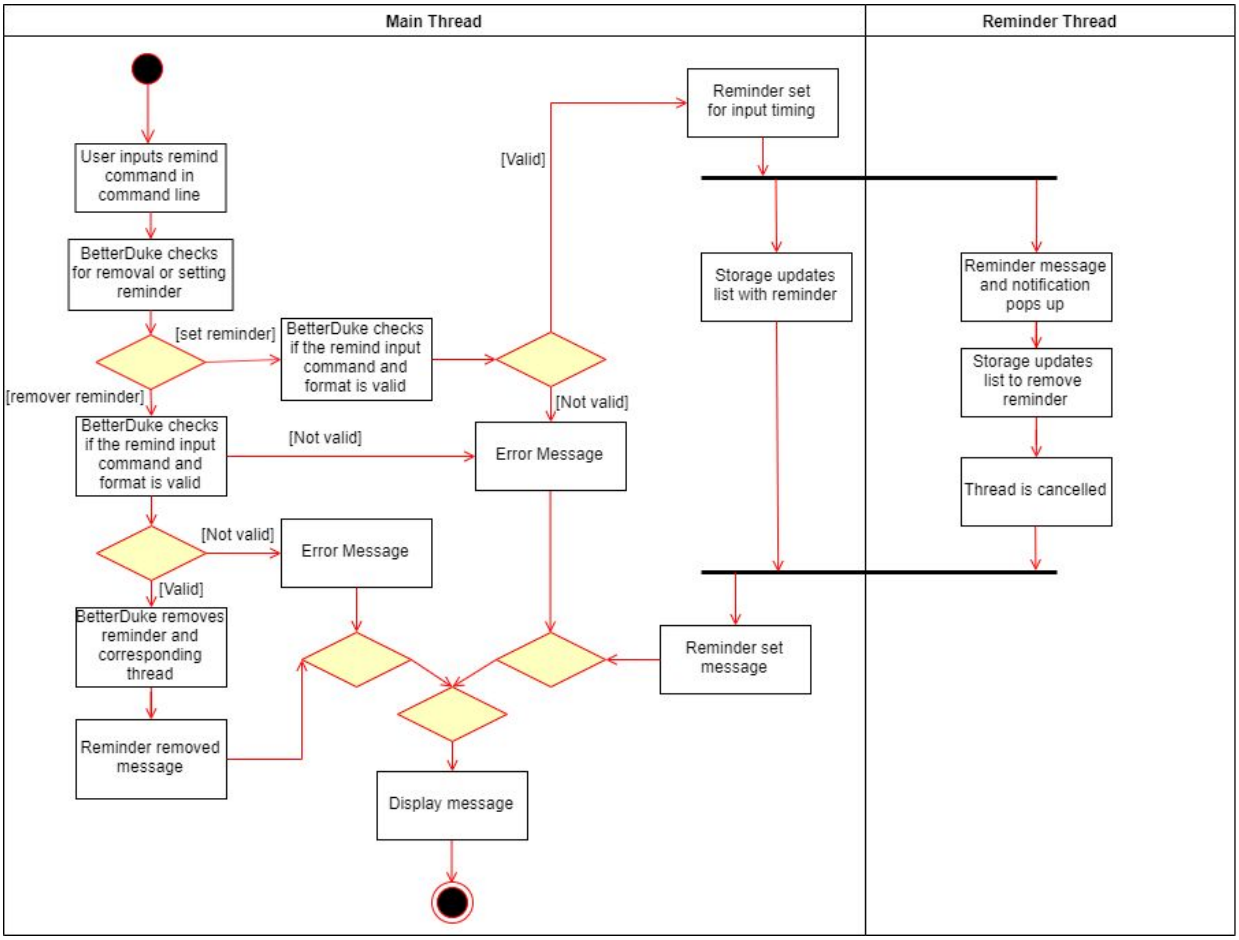
These operations are exposed in RemindCommand as RemindCommand#execute().

Below is a step by step sequence of what happens when a user sets a reminder explained.

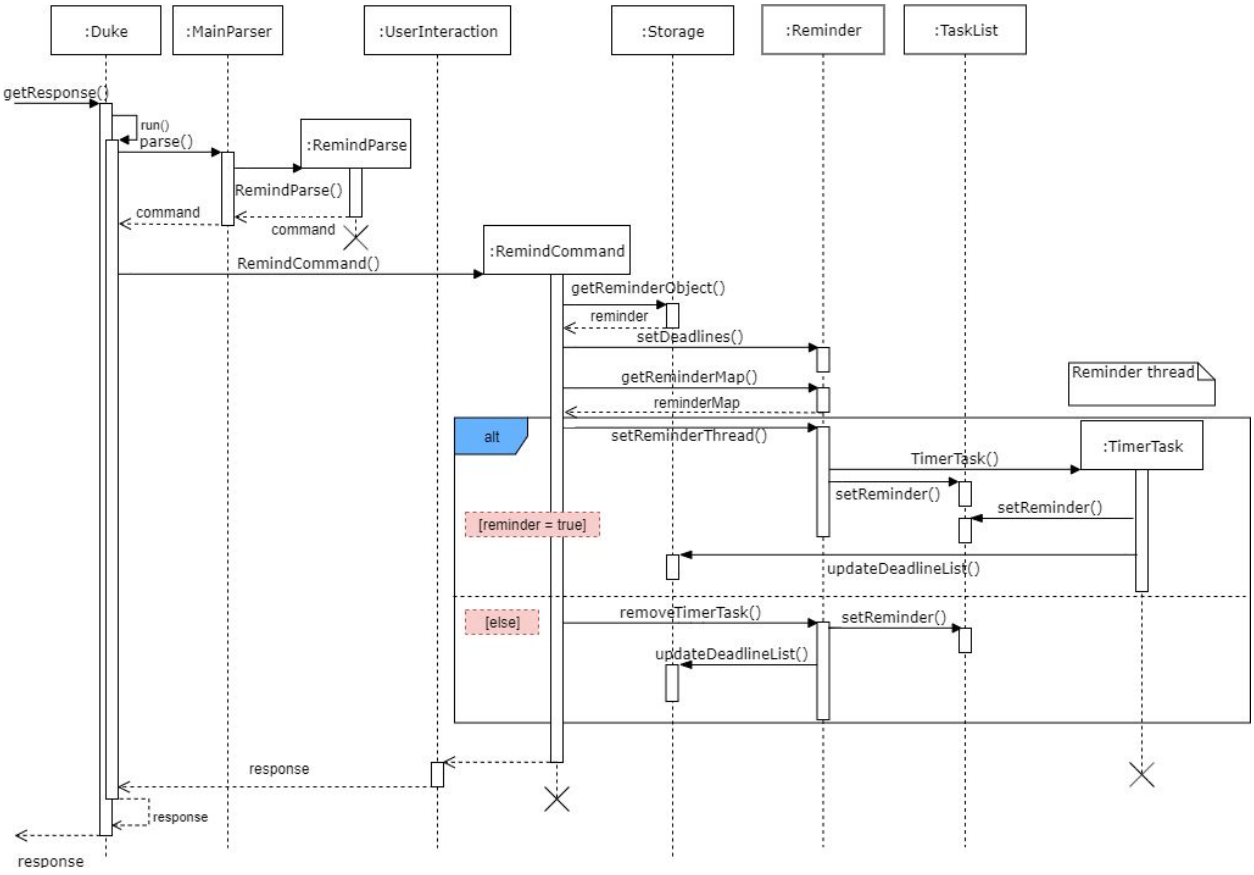
1. The user calls on remind command using the following command line input. `remind/set...` to set a new reminder. After command has been parsed, RemindCommand is created.
2. RemindCommand#execute() then retrieves Reminder object from the storage using Storage#getReminderObject() to enable methods in Reminder for RemindCommand.
3. Multiple checks are further conducted in RemindCommand#execute() to ensure a valid remind command input before calling Reminder#setReminderThread().
4. Reminder#setReminderThread() then makes use of Timer and TimerTask for future execution in the JavaFx main thread.
5. A String message is then returned back by calling Ui#showReminder() from RemindCommand for display by JavaFx graphic user interface.

Removal of a reminder is exactly the same as setting a reminder, with the exception of the step 4 and 5. In step 4, Reminder#removeTimerTask is called instead and thread is removed as timer is cancelled in this method. In step 5, Ui#showCancelReminder is called by RemindCommand instead.

Below is an activity diagram which shows what happens when a user types the remind command:



Below is a sequence diagram which shows how set reminder operation works:



4.2.2 Additional implementation outside of RemindCommand

In an effort to retain reminders set by users after software's process is terminated, remind feature is additionally facilitated by Storage and Reminder to set reminders automatically based on the save data. However, only the following core operation is implemented:

- Reminder#setReminderThread() - Runs a new thread using Timer and TimerTask for a reminder pop-up at set timing by user.

This operation is exposed in Storage as Storage#setReminderOnStart().

Sequence is similar to the main implementation of setting reminder when it reaches Reminder#setReminderThread(). However, initial steps are as follows:

1. The user launches the application. Storage and Reminder objects are created in Duke.

2. Reminder object is passed to storage by calling the method Storage#setReminderObject() in Duke to enable methods in Reminder for Storage.
3. Duke then calls method Storage#setReminderOnStart() to execute Reminder#setReminderThread() based on the save data read in Storage.

This operation only happens when there are reminders set by user after application process is terminated.

4.2.3 Reasons for current implementations

4.2.3.1 Threading

Threading is used because of the nature of setting a reminder. As a result, Timer class and TimerTask class from Java utilities package is used to facilitate scheduling of reminder tasks for future execution in JavaFx main thread.

