

# Jiang Yuxin - Project Portfolio

## PROJECT: SugarMummy

---

### Overview

SugarMummy is a desktop application used to manage a type-II diabetic lifestyle. The user interacts with it using a CLI, and it has a GUI created with JavaFx. It is written in Java and has about 24 kLOC.

### Summary of contributions

¥ Major enhancement: added a calendar system with real-time reminder

- ! What it does: allows the user to add calendar entries(reminder and event) and automatically generate real-time reminders.
- ! Justification: This feature improves the product significantly because a diabetic patients could be busy with their work or study already and they may need to keep track of important events, such as appointment with doctor, buying medicine regularly and be reminded at certain time so they do not forget, such as injecting insulin.
- ! Highlights: This enhancement deals with date and time and scheduling tasks using the java multithreaded API [ScheduledExecutorService](#). It provides both dynamic and static viewing of calendar entries. When the app is running concurrently, reminders are automatically shown in the side pane according to the time set by the user without interrupting user typing other commands. Reminder list can be updated at times during the running time of the app, it requires the scheduler to update scheduled tasks. Also reminders can recur daily or weekly so scheduler needs to adapt those reminders to an one-time reminder on certain date. Moreover, to avoid potential duplicate reminder shown in the side pane, the calendar checks any reminders that overlap with each other.

¥ Minor enhancement:

- ! added a calendar command that allows the user to view monthly calendar and a list of calendar entries on a day or each day in a month or in a week.
- ! added a pane for showing reminders that the user might miss.

¥ Code contributed: [[View RepoSense](#)]

¥ Other contributions:

- ! Project management:
  - " Wrote additional tests for existing features to increase coverage. (Pull request: [#21](#))
  - " Clean up unused code. (Pull request: [#96](#))
  - " Update UG,DG and refactor package. (Pull request: [#109](#))
- ! Documentation:

" Update intro, quickstart and summary of user guide. (Pull request: [#19](#))

" Draft about us page. (Pull request: [#21](#))

## Contributions to the User Guide

*Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.*

### CALENDAR AND REMINDER

Command arguments for calendar and reminder:

¥ **d/DESCRIPTION** Description can take any values, and it should not be blank. If more than one are present, the last argument is taken.

¥ **dt/DATETIME** The format of date time is: **yyyy-mm-dd hh:mm** and the number should be valid. If more than one are present and the command only accepts one **dt/**, the last argument is taken. Otherwise, see below in **event** command for more details.

¥ **r/REPETITION** Repetition can only take value **once**, **daily** or **weekly**(case insensitive). If more than one are present, the last argument is taken.

¥ **td/TIME\_DURATION** The format of time duration is: **hh:mm** and it should not be blank. e.g. **01:30** represents 1 hour and 30 minutes. If more than one are present, the last argument is taken.

¥ **ym/YEAR\_MONTH** The format of year month is: **yyyy-mm** and the number should be valid. If more than one are present, the last argument is taken.

¥ **ymd/YEAR\_MONTH\_DAY** The format of year month day is: **yyyy-mm-dd** and the number should be valid. If more than one are present, the last argument is taken.

¥ **ymw/YEAR\_MONTH\_DAY** The format of year month day is: **yyyy-mm-dd** and the number should be valid(same as year month day). If more than one are present, the last argument is taken. See below in **calendar** command for more details if more than one of **ym/**, **ymd/**, **ymw/** are present.

#### Add a reminder: **reminder**

Format: **reminder d/DESCRIPTION dt/DATETIME [r/REPETITION]**

Add a reminder at a specific time and date. There is no repetition by default and user can specify a daily or weekly or no repetition optionally. If it is a repeated reminder, the date inputted is the starting date of the reminder(reminder with date time before the current date time is allowed). And it provides duplicate and conflicting checking as well as auto merging for reminders added. For example, **reminder d/insulin inject dt/2019-11-25 17:30 r/daily** adds a new reminder of insulin injection at 17:30 everyday from 2019 Nov 25.

