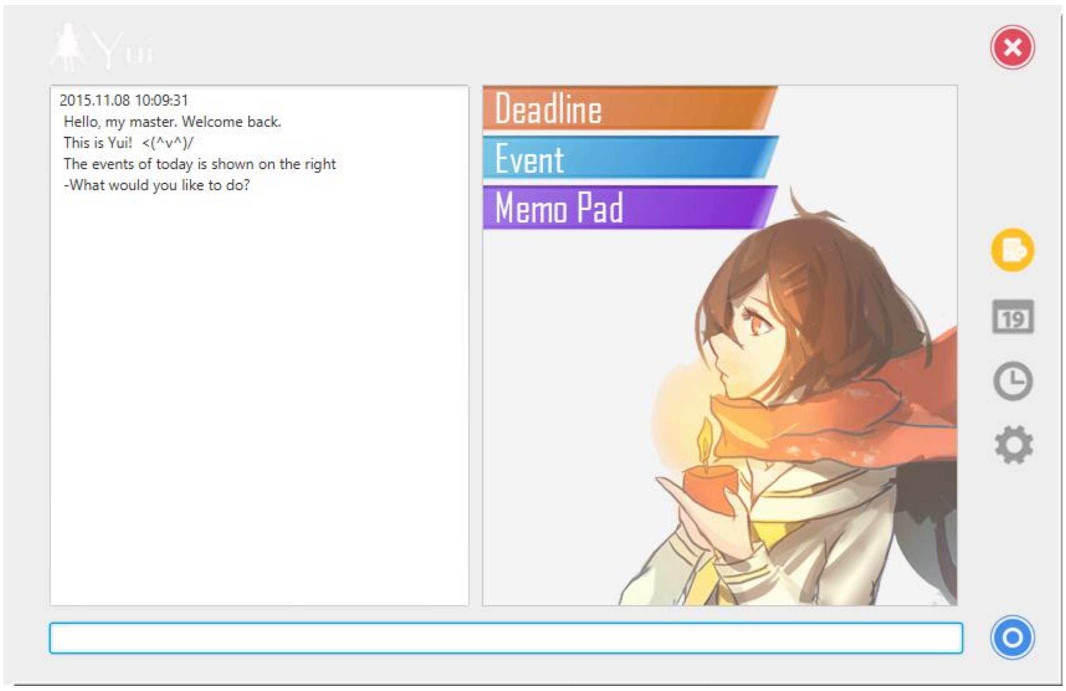
Yui Project Menu



2015.11.08 10:09:31
Hello, my master. Welcome back.
This is Yui! <(^v^)/
The events of today is shown on the right
-What would you like to do?

Supervisor: Karan Kamath Extra feature: Recurring Task & Good GUI

Liu Hanyang Team leader Git expert and Eclipse expert UI Designer	Justin Erh Documentation Code Quality	Mao Chenyang Testing Integration	Le Nguyen Scheduling and tracking Deliverable and deadline

Credit: We used JavaFx API and jintellitype API for our UI design. More information can be found in the UI component detail on page 32.

User Guide

Liu Hanyang (A0133992X)

Mao Chenyang (A0127142R)

Le Nguyen (A0127821J)

Justin Erh (A0125622N)