4.2.3.2 Reminder class

Main reason for a separate class from RemindCommand for main implementation is because the lifeline of Reminder needs to be as long as Duke. This cannot happen with RemindCommand as the lifeline of RemindCommand gets destroyed when it returns a String message. As a result of this, all thread and reminder data can be stored indefinitely so that removal/setting of thread and reminder can happen anytime throughout the lifetime of the application.

Additionally, this facilitates the additional implementation mentioned above, allowing for threading to happen automatically when the application is launched.

4.2.3.3 Availability for deadlines only

Main reason for reminder being available only for deadlines is because of the nature of academic events. Academic events such as lectures and tutorials are already fixed at a certain time period which warrants the unavailability of reminders for events in the week view table.

4.2.4 Known issues

4.2.4.1 ControlsFX notifications without stage

Current reminder pop-up implementation uses AlertBox class that was manually coded and Notifications by ControlsFX. When application is hidden in system tray, NullPointerException is thrown because stage is not shown. As a result, an empty

transparent window is created to be used as the stage to bypass this issue for the notification to pop-up. This window has to either be closed manually or closed by clicking on notification pop-up. This is a known issue that ControlsFX closed without fixing.

4.2.4.2 Save data limitation

Issue only happens when multiple reminders are set for the same task. If application is terminated before reminder pops up, only the latest time for which a reminder was set for that particular task would be added to a new thread when application is re-launched. All old reminder timings for that particular task would be gone.

4.2.5 Design considerations

Aspect: Display reminder notification to user.
Alternative 1: Notification pop-up (Current implementation)
Pros: Notification can be seen even with software minimized.
Cons: An extra transparent window has to be opened for the notification (See 4.2.4.1).
Alternative 2: AlertBox notification
Pros: Application icon will be flashing to alert users and AlertBox message will appear indefinitely until the user closes it.
Cons: User will not be able to see what notification it is and user cannot use the software until AlertBox message is closed.

4.3 Find Free Time feature

4.3.1 Implementation

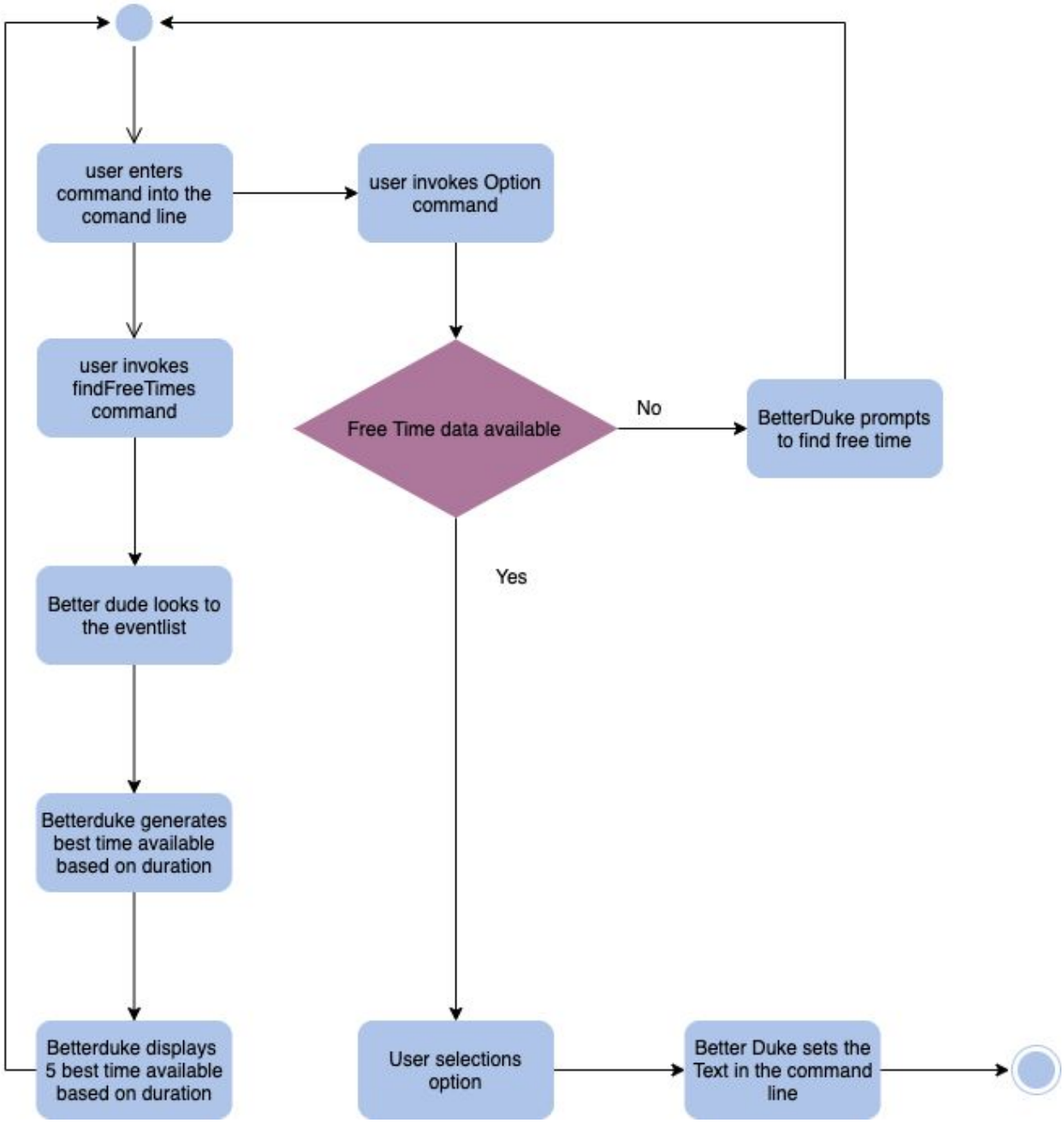
The find free times feature mechanism is facilitated by `FindFreeTimes` command and `RetrieveFreeTimes` command. The command `FindFreeTimes` which extends `Command` class with an abstract execution method specific to `FindFreeTimesCommand`. It then calls on `RetrieveFreeTimesCommand` if the user decides to use one of the 5 suggested options.

- `FindFreeTimesCommand#mapDataMap()` - Maps the list of events that is after the current date and time into dataMap
- `FindFreeTimesCommand#findFindTime()` - Finds the best time available with the list of events in dataMap into freeTimeData
- `FindFreeTimesCommand#setOutput()` - Generates the output to be shown using data in the freeTimeData

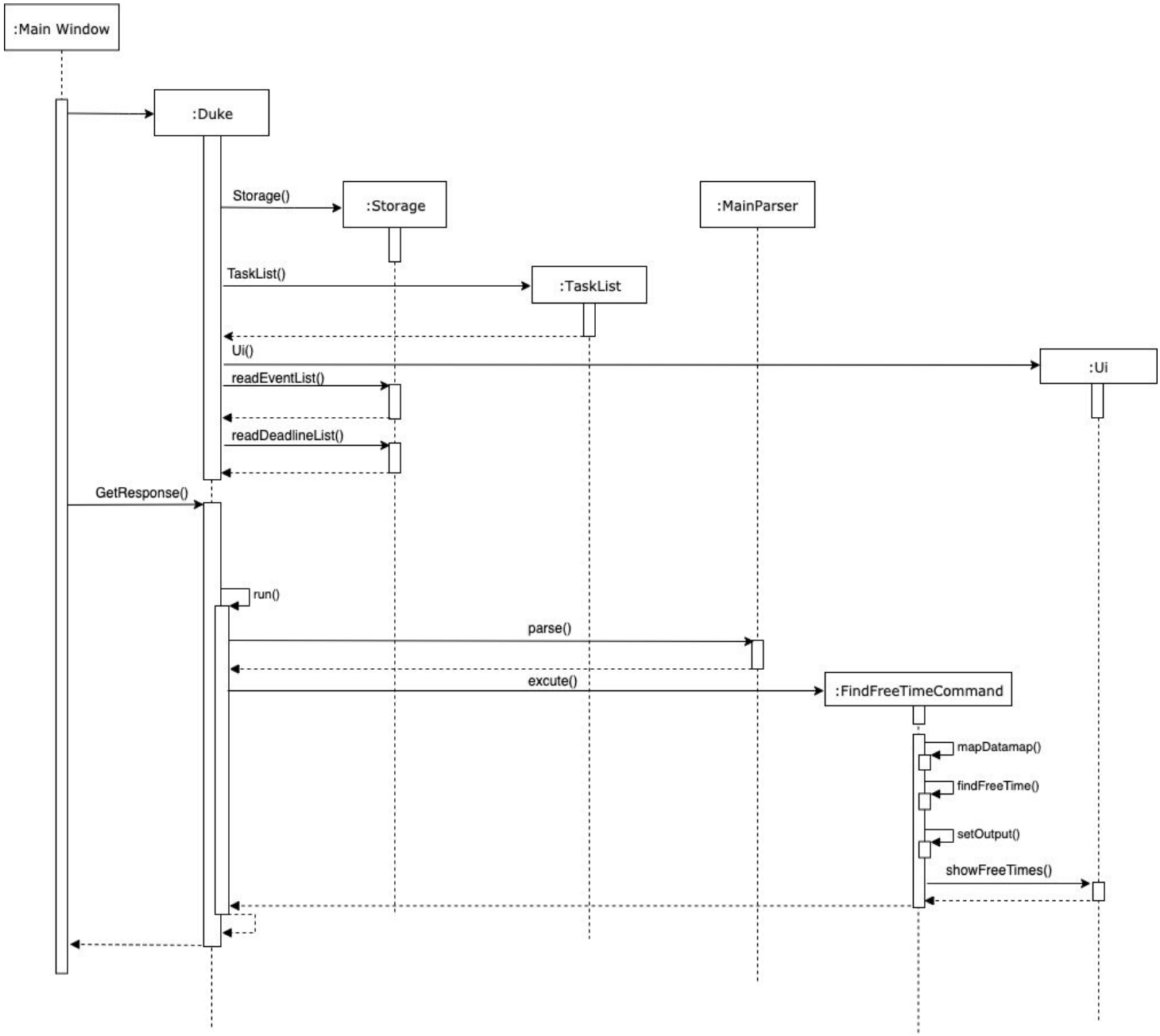
These operations are exposed in FindFreeTimesCommand as FindFreeTimesCommand#execute().