¥ More on duplicate and conflicting check and auto merging

! To avoid duplicate reminders, the app will not accept reminders that already exist in the system(same description, date time and repetition). Similarly, reminders that are completely covered by an existing reminder will not be added again. For example, if the previous

reminder command is successful, `reminder d/insulin inject dt/2019-12-01 17:30 r/weekly` will not add the new reminder because it is completely covered (whenever the later one shows up, the previous one shows up at the same time with the same description). `reminder d/insulin inject dt/2019-12-01 12:00 r/daily` is not considered covered because there could be two injections one day.

! Moreover, new reminder that covers one or more existing reminders can be added and those existing ones will be removed automatically. For example, if `reminder d/insulin inject dt/2019-11-20 17:30 r/daily` is typed after the previous one, it replaces the insulin inject from 2019-11-25 because whenever the previous reminder shows up, the new reminder shows up with the same description.

! However, if the new reminder overlaps with existing ones but they cannot be merged, it will not be added. To illustrate, `reminder d/insulin inject dt/2019-11-18 17:30 r/weekly` is rejected because on 2019 Nov 18 it shows up at 17:30 alone. After 2019 Nov 20, this reminder will be a duplicate of the previous one every Monday.

#### ¥ Reminder list side pane

! On the right side of the app window, there is a pane for reminder list with the date showing on the top. The reminder list pane is split into two parts: a real-time reminder list on top of a probably missed reminder list.

! The real-time reminder list is empty if there is no reminder set at current time. If the app is open, the reminders will show up in the pane at the time set by the user. For example, if the user opens the app at 17:00 on 2019 Nov 30. A reminder of insulin injection will show up at 17:30 if the user has not closed the app.

! The probably missed reminder list includes all reminders that should exist in at the time between the starting of day and the time when the user opens the app. For example, if the user opens the app at 17:00 on 2019 Nov 30, it lists all reminders from 00:00 to 16:59 on that day. However, if the user adds a reminder on 2019 Nov 30 17:10 at 17:20 on that day, it will neither exist in the real-time reminder list pane nor the missed reminder list pane.

## Add an event: `event`

Format: `event d/DESCRIPTION dt/DATETIME [dt/DATETIME] [td/TIME_DURATION]`

Add an event with a starting time and an optional ending time(starting time before the current date time is accepted). There is no reminder for an event by default. User can also create a reminder for this event by specifying a time duration and a one time reminder will show up at the starting time of the event minus this time duration. For example, `event d/appointment dt/2019-12-20 14:00 dt/2019-12-20 15:00 td/01:30` adds an appointment event on 2019 Dec 20 from 14:00 to 15:00 to the calendar and a reminder for it at 12:30 on that day.

If only one `dt/` is present, it will be recognized as the starting date time. If there are more than one `dt/`, the first and second one are interpreted as starting and ending date time. Starting date time should come before ending date time. If they are equal, then ending date time is discarded. A duplicate event(same description and starting date time) will be rejected. However, events that overlap with existing ones are accepted with a message shown to the user. For example, `event d/appointment dt/2019-12-20 14:00 dt/2019-12-20 16:00` will be rejected after the previous event commands. `event d/test dt/2019-12-20 13:00 dt/2019-12-20 14:10` is accepted with a message

shown.

## View calendar entries: `calendar`

Format: `calendar` [`ym`/YEAR\_MONTH] [`ymw`/YEAR\_MONTH\_DAY] [`ymd`/YEAR\_MONTH\_DAY]

View all calendar entries or calendar entries in a month or in a week or on a day.

¥ If none of the three arguments is present, it will show all calendar entries added by the user.

¥ If one or more of the three arguments are present, it accepts the one with the smallest time unit.  
e.g. if `ymw` and `ymd` are both typed in, it only shows entries on the given date if the date is valid.

! `ymd`/ shows the monthly calendar of given date and below the calendar is a list of reminders and events on that day. Particularly, if no date is provided, it will show calendar entries on the current date.