Contents

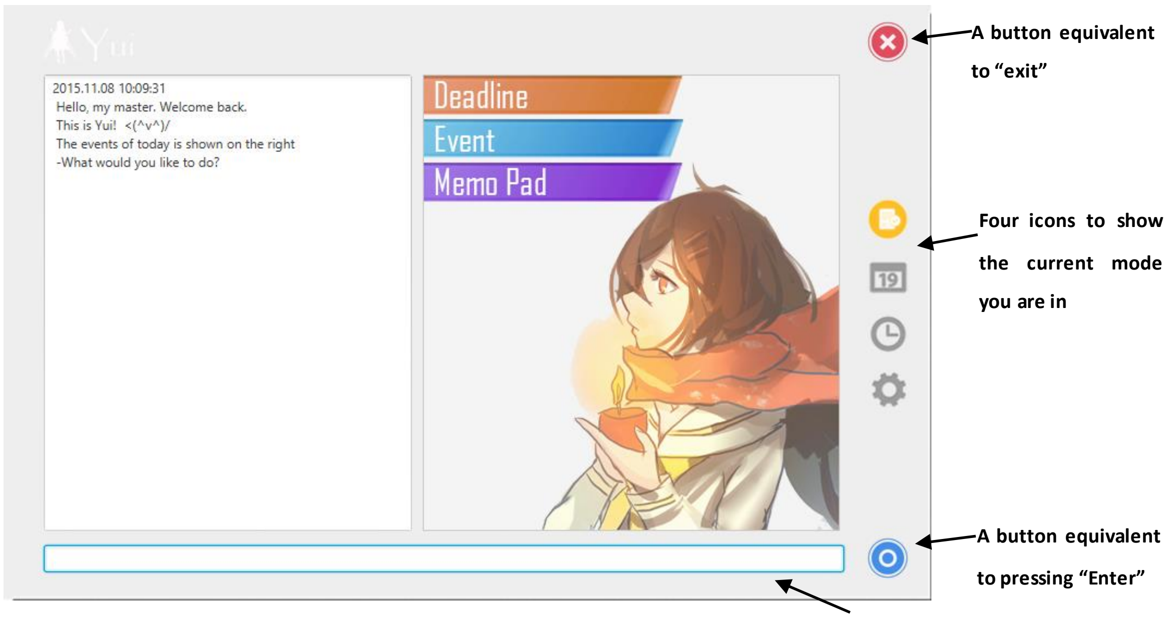
1 Introduction

Thank you for choosing Yui! Yui is an advanced task management software. It can be used to record your upcoming tasks and give you reminders when necessary. Users are able to control and use all the functionalities in this software by just keyboard commands, such as adding tasks, searching tasks and setting data path.

2 Quick Start

You can always find and download our latest release at https://github.com/cs2103aug2015-t11-2j/main/releases.

1) To ensure Yui runs on your machine, please download JDK 8u60 from the Oracle official website: http://www.oracle.com/technetwork/java/%20javase/downloads/index.html
2) To start our app, double click on the Yui.jar file.
3) Once you open the app, a window similar to Figure 2.1 will appear and you can start to manage your own event list.
4) If this is not the first time you open this app, you will see the events for today on the right hand side in order to allow you to quickly check them.
5) If you want to see all the events, type in "readall". To learn more, please read section 3 on more commands and functions.
6) You can also type in "help" to see the list of commands available.


Yui
Deadline
Event
Memo Pad
A button equivalent to "exit"
Four icons to show the current mode you are in
A button equivalent to pressing "Enter"
Type your commands here

2015.11.08 10:09:31
Hello, my master. Welcome back.
This is Yui! <(^v^)/
The events of today is shown on the right
-What would you like to do?

Figure 2.1. Demo of the interface of Yui

3 Basic Functions

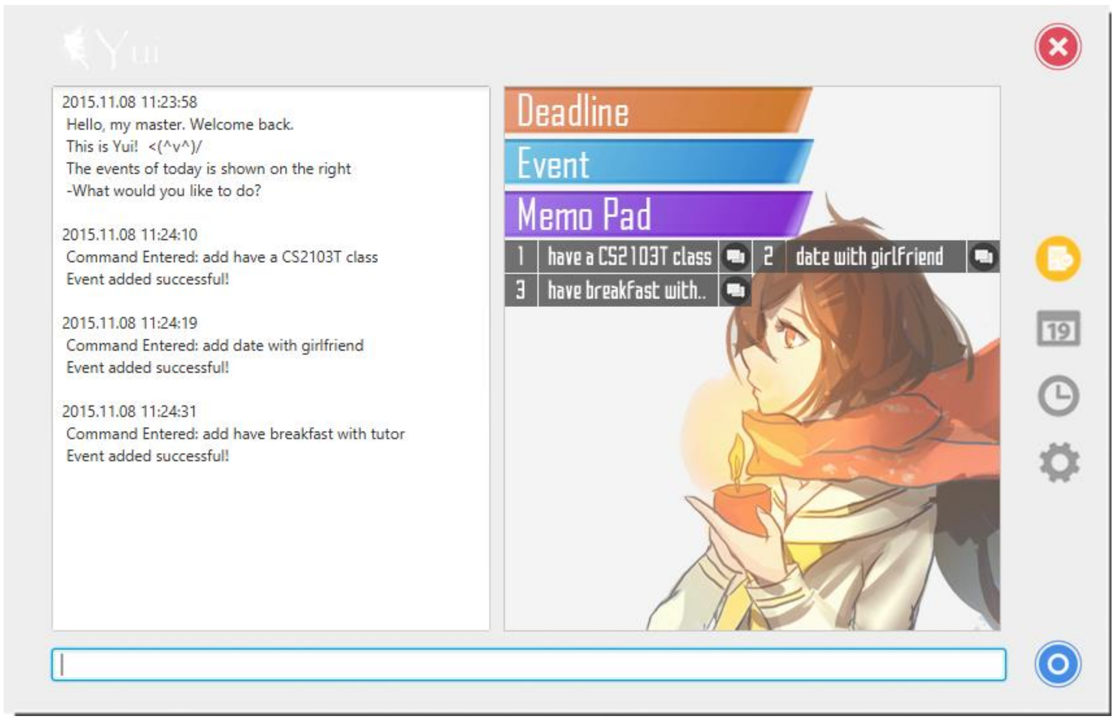
This section will give you an idea about what you can do with Yui and guide you to start using it. You will find this section most useful if you are a light user. Please read through all contents in this section before you actually start to use the app!