Below is the implementation when `FindFreeTimesCommand` is called.
Step 1: The user launches the application, BetterDuke and enters an input `find 'x' hours`, where 'x' is a digit between 1 -16.
Step 2: BetterDuke will display the 5 best time available based on the duration, 'x' and the list of event in the storage, entered by the user through an alert box.
Step 3: If the user choose not to use any of the free times displayed to him, skip this step and step 4. Otherwise, user can choose to either the first option, the second option, the third option, the fourth option, or the five option by entering `retrieve/ft 1`, `retrieve/ft 2`, `retrieve/ft 3`, `retrieve/ft 4` or `retrieve/ft 5` in the command line respectively.
Step 4: The application will automatically enter the input choice chosen by the user into the command line and run the program again using this input.

The following Activity diagram shows find free time feature operation flow.



The following sequence diagram is how the `FindFreeTimes` command operation works.



4.3.2 Additional implementation outside of `FindFreeTimesCommand`.

If the user were to choose to either get the previous first or second, third, fourth or fifth option from the list displayed to them, the following operation will be implemented:

- `RetrieveFreeTimesCommand#getSelectedOption()` - Sets the text in command line with the option selected.

4.3.3 Reasons for current implementation

The user will not need to browse through the list of events to look for available time. As the algorithm will find the best hour in the available period this will provide a smooth usage for the user to manage his/her time.

4.3.4 Design Considerations

Aspect: Display of the free time slots to the user
Alternative 1: AlertBox
Pros: Easy orientation of the details as AlertBox is resizable. By example, ‘option 1:’ new line ‘option 2:’ and etc.
Cons: AlertBox data is static user may forget options offered and unable to retrieve data findFreeTimeCommand to be rerun
Alternative 2: TableView
Pros: The data is storage and populated in the TableView allows user to revisit data
Cons: The data shown have to be wrapped to the TableColumn size results in formatting issues

4.4 Retrieve Free Time Feature

4.4.1 Implementation

The command `RetrieveFreeTimes` which extends `Command` class with an abstract execution method specific to `RetrieveFreeTimesCommand`. It then calls on `FindFreeTimesCommand` to retrieve the 5 suggested option.

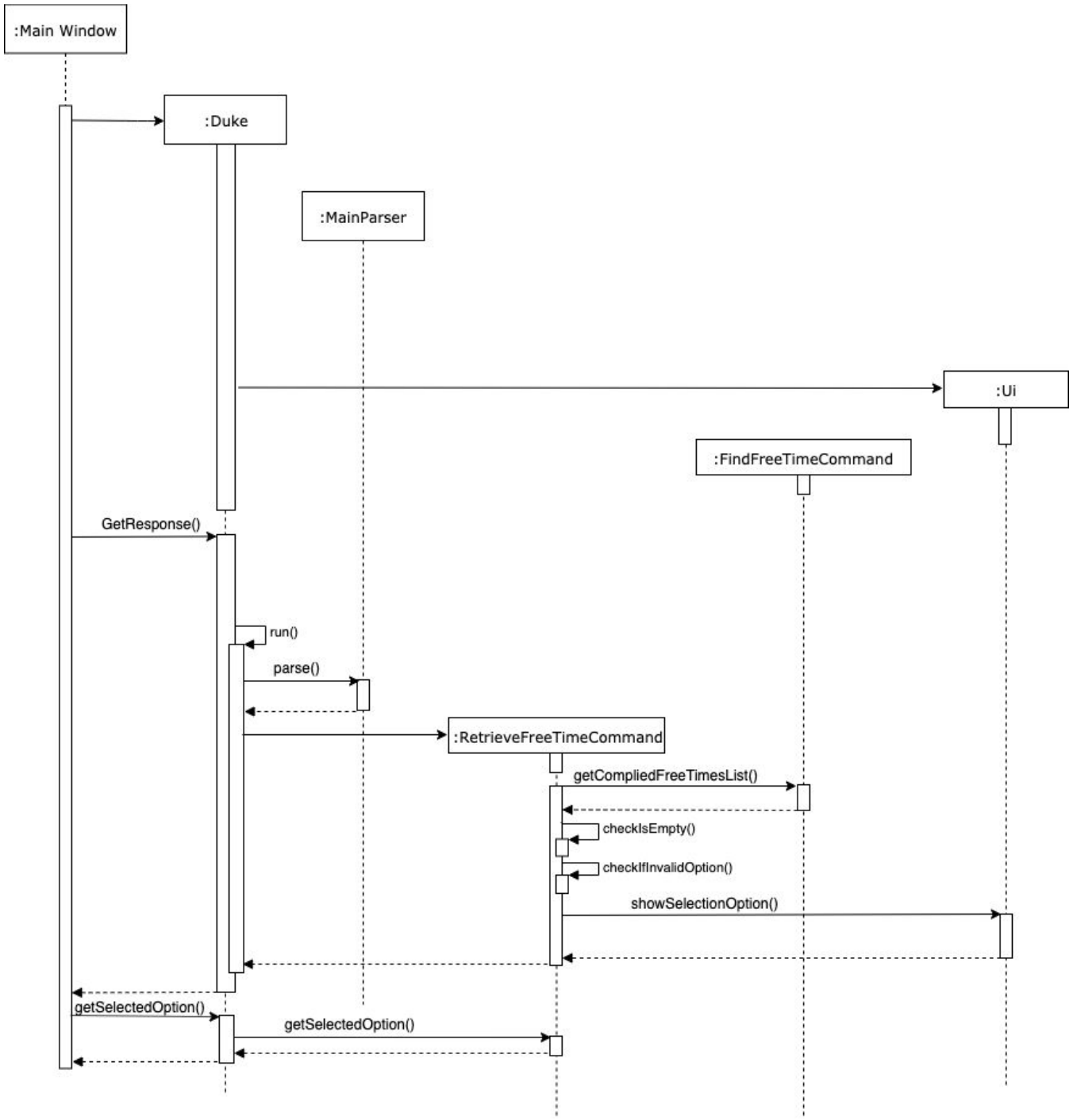
- `FindFreeTimesCommand#getCompiledFreeTimesList()` - Retrieve the list of options generated by findFreeTimesCommand
- `RetrieveFreeTimesCommand#checkIsEmpty()` - Checks if the user requested for find time with findFreeTimesCommand before entering this command.
- `FindFreeTimesCommand#checkInvalidOption()` - Checks if the user entered a valid option.

- `FindFreeTimesCommand#setOutput()` - Generates the output to be shown using data in the freeTimeData

These operations are exposed in RetrieveFreeTimesCommand as RetrieveFreeTimesCommand#execute().

Below is the implementation when `RetrieveFreeTimesCommand` is called.
Step 1: The user launches the application, BetterDuke and enters an input `find 'x' hours`, where 'x' is a digit between 1 -5.
Step 2: BetterDuke will display the 5 best time available based on the duration, 'x' and the list of event in the storage, entered by the user through an alert box.
Step 3: If the user choose not to use any of the free times displayed to him, skip this step and step 4. Otherwise, user can choose to either the first option, the second option, the third option, the fourth option, or the five option by entering `retrieve/ft 1`, `retrieve/ft 2`, `retrieve/ft 3`, `retrieve/ft 4` or `retrieve/ft 5` in the command line respectively .
Step 4: The application will automatically enter the input choice chosen by the user into the command line and run the program again using this input.

The following sequence diagram is how the `RetrieveFreeTimes` command operation works.



4.4.3 Reasons for current implementation

The user will be able to view the options provided through the response box. This will allow the user to better view the data instead of an AlertBox where the data is static. The ability to select previous free time found through options will reduce the possibility where human error generates incorrect input.

4.4.4 Design Considerations

Aspect: Display of the selected free time slots to the user through a Response Box and self populated
Pros: Allows ensures double validation from the user. Example, Command Box and Response Box this ensures confirmation and prevents wrong entry from human error
Cons: The user may accidentally edit the data provided in the Command Box

4.5 Show Previous Feature

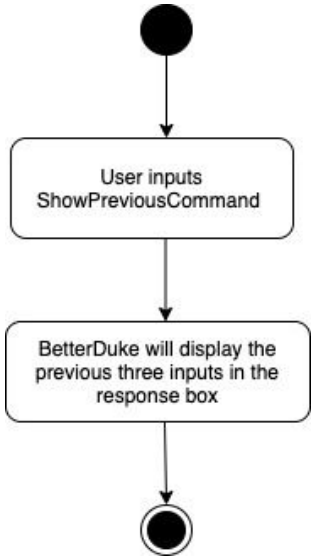
4.5.1 Implementation

The Show Previous feature mechanism is facilitated by `ShowPreviousCommand`. The `ShowPreviousCommand` extends `Command` class with an abstract execution method specific to `ShowPreviousCommand`.