! `ymw`/ shows the monthly calendar of given date and below the calendar is a list of reminders and events on each day in the week. `ymw`/ accepts a date also and the week picked is 7 days from Monday to Friday which include the given date. Particularly, if no date is provided, it will show calendar entries in the current week. For example, `calendar ymw/2019-12-03` will show a monthly calendar of 2019 December and calendar entries from Dec 2 to Dec 8.

! `ym`/ shows the monthly calendar of given month and below the calendar is a list of reminders and events on each day in the month. `ym`/ accepts a month. Particularly, if no date is provided, it will show calendar entries in the current week.

! For the monthly calendar shown, the subscript number beside each date is the number events and reminders on that day. And the current date is in a different color.

## Snooze reminder: `snooze` [`coming in v2.0`]

Format: `snooze` [`tp`/TIME\_PERIOD]

Silence the current reminder and activate it after a time period.

## Complete a task: `complete` [`coming in v2.0`]

Format: `complete` [`INDEX`] [`t`/TIME]

Mark all the reminders before now as completed if no argument provided. Otherwise, mark only those tasks with indices provided or before the specific time as completed.

## Set time zone: `timezone` [`coming in v2.0`]

Format: `timezone` `tz`/TIME\_ZONE [`t`/TIME] [`t`/TIME\_END]

Set the time zone of the application permanently or in any time interval(e.g. For travelling).

## Cancel an entry in calendar [`coming in v2.0`]

Format: `cancel` [`et`/ENTRYTYPE] [`d`/DESCRIPTION] [`dt`/DATETIME] [`dt`/DATETIME]

Delete an event that you do not want to keep track anymore.

# Contributions to the Developer Guide

*Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.*

## Calendar feature

### Implementation

#### Overview

The calendar feature is mainly supported by `Calendar` along with a `Scheduler`. `Calendar` stores internally a `calendarEntries` list, a `pastReminders` list and a `Scheduler`. It also handles checking for duplicate and overlapping calendar entries. Calendar entries consists of `Reminder`s` and `Event`s`. `calendarEntries` list is an unique list for all calendar entries added by the user. Aside from it, `pastReminders` list is dynamically determined by time. The `Scheduler`, which utilizes `java ScheduledExecutorService` is responsible for adding reminders at specific time to the `pastReminders` list and all reminders in this list is shown to the user. The real-time reminder works parallel with all other features in this app. Also `Scheduler` keeps track of the current date and the starting time of running the app.

Calendar implements the following operations:

- ¥ `Calendar#addCalendarEntry()` Adds a new calendar entry to the calendar.
- ¥ `Calendar#addPastReminder()` Adds a reminder to the past reminders list.
- ¥ `Calendar#getCalendarEntryList()` Gets a list of calendar entries.
- ¥ `Calendar#getPastReminderList()` Gets a list of past reminders.
- ¥ `Calendar#schedule()` Schedules a series of upcoming reminders.

These operations are exposed in the `Model` interface as `Model#addCalendarEntry()`, `Model#addPastReminder()`, `Model#getFilteredCalendarEntryList()`, `Model#getPastReminderList()` and `Model#schedule()` respectively.

#### Reminder and Event class

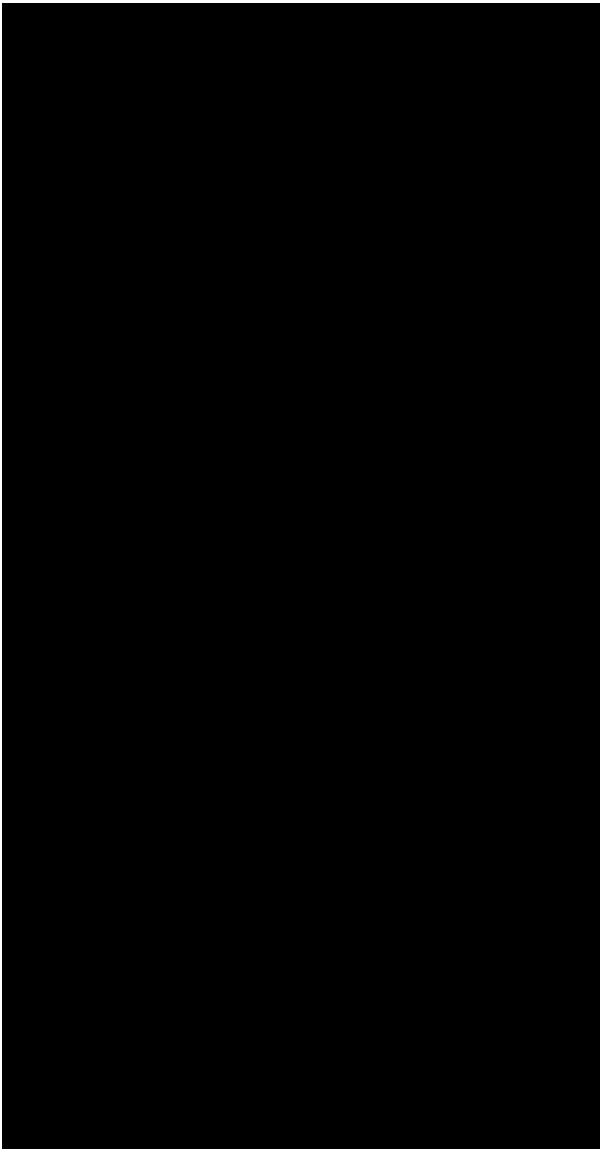
Basically, both `Reminder` and `Event` which extend from the abstract class `CalendarEntry` consist of a `Description` and a `DateTime` field.

For `Reminder`, the `DateTime` field represents the time of the reminder and the date from which the reminder starts. It has a field of `Repetition`, which is an enum class representing `Daily`, `Weekly` or `Once` repetition type of the reminder.

For `Event`, it has another optional `DateTime` field and an optional `Reminder` field. The compulsory `DateTime` field is the starting date time while the optional one is the ending date time. The `Reminder` is

for an auto reminder created by the app if the user requires in the command.

See the class diagram below for calendar related classes.



#### EventCommand and ReminderCommand class

To add events or reminders, the calendar system implemented `EventCommand` and `ReminderCommand`. Before adding an event, the calendar system will check whether any duplicate event exists and any events overlap with the new event by comparing the `DateTime` attributes.

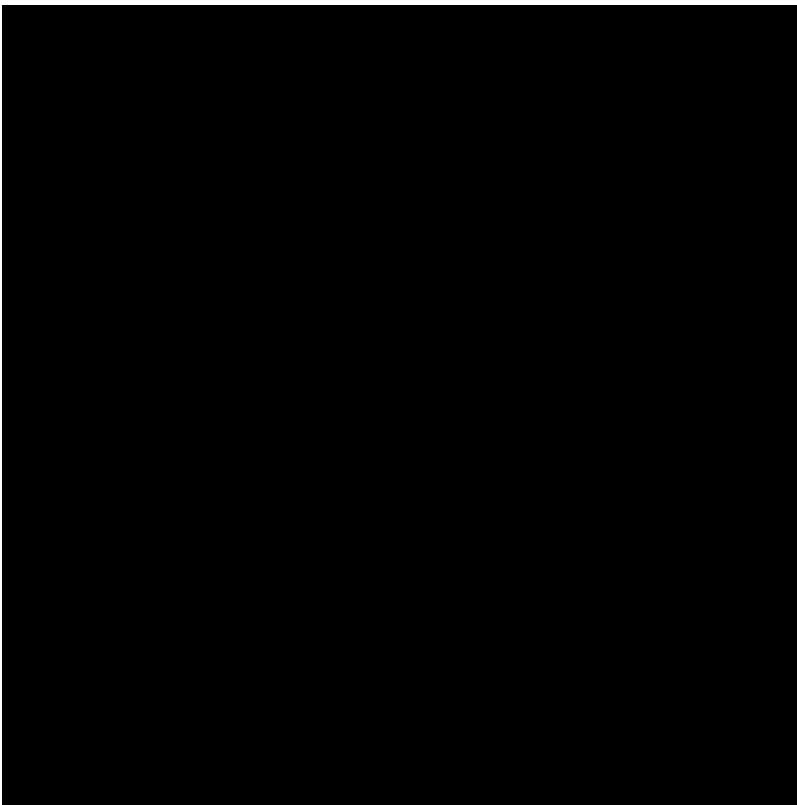
The execution of the `ReminderCommand` involves checking whether any duplicate reminder exists also. Due to probable recurrence of reminders, it also checks if any reminder can fully cover the new reminder by comparing `DateTime` and `Repetition`. In this case, the new reminder will not be added. Besides, if the new reminder can cover other reminders, new reminder will replace them. For those overlap but cannot be resolved, the system will not add the new reminder either.