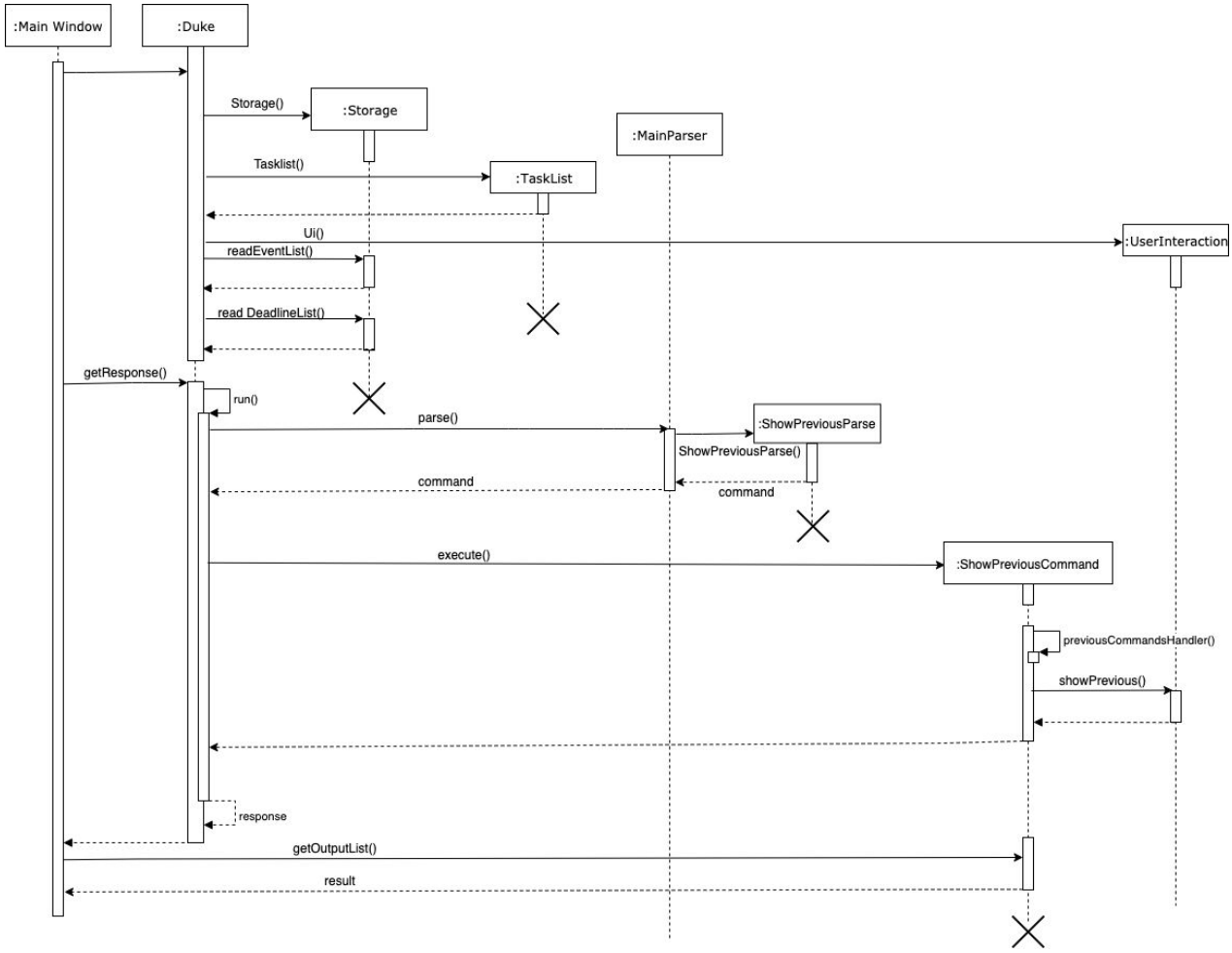
- `ShowPreviousCommand#previousCommandsHandler()` - Adds the required inputs that the user requested to see into the `outputList`.
- `ShowPreviousCommand#getOutputList()` - Generates the `previousCommandsList`.

Below is the implementation when `ShowPreviousCommand` is called.
Step 1: The user launches the application, BetterDuke and enters an input `show/previous 3`.
Step 2: BetterDuke will display the previous three inputs entered by the user in the response box.

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



The following sequence diagram is how the `ShowPreviousCommand` operation works when the user enters `show/previous` in the command line.



4.5.2 Reasons for current implementation

This implementation is useful as it brings more convenience for the user, especially when there is an error in the formatting of the commands which causes exceptions. This implementation allows user to see the previous inputs entered so that they can decide whether they want to re-type those commands in the command line.

4.5.3 Design Considerations

Aspect: How to display previous inputs when executes

- Alternative 1 (current choice): Shows previous inputs in a response box, allocated space in the GUI
    - Pros: User can easily refer to that response box to see the information presented to them. They will not have to remember the information and have fear of forgetting them.

    - Cons: It may be a little messy if all the information provided by the application are presented in that box.

- Alternative 2: Show previous commands in a pop up alert box
    - Pros: Easy to implement. Small and concise way to display information to the user
    - Cons: User may forget what the previous three commands are after closing the pop-up alert box.

Aspect: Data structure to support show previous commands

- Alternative 1 (current choice) :Use of array list to store all the user inputs
    - Pros: Easy to implement as there are frequent additions into the list. Additionally, array list is a variable length Collection class.
    - Cons: Significant memory is needed if there are many user inputs.

4.6 Retrieve Previous Feature

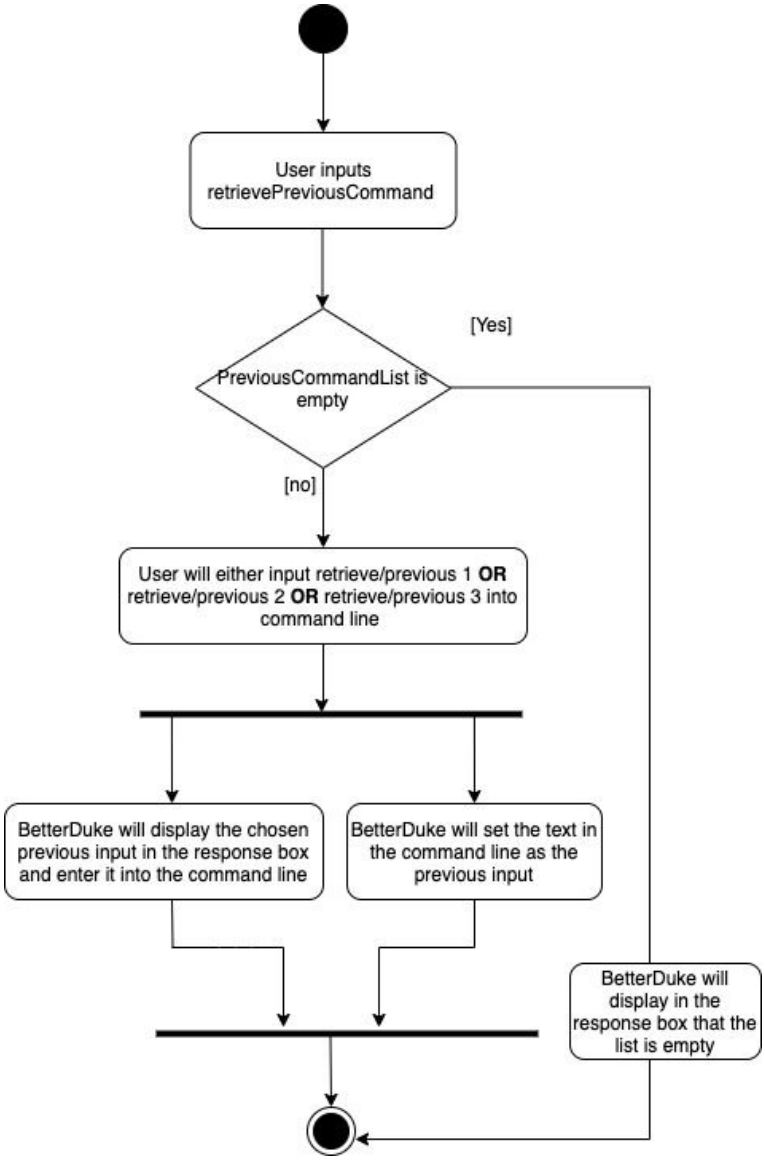
4.6.1 Implementation

The Retrieve Previous feature mechanism is facilitated by `RetrievePreviousCommand`. The `RetrievePreviousCommand` extends `Command` class with an abstract execution method specific to `RetrievePreviousCommand` if the user decides to use one of the previous inputs.

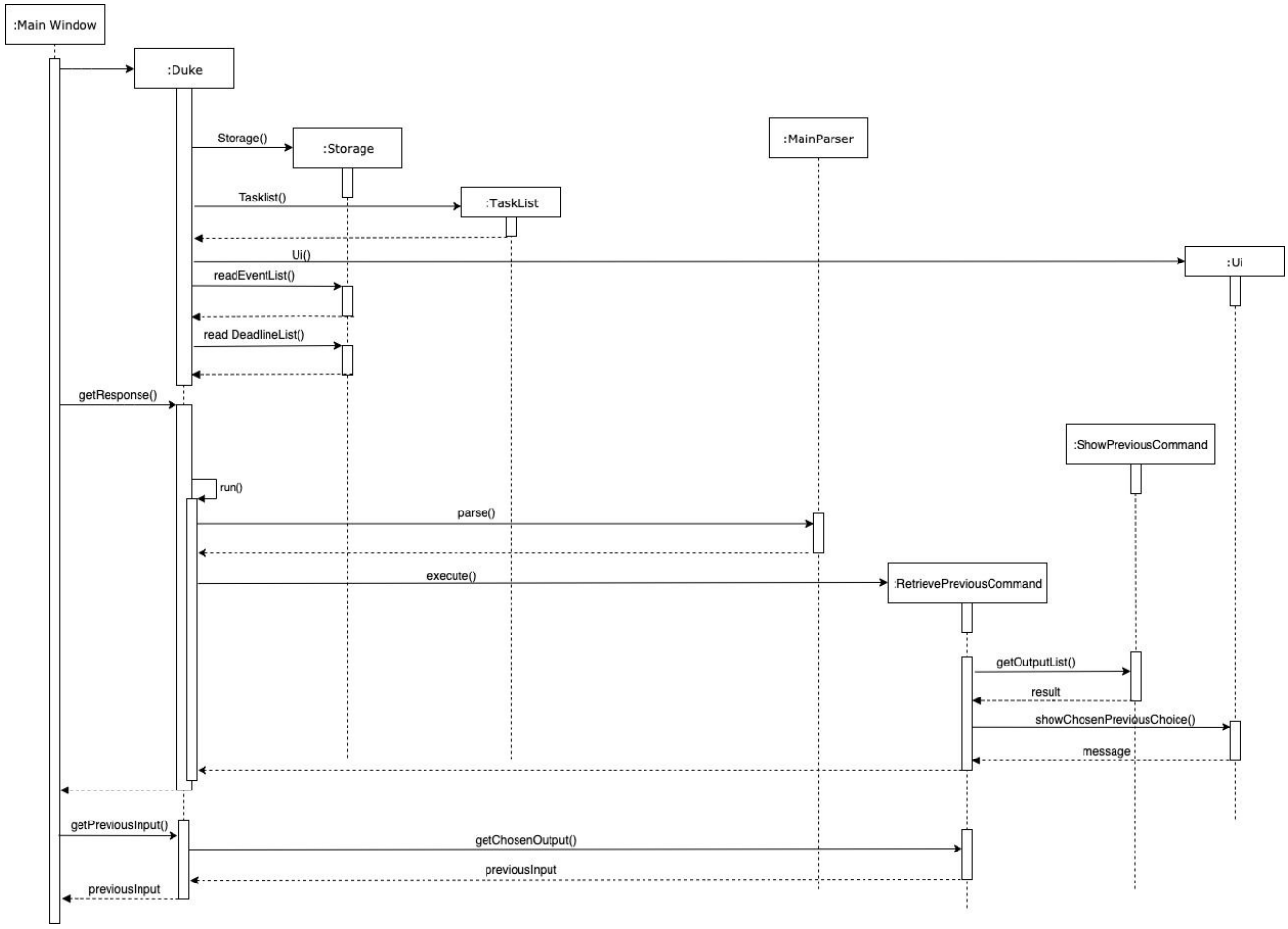
- `RetrievePreviousCommand#getChoosenOutput()` - Generates the `retrievedOutput`.

Below is the implementation when `RetrievePreviousCommand` is called.
Step 1: The user launches the application, BetterDuke and enters an input `retrieve/previous 3`.
Step 2: BetterDuke will display the previous input with index 3 from the `getOutputList()` method of `ShowPreviousCommand` in the response box.
Step 3: The application will automatically enter the input choice chosen by the user into the command line which is returned by `getChosenOutput()`.

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



The following sequence diagram is how the `RetrievePreviousThreeCommand` operation works when the user enters `retrieve/previous 3` in the command line.



4.6.2 Reasons for current implementation

This implementation is beneficial as it brings more convenience to the user. The user will not have to re-enter a lengthy command if he/she previously entered the command in the wrong format. By using this feature, the user can just retrieve the previous input and edit the incorrect command in the command line box.

4.6.3 Design Considerations

Aspect: Display of the previous command requested to the user

- Alternative 1 (current choice): Display of the requested previous command through a response box
  Pros: Allows reference for the user so that they would not type the wrong formatted command the next time.
  Cons: The response box may be filled with responses by BetterDuke and the user may get confused.

4.7. Show recommended workload feature

4.7.1 Implementation

The show recommended workload feature provides a recommended schedule based on user's current already inputted tasks which includes events and deadlines. The feature is facilitated by `ShowWorkloadCommand` which extends the Command abstract class.

- ShowWorkloadCommand#isValid() - Checks whether the selected week of events and deadlines corresponds to the workload week.
- ShowWorkloadCommand#findBestDay() - Finds the best day to place the deadlines into the workload week.
- ShowWorkloadCommand#findMinimum() - Finds the day which the least number of tasks scheduled.
- ShowWorkloadCommand#dayToInt() - Converts the day in a week to integer format.

Given below is a step by step sequence of what happens when user executes show recommended workload explained.

Step 1: The user keys **`show/workload,`** after command is parsed, `ShowWorkloadCommand` is created and will first obtain the current academic week from the date the user executes the command.

Step 2: ShowWorkloadCommand#execute() then retrieves the eventMap and deadlineMap from TaskList.

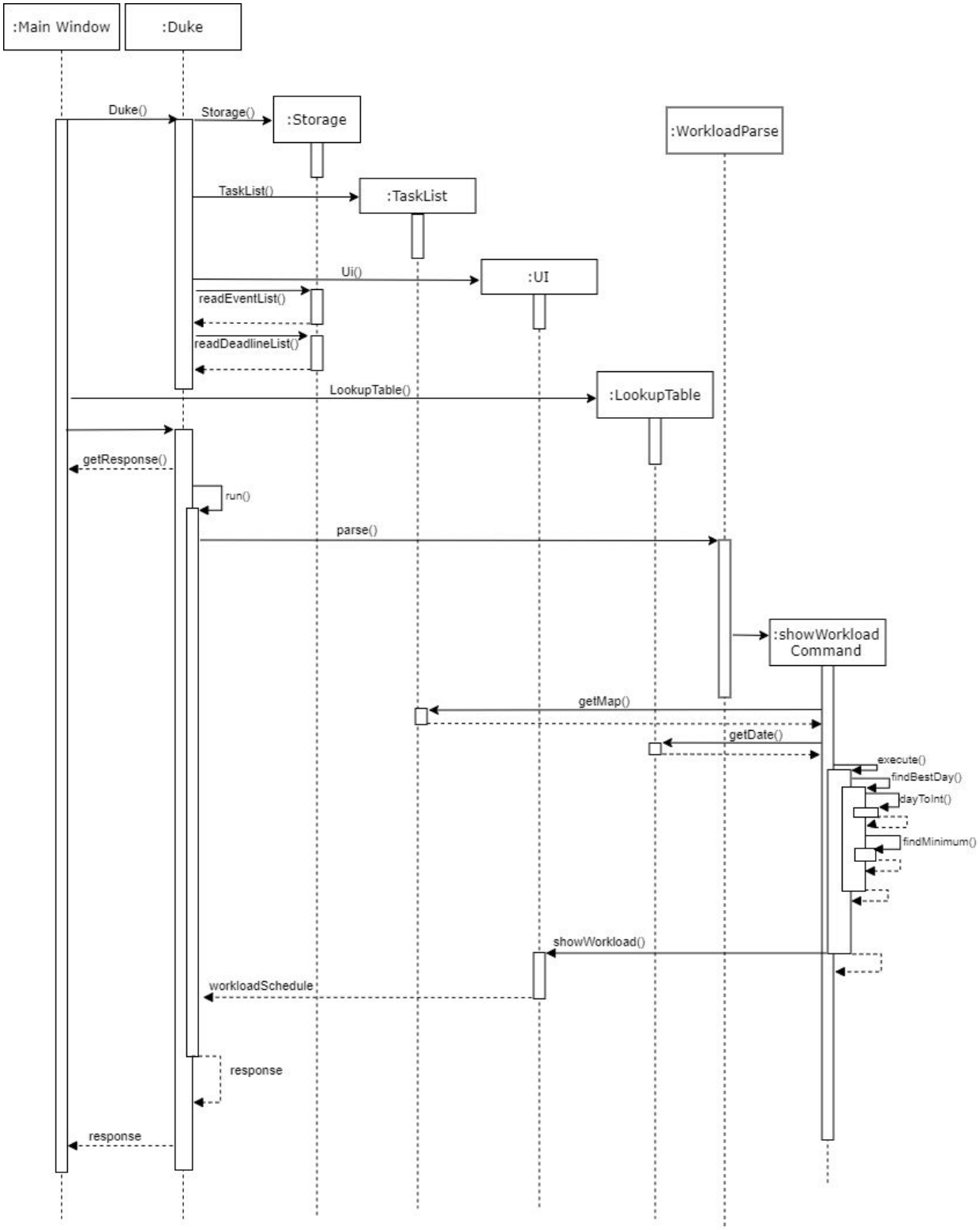
Step 3: ShowWorkloadCommand#execute() then extract events from the eventMap that are in the immediate coming week using ShowWorkloadCommand#isValid() and insert into a hashmap named workloadMap.

Step 4: Next, deadlines that are valid are also extracted using ShowWorkloadCommand#isValid(), and subsequently finds the best day to fit into the schedule using ShowWorkloadCommand#findBestDay().