#### Scheduler class

To show reminders at certain time, the `Scheduler` utilizes java `ScheduledExecutorService` to schedule a future task with a delay. It creates two inner classes implementing the `Runnable` interface for tasks. The `ReminderAdder` class represents a task of adding a reminder to the past reminder list. `Initializer`

class represents a task of initializing the scheduler at the beginning of a day.

The following activity diagram shows how scheduler works when it is called to schedule tasks:



#### ¥ Range of tasks scheduled

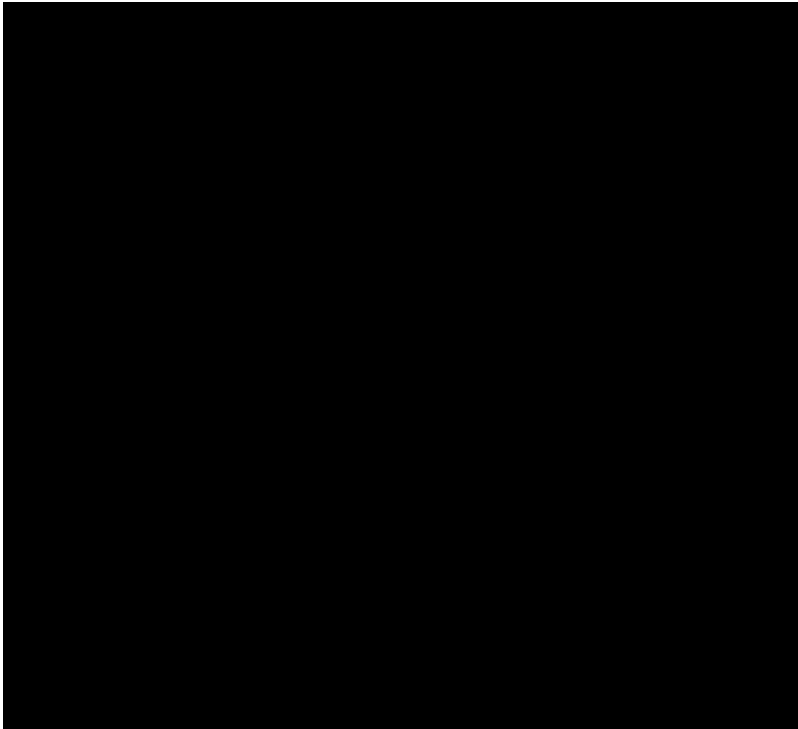
! To calculate the accurate delay time for each reminder, the scheduler keeps track of starting time for all the scheduled tasks and the delay is the time duration between this starting time and the reminder time. Apart from the starting time, there is a deadline for the scheduler which limits the time range for scheduled reminder tasks. So the scheduler only schedules tasks for reminders between the starting time and the deadline. In addition, before scheduling tasks, all reminders with the same time are grouped together by using a **TreeMap** to map from each time to a list of reminders so they can be added to the past reminder list together.