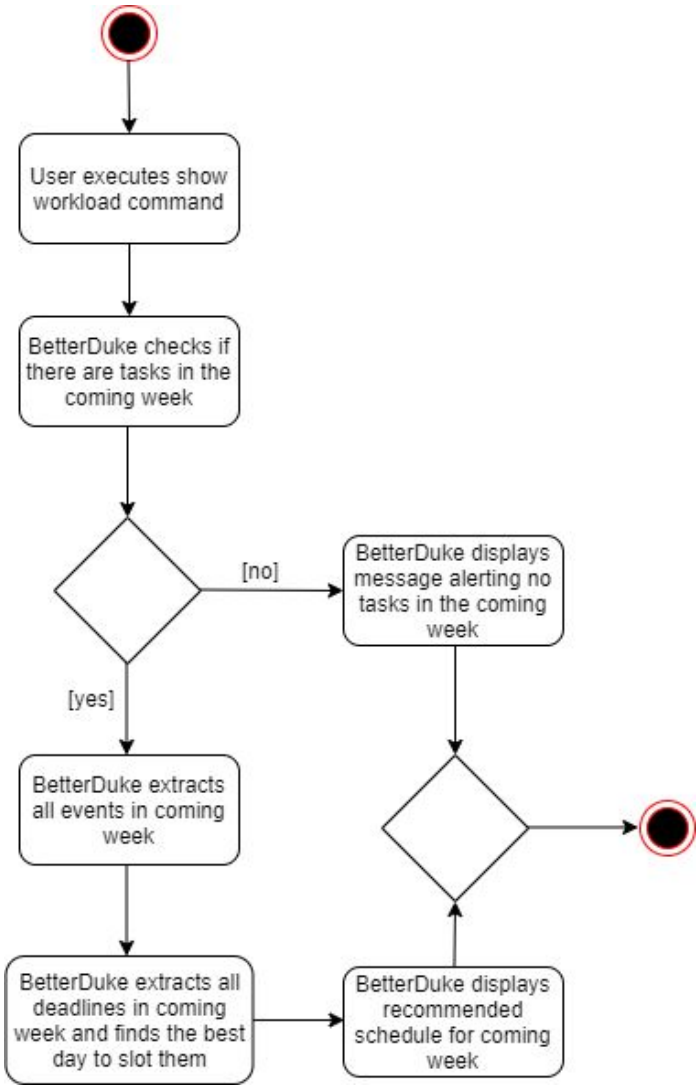
Step 5: ShowWorkloadCommand#findBestDay() calls ShowWorkloadCommand#dayToInt() as well as ShowWorkloadCommand#findMinimum() to execute fully and insert the valid deadlines into the workloadMap.

Step 6: BetterDuke then prints out the workloadMap to the user showing the recommended schedule for the immediate coming week.

The following sequence diagram shows how the show workload operation works:


:Main Window
:Duke
Duke()
Storage()
:Storage
:WorkloadParse
TaskList()
:TaskList
Ui()
:UI
readEventList()
readDeadlineList()
LookupTable()
:LookupTable
getResponse()
run()
parse()
:showWorkload Command
getMap()
getDate()
execute()
findBestDay()
dayToInt()
findMinimum()
showWorkload()
workloadSchedule
response
response

The following activity diagram summaries what happens when user executes command:



4.7.2 Design Considerations

Aspect: How show recommended workload executes

· Alternative 1 (current choice): Shows recommended workload in a response box.

  - Pros: Easy to implement, small and concise way to present schedule to user.

- Cons: Response box may be too small to see for the user.

- Alternative 2: Betterduke keeps all recommended schedules and display it to the user alongside normal schedule.
    - Pros: User can better toggle between normal and recommended schedules.
    - Cons: May have performance issues in terms of memory usage.

Aspect: Data structure to support show recommended workload commands

- Alternative 1 (current choice): Use a hashmap to store all tasks in the workload week.
    - Pros: Easy to store tasks based on common date as the key.
    - Cons: Hashmap is not thread-safe and there is potential of collision.
- Alternative 2: Use list to store all tasks in the workload week.
    - Pros: Easy to implement, able to add or remove tasks at a particular position.
    - Cons: Significant memory is required for a large list.

4.7.3 Future optimisation considered

- Show recommended workload feature only supports for the immediate coming week.
    - Optimisation: Provide support for any academic week the user inputs.
- Deadline tasks are added into recommended schedule based on first extracted from the deadline hashmap.
    - Optimisation: Slot the deadline tasks based on earliest deadline first.

4.8. Recurring feature

4.8.1 Implementation

The recurring feature provides a way to let users add or delete events that are happening weekly or biweekly in nature. The feature is facilitated by `RecurringCommand` which extends the Command abstract class.

- RecurringCommand#getNextWeekDate() - gets the date of 7 days later.
- RecurringCommand#getFollowingWeekDate() - gets the date of 14 days later.

Given below is a step by step sequence of what happens when user executes recurring feature explained.

Situation 1: user **adds a weekly recurring** event.

Step 1: The user keys **`recur/weekly ...,`** after command is parsed, `RecurringCommand` is created.

Step 2: The event is added on the start date, subsequently the date of next week is retrieved using RecurringCommand#getNextWeekDate() and replaced as the new start date.

Step 3: The new start date is checked if it has passed the given end date.

> 3.1: If the end date has been reached or passed, BetterDuke displays a message indicating the recurring event has been added between the given start and end date.
>
> 3.2: If the end date has not been reached or passed, the event is added on the new start date and the process repeats from Step 2.

For **adding biweekly recurring** event:

The above steps remain the same except the date of following week is retrieved using RecurringCommand#getFollowingWeekDate() and replaced as the new start date.

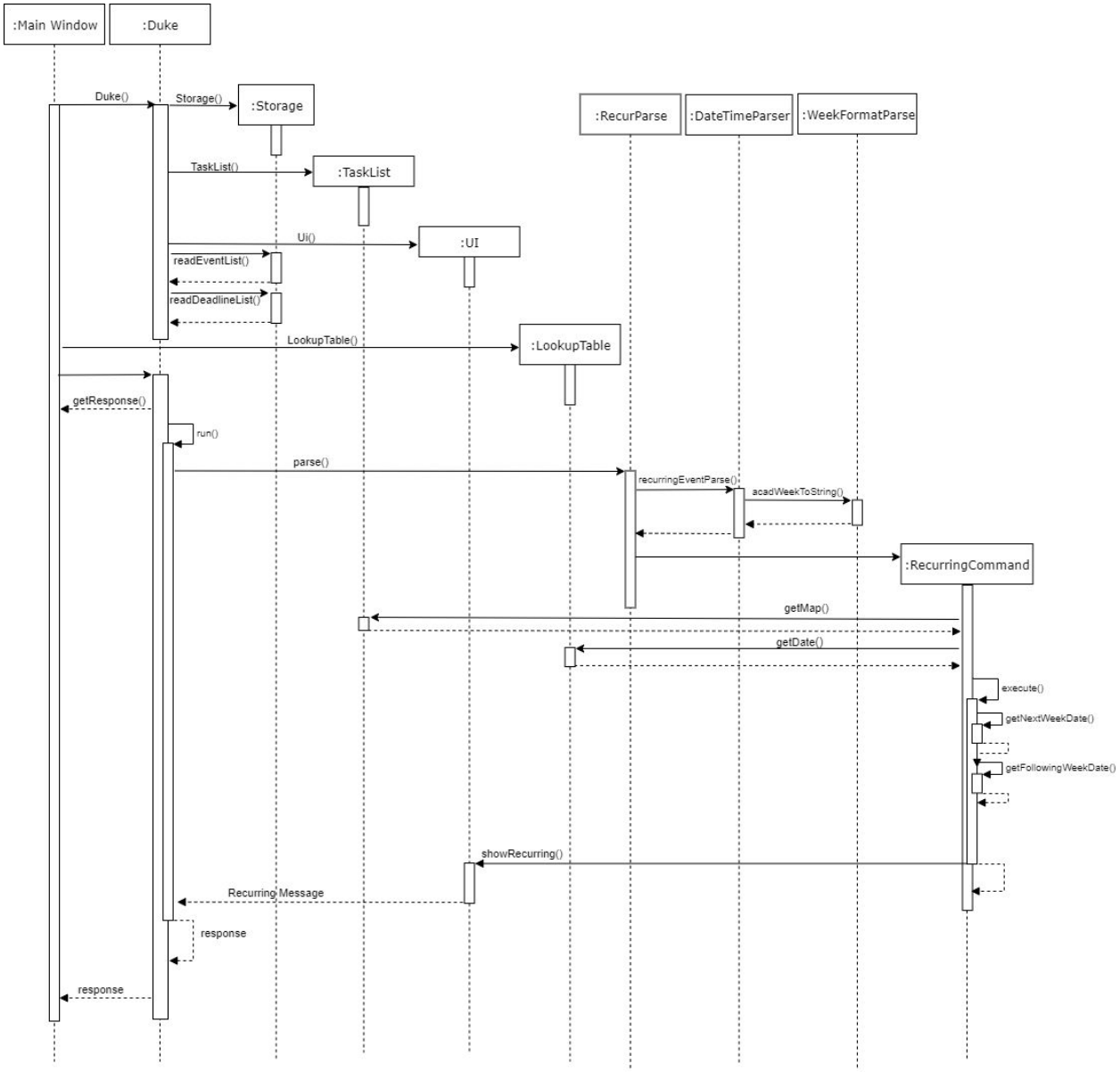
For **removing weekly recurring** event:

The above steps remain the same except event is removed.

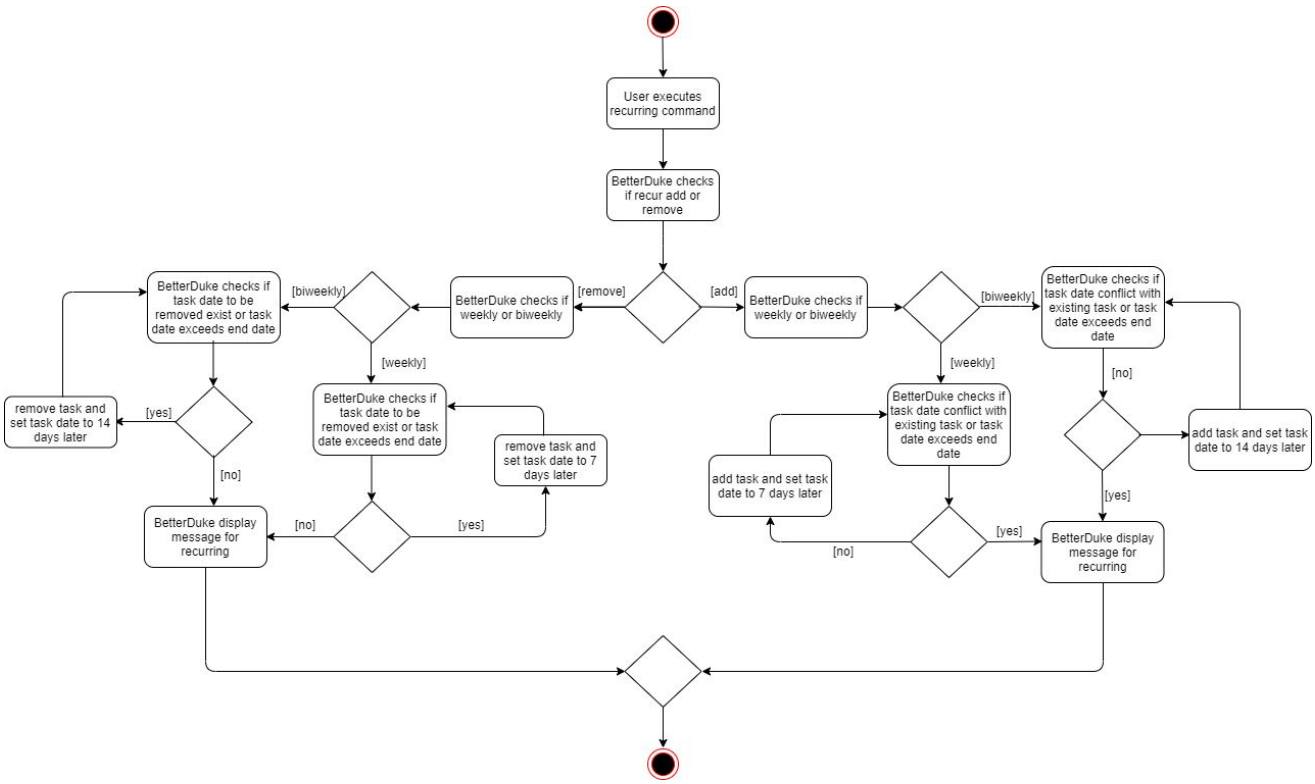
For **removing biweekly recurring** event:

The above steps remain the same except event is removed and RecurringCommand#getFollowingWeekDate() is used to replace as the new start date.

The following sequence diagram shows how the recurring event operation works:



The following activity diagram summaries what happens when user executes command:



4.8.2 Design Considerations

Aspect: How recurring command executes

- Alternative 1 (current choice): Recurring command adds or removes tasks starting from the given start date to the end date at a given time period.
    - Pros: Easy way to add recurring tasks by just increasing the dates.
    - Cons: Recurring command has to iterate through from the start to date end and checking each time if there is a conflict in events, if there is a conflict, memory is wasted in adding the earlier previous events.
- Alternative 2: Betterduke checks all events from the given start and end date and check for conflicting events before executing adding of recurring task.
    - Pros: Preferred method to check before execution.
    - Cons: May have performance issues in terms of memory usage.

4.8.3 Future optimisation considered

- Recurring command only supports adding and removing events.
    - Optimisation: Provide support for setting recurring events to completion.

Appendix A: Prioritised user stories

Status: Completed, To be completed, To be modified, KIV
Priorities: High (must have) - ***,Medium (nice to have) - **,Low (unlikely to have) - *

Index	Priority...	As a/an...	I want to...	So that I...	Status
1	***	Beginner user	View tasks based on type	Can know how many tasks I have for each type	Completed
2	***	New user	View the list of tasks for the current week	Can know the plans for the week	Completed
3	***	Beginner user	Set to-do reminders	Can be reminded of my daily tasks.	Completed
4	***	Advanced user	Strike off tasks and track my progress of the week	Can ensure I am on schedule	Completed
5	***	New user	Show upcoming tasks within a week	Can be better prepared for upcoming tasks.	Completed
6	***	Advanced User	Find tasks via keyword	Can filter tasks	Completed
7	***	Advanced user	Snooze tasks	Can reschedule the dates for tasks if it cannot be done within deadline	To be modified

8	***	Beginner user	Find free slots	Can plan my schedule with little to no clash	Completed
9	***	All user	Avoid duplicate task	Will not see unnecessary data	Completed
10	***	Beginner user	View all due tasks	Can easily keep track of upcoming deadlines	Completed
11	***	Beginner user	View schedule by module	Can view tasks by classification	To be completed
12	***	Advanced user	Input recurring activities	Can avoid adding the same activity again	Completed
13	***	Advanced user	Input multiple dates for a task	Can refer to the dates in the list until i can confirm the timing	To be modified
14	***	Advanced user	Set a range (ie between 15/01/2019 and 25/01/2019) for todo tasks	Can set reminders on the start and end date	To be modified
15	***	Beginner user	Enter my tasks according to academic calender	Can smoothly organize all my tasks	Completed
16	***	Advanced User	Be able to set a fixed duration for a task without entering start time	so that I do not have to search for a time to allocate my task.	To be modified

17	**	Advanced user	Use shorter versions of a command	Can type a command faster	KIV
18	**	Beginner user	Have an URGENT reminder	Can be reminded of tasks that have been left undone for 3 days	Completed
19	**	All users	Get quote of the day	Will be motivated to do my tasks	Completed
20	**	Advanced user	Change to font size of the display	Customize font size to my liking	KIV
21	**	Beginner user	Enter the timing for the activities	Can know the plans for my day	Completed
22	***	New User	Read help function	Can get answers to my questions quickly.	Completed
23	**	Advanced user	Track the task completion rate	Can be motivated to complete more tasks.	KIV
24	**	Beginner user	Label ‘IMPORTANT’ on task	Can show the importance of the task.	KIV
25	*	Advanced user	Sync the reminders to my computer	Can get timely notifications	Completed

26	**	Advanced user	Rank events by level of importance	Can plan my day more efficiently	To be completed
27	*	Beginner user	View the tasks within a range of dates	Can see all tasks in a period	Completed
28	*	Advanced user	Create a reminder with alarm	Can re-emphasize the reminder	KIV
29	*	Advanced User	Customise theme of the application	Can beautify the application with my creativity	To be completed
30	***	Advanced User	Get recommended weekly workload based on my list of current task	Can better manage my schedule	To be completed
31	***	Advanced User	Show previous inputs	Can know which input is typed incorrectly	Completed
32	***	Advanced User	Retrieve previous input	Can change the incorrectly typed input easily instead of retyping it	Completed

Appendix B: Use cases

(For all use cases below, the **System** is Betterduke and the **Actor** is the user, unless specified otherwise)

Use case: Add task
MSS

1. User requests to add a specific task
2. Betterduke adds the task
3. User requests to show schedule
4. Betterduke shows the schedule of updated tasks
   Use case ends.

Extensions

- 1a. The given add command is in invalid format
  Valid format:
  EVENT: `add/e` module_code description `/at` date `/from` time `/to` time
  DEADLINE: `add/d` module_code description `/by` date time
  - Betterduke shows an error message.
    Use case resumes at step 1.
- 2a. The given task's date and time coincides with existing tasks
  - Betterduke detects anomaly and shows anomaly message
    Use case resumes at step 1.

Use case: Delete task
MSS

1. User requests to show schedule
2. Betterduke shows schedule of tasks
3. User requests to delete a specific task in the schedule
4. Betterduke deletes the task
5. User requests to show schedule
6. Betterduke shows the schedule of updated tasks
   Use case ends.

Extensions

- 1a. The given delete command is in invalid format
  Valid format:
  EVENT: `delete/e` module_code description `/at` date `/from` time `/to` time
  DEADLINE: `delete/d` module_code description `/by` date time
  - 1a1. Betterduke shows an error message.

Use case resumes at step 2.

Use case: Show Previous
MSS

1. User requests to show previous inputs that starts with `add/d`
2. BetterDuke shows all previous inputs starting with `add/d`

Extensions

- 1a. The given command is invalid
  Valid format:
  `show/previous <command type>`. In this case, `command type` is `add/d`.
  `show/previous x`, where `x` is an integer
  - Betterduke shows an error message.
    Use case resumes at step 2.

Use case: Retrieve Previous
MSS

1. User requests to retrieve previous input with index 2
2. BetterDuke shows the chosen previous input and sets the text in the command line as the chosen previous input

Extensions

- 1a. The given command is invalid
  Valid format:
  `retrieve/previous x`, where `x` is an integer
  - Betterduke shows an error message.
    Use case resumes at step 2.

Use case: Set reminder
MSS

1. User requests to set reminder
2. Betterduke sets reminder
3. User waits for reminder
4. Betterduke displays reminder for user

Extensions

- 1a. The given command is invalid
  Valid format:
  `remind/set` module_code description `/by` date time `/to` date time
  - Betterduke shows error message

Use case resumes at step 1.

Use case: Remove reminder
MSS

1. User requests to remove reminder
2. Betterduke removes reminder

Extensions

- 1a. The given command is invalid
  Valid format:
  `remind/rm` module_code description `/by` date time `/to` date time
  - Betterduke shows error message
    Use case resumes at step 1.

Use case: Finding free time slot
MSS

1. User requests to show schedule
2. Betterduke shows schedule of tasks
3. User requests to find free time slot
4. Betterduke shows nearest free time slot available
5. User selects a time slot
6. Betterduke generates selected slot in the command line

Extensions

- 1a. The given command is invalid
  Valid format: `Available 'x' hours, where 'x' is a digit`
  - Betterduke shows error message
    Use case resumes at step 2.