! After the app is launched, the **Scheduler** is called to initialize tasks. The starting time is set to be the starting date time and the deadline is set to be 23:59 on the same day. Thus only reminders on the current date is scheduled. During the app running, **Scheduler** can be called to reschedule tasks because of adding new reminders or removing reminders. This will adjust the starting time for scheduling to the current time while the deadline remains the same. Due to the limited number of threads and unknown number of upcoming reminders, the scheduler will cancel all the tasks that have not been executed and then schedule upcoming reminders that fall between the new starting time and the deadline. Each time any reminder being added or removed, the scheduler is triggered to reschedule tasks.

#### ¥ How to move on to next day

Besides scheduling tasks of adding reminders, the scheduler always schedules a task with **Initializer** class for initializing right after the current deadline, so that it can transfer smoothly to the next day if the app is open overnight. This initializer will set the deadline to be end of the next day, update date and schedule tasks.

The following activity diagram shows an example of how event command and scheduler work together:



#### Example usage scenario

Given below is an example usage scenario and how the calendar behaves at each step.

Step 1. The user launches the application for the first time on Dec 14 2019 09:00(local time). The `Calendar` will be initialized with the initial calendar state, which includes an empty calendar entry list and an empty past reminder list.

Step 2. The user executes `reminder d/insulin injection dt/2019-12-14 17:30 r/daily` command to add a new reminder of 'insulin inject' at 17:30 every day. The `reminder` command calls `Model #addCalendarEntry()`, causing the modified state of the calendar after the reminder command executes to be saved in the `calendarEntries` list. Subsequently, it calls `Model #schedule()` which forces the scheduler to update the upcoming reminders.

Step 3. The user executes `event d/meeting dt/2019-12-14 14:30 tp/00:30` command to add a new event with an auto reminder scheduled 30 minutes before the event. It calls `Model #addCalendarEntry()`, causing a new event as well as a new reminder saved in the `calendarEntries` list. Subsequently, it calls `Model #schedule()` which forces the scheduler to update the upcoming reminders.

#### NOTE

If an event or reminder command fails its execution, it will not call `Model #addCalendarEntry()`, so the calendar state will not be saved into the `calendarEntryList`.