Use case: Show recommended weekly workload
MSS

1. User requests to show schedule
2. Betterduke shows schedule of tasks
3. User requests to show recommended weekly workload
4. Betterduke shows recommended workload for next week

Extensions

- 1a. The given command is invalid
  Valid format:

`show/workload`

- Betterduke shows error message
  Use case resumes at step 2.

Use case: Filter with keyword

MSS

1. User requests to show schedule
2. Betterduke shows schedule of tasks
3. User requests to find tasks based on keyword
4. Betterduke shows all tasks found based on keyword

Extensions

- 1a. The given command is invalid
  Valid format:
  `show/filter keyword`
  - Betterduke shows error message
    Use case resumes at step 2.

Appendix C: Non-functional requirements (NFR)

4.1. Performance requirement: System should respond within 1sec
For any responses > 1 sec, the System should provide feedback indicating when it expects to be done, otherwise users will not know what to expect.

4.2. Quality requirement: The system should be usable by a beginner who has never used an organizer application before.

4.3. Process requirements: the project is expected to adhere to a schedule that delivers a feature set every one month.

4.4. Environment requirements: Jar file will be packaged and uploaded by both Windows and Mac to be compatible in both technical environment.

4.5 Technical requirements: The system should work on both 32-bit and 64-bit environments.

4.6 Notes about project scope: The product does not support physical printing.

Appendix D: Instructions for Manual Testing

Given below is the instructions to test the app manually.

D.1 Adding an event/deadline

Deadlines

1) Test Case: `add/d CS2113T Test deadline /by week 12 mon 1300`
   Expected: Test deadline will be added to events list.
   Test deadline populated into deadline table in GUI

Events

2) Test Case: `add/e CS2113T Test event /at week 12 mon /from 1200 /to 1300`
   Expected: Test event will be added to events list.
   Test event populated into calendar in GUI

Incorrect commands to try

1) Test Case: `add/e CS2113T Test event at week 12 mon from 1200 to 1300`
   Expected: Test event not added. Incorrect format error shown in response bar.

2) Test Case: `add/e Test event /at x /from 1200 /to 1300` (where x is a date that does not fall in the academic calendar
   Expected: Test event not added. Incorrect format error shown in response bar.

3) Test Case: `add/e` or `add/d`
   Expected: Incorrect format will be shown in response bar.

D.2 Mark an event or deadline as done

Deadlines

1) Test Case: `done/d CS2113T Test deadline /by week 12 mon 1300`
   Expected: Test deadline will be added to events list.
   Test deadline populated into deadline table in GUI

Events

2) Test Case: `done/e CS2113T Test event /at week 12 mon /from 1200 /to 1300`
   Expected: Test event will be added to events list.
   Test event populated into calendar in GUI

   Incorrect commands to try

1) Test Case: `done/e CS2113T Test event at week 12 mon from 1200 to 1300`
   Expected: Test event not added. Incorrect format error shown in response bar.

2) Test Case:`done/e Test event /at x /from 1200 /to 1300` (where x is a date that does not fall in the academic calendar
   Expected: Test event not added. Incorrect format error shown in response bar.

3) Test Case: `done/e` or `done/d`
   Expected: Incorrect format will be shown in response bar.

D.3 Deleting an event/deadline

Deadlines

1) Test Case: `delete/d CS2113T Test deadline /by week 12 mon 1300`
   Expected: Test deadline will be added to events list.
   Test deadline populated into deadline table in GUI

Events

2) Test Case: `delete/e CS2113T Test event /at week 12 mon /from 1200 /to 1300`
   Expected: Test event will be added to events list.
   Test event populated into calendar in GUI

Incorrect commands to try

1) Test Case: `delete/e CS2113T Test event at week 12 mon from 1200 to 1300`
   Expected: Test event not added. Incorrect format error shown in response bar.

2) Test Case:`delete/e Test event /at x /from 1200 /to 1300` (where x is a date that does not fall in the academic calendar
   Expected: Test event not added. Incorrect format error shown in response bar.

3) Test Case: `delete/e` or `delete/d`
   Expected: Incorrect format will be shown in response bar.

D.4 Filtering by keyword

1) Test Case: `show/filter key` (If the keyword is present)
   Expected:Tasks with keyword will be shown in the response bar
2) Test Case: `show/filter key` (If the keyword is not present)
   Expected:Error message indicating no such task with keyword found will be shown in the response box.

Incorrect commands to try

1) Test Case: `show/filter` , `show filter`, `show/filter "space"`
   `Expected:` Error message indicating incorrect format will be shown in the response box.

D.5 Find free time

1) Test Case: `find/time 5 hours`
   Expected: Period blocks with 5 hours will be shown in the response bar

Incorrect commands to try

1) Test Case: `find/time 5`
   Expected: Error message indicating incorrect input will be shown in the response box.
   Test Case: `find/time 0 hours`, `find/time 17 hours`
   Expected: Error message indicating incorrect duration will be shown in the response box.

D.6 Retrieve free time found

1) Test Case: `retrieve/time 5` (If find free time not invoked)
   Expected: Error messaging indicating find free time command not invoked
2) Test Case: `retrieve/time 5` (If find free time invoked)
   Expected: Selected option will be shown in the response box and the command line will populate the option selected in event command format

Incorrect commands to try

1) Test Case: `retrieve/time 0` , `retrieve/time 6`
   Expected: Error message indicating incorrect option will be shown in the response box.

D.7 Show Previous

1) Test Case: `show/previous 5` (if user inputs 6 times or more)
   Expected: The response bar will show the previous 5 inputs entered by the user.

Incorrect commands to try

1) Test Case: `show/previous`
   Expected: Error message indicating incorrect format will be shown in the response box.

2) Test Case: `show/previous 5` (if user inputs less than 6 times)
   Expected: Error message indicating that there are lesser number of previous inputs than what is requested.

D.8 Retrieve Previous

1) Test Case: `retrieve/previous 5` (if the showPreviousList has 5 items)
   Expected: The response bar will display the fifth previous input from the showPreviousList and the command line will display this input.

Incorrect commands to try

1) Test Case: `retrieve/previous`
   Expected: Error message indicating incorrect format will be shown in the response box.

2) Test Case: `retrieve/previous 5` (if the showPreviousList has less than 5 items)
   Expected: Error message indicating that there are lesser number of previous inputs to choose from than what is requested.

D.9 Recur

Add weekly recurring events:

1) Test case: `recur/weekly CS2113T Test event /start week 13 mon /to exam week mon /from 1300 /to 1400`
   Expected: Test event will be added every week from week 13 to exam week inclusive into the list and populated into GUI.

Remove weekly recurring events:

2) Test case: `recur/rmweekly CS2113T Test event /start week 13 mon /to exam week mon /from 1300 /to 1400`
   Expected: Test event will be removed every week from week 13 to exam week inclusive from the list and the GUI.

Incorrect commands to try:

1) Test case: `recur/weekly CS2113T Test event start week 13 mon to week 9 mon from 1300 to 1400`
   Expected: recurring event not added. Error message indicating invalid format will be shown in the response box.
2) Test case: `recur/weekly CS2113T Test event /start week 13 mon /to exam week mon /from 1500 /to 1000`
   Expected: recurring event not added. Error message indicating invalid time format will be shown in the response box.
3) Test case: `recur/weekly CS2113T Test event /start week 13 mon /to week 9 mon /from 1300 /to 1400`
   Expected: recurring event not added. Error message indicating invalid time format will be shown in the response box.

D.10 Remind

Set reminders:

1) Test case: `remind/set CS2113T Test deadline /by week 12 mon 1300 /on week 12 mon 1200`
   Expected: A reminder will be set for CS2113T Test deadline. Reminder set message will be shown in the response box. A notification will then be shown on week 12 mon 1200.

Remove reminders:

2) Test case: `remind/rm CS2113T Test deadline /by week 12 mon 1300 /on week 12 mon 1200`
   Expected: Reminder that was set for CS2113T Test deadline will be removed. Reminder removed message will be shown in the response box.

Incorrect commands to try:

1) Test case: `remind/set CS2113T Test deadline /by week 12 mon 1300 /on week 12 mon 1400`
   Expected: Reminder not set. Error message indicating reminder date and time is after Test deadline date and time will be shown in the response box.
2) Test case : `remind/rm CS2113T Test deadline /by week 12 mon 1300 /on week 12 mon 0600`

Expected: Reminder not removed. Error message indicating that there are no such reminders will be shown in the response box.

3) Test case: `remind/set CS2113T Test deadline by week 12 mon 1300 on week 12 mon 1200`
   Expected: Reminder not set. Error message indicating invalid format will be shown in the response box.

D.11 Week selection

1) Test Case: `show/week 5`, `show/week reading`
   Expected: The days event list will be populated with the events in the selected week

   Incorrect commands to try

1) Test Case: `show/week 13`
   Expected: Error message indicating incorrect week will be shown in the response box.
2) Test Case: `show/week`
   Expected: Error message indicating incorrect input will be shown in the response box.

Appendix F:FAQ

Q: Do i have to install the application?
A: No you can just run the [betterduke].exe file

Q: If I am unsure about the commands, how do i get help?
A: Just enter “help” into the command line and BetterDuke will show you a comprehensive guide on how to use our app

Q:How do I find specific tasks based on a keyword
A: You can make use of the `filter` function to look for such tasks.

Q: Can this application be run on different OS?
A: Yes, it can be run on Mac OS and Windows OS.

Q: Do I have to edit the data files?
A: No, please try to not modify the data files. However, if you choose to do so, do it at your own discretion but at a risk of losing your data.