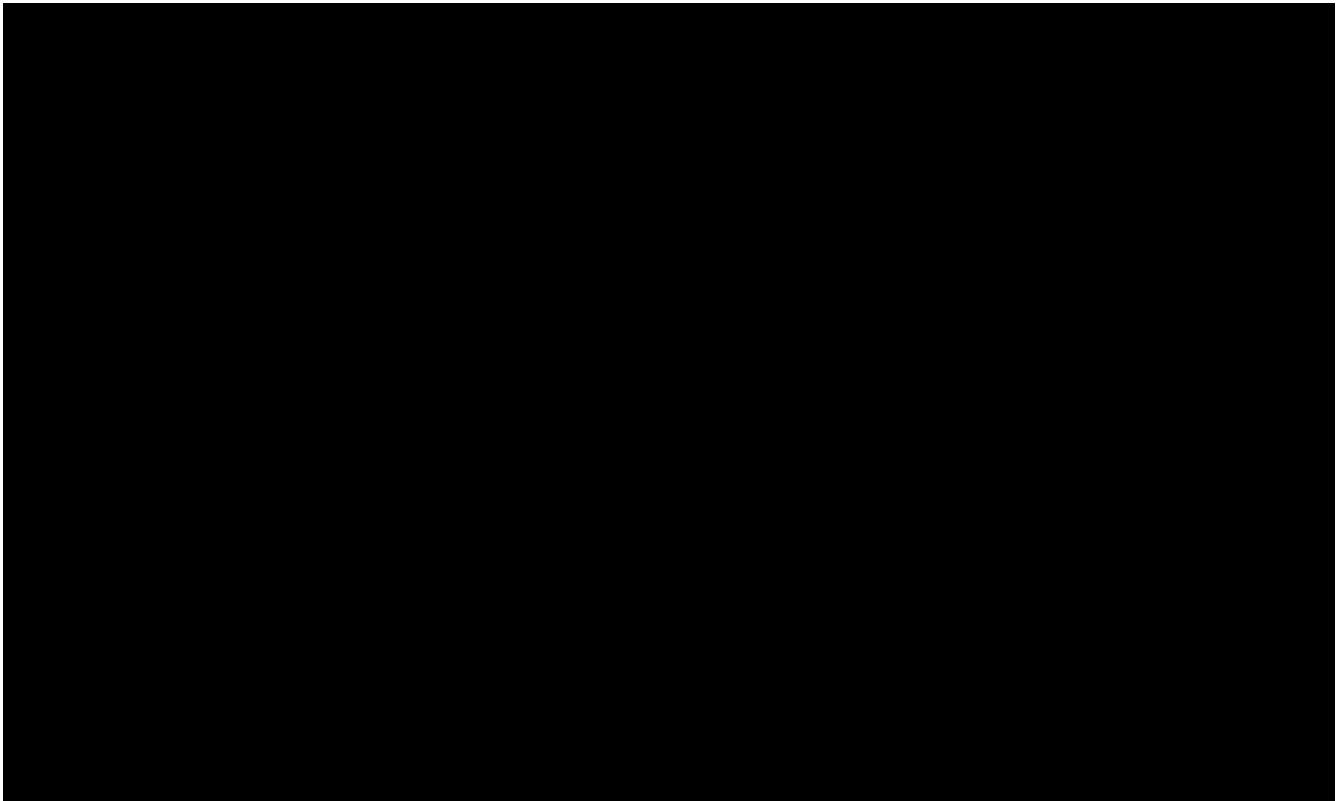
Step 4. At 14:00, a scheduled task is executed to call `Calendar #addPastReminder()` and it adds the dinner event reminder to the `pastReminders` list.

Step 5. At 17:30, a scheduled task is executed to call `Calendar #addPastReminder()` and it adds the



dinner event reminder to the `pastReminders` list.

The following sequence diagram shows how a single `reminder` command works:



#### CalendarCommand class

`CalendarCommand` is implemented for showing calendar entries and monthly calendar. It consists of a `YearMonth`, an optional `YearMonthDay`, `isShowingWeek` and `isShowingRaw` fields. The result of executing a `CalendarCommand` is a subclass `CalendarCommandResult` extending from `CommandResult`, which will be considered differently in the `MainWindow`. `MainWindow` will create a pane depending on the attributes of `CalendarCommandResult`.

## Design Considerations

Aspect: How scheduler updates upcoming reminders

- ¥ Alternative 1 (current choice): Cancels all scheduled reminders and reschedule according to the updated reminder entries.
  - ! Pros: Easy to implement.
  - ! Cons: May have to do duplicate work of scheduling. May have performance issues in terms of time.
- ¥ Alternative 2: Updates scheduled reminders according to the newly added reminder.
  - ! Pros: Will has less repeated work.
  - ! Cons: More work to do on deciding which tasks to cancel.

Aspect: Period of updating scheduler.

¥ Alternative 1 (current choice): Updates scheduler at 23:59(local time) every day.

! Pros: Good consistency.

! Cons: May have a large number of scheduled tasks which will not be executed before the application is closed.

¥ Alternative 2: Updates scheduler every hour.

! Pros: More flexible scheduling without concerning date and less scheduled tasks.

! Cons: May cause overhead due to frequently updating.

Aspect: Resolution of overlapping reminders

¥ Alternative 1 (current choice): Force to replace subset reminders with new superset reminders which fully cover existing reminders.

! Pros: Avoid duplicate reminders added which the user may not be aware of.

! Cons: May remove some reminders that the user was not intent to do.

¥ Alternative 2: Asks for user's permission before proceeding.

! Pros: Can avoid unintentionally reminders deleting.

! Cons: May cause some duplicate reminders.