3.1 Add events

To add simple events (those without a time that you just want to do someday) to Yui, type "add `<event>`" into the software then hit 'Enter'. These events are defined as "Memo Pad" events hence will appear under the heading "Memo Pad". In addition, when you refer to a specific event in the list, you need to use the event type and its index. The event type for "Memo Pad" is "m" and the first three events in it will be "m1", "m2" and "m3".

Examples:

- `add have a CS2103T class`
- `add date with girlfriend`
- `add have breakfast with tutor`



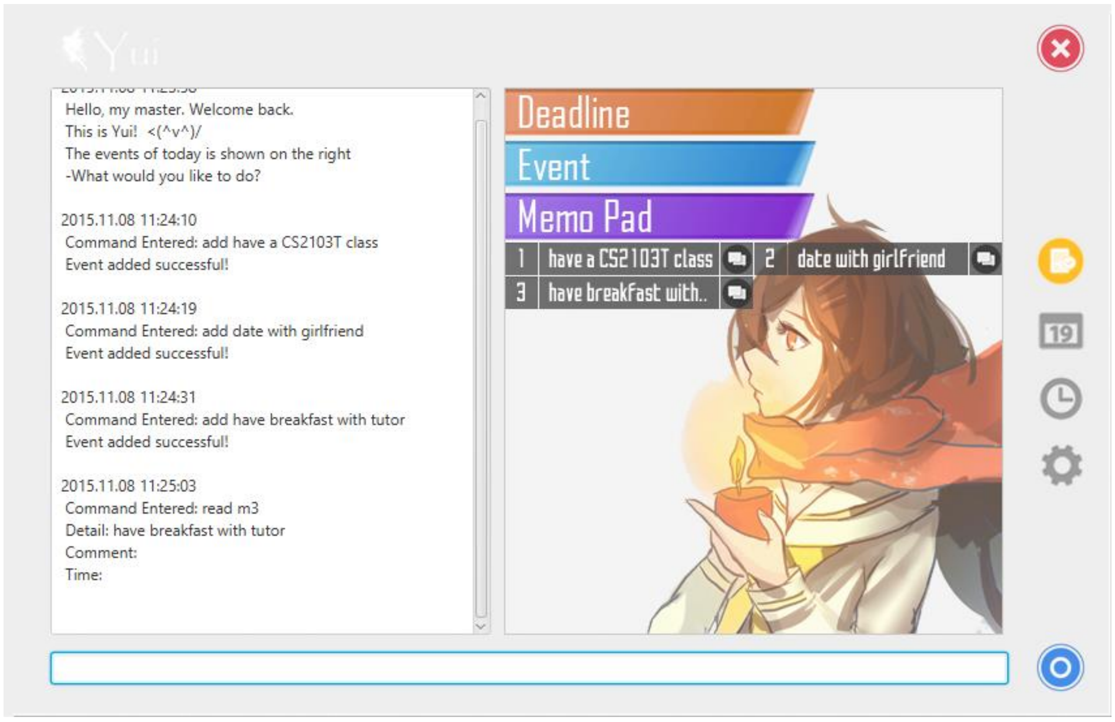
```
2015.11.08 11:23:58
Hello, my master. Welcome back.
This is Yui!  <(^v^)/
The events of today is shown on the right
-What would you like to do?

2015.11.08 11:24:10
Command Entered: add have a CS2103T class
Event added successful!

2015.11.08 11:24:19
Command Entered: add date with girlfriend
Event added successful!

2015.11.08 11:24:31
Command Entered: add have breakfast with tutor
Event added successful!
```

Figure 3.1. Response after adding three simple events

Note that the third event is too long and the extra contents will be cut off and replaced by ".." to show you there is extra information. You can find out how to see the cut off part in the next section.

3.2 Read events

Type "`read <event type and index>`" to display the event you specified. For example, you want to see the third "Memo Pad" event you entered because the name is too long and you cannot see the full name in the list. The figure on the next page shows what will happen if you type in "read m3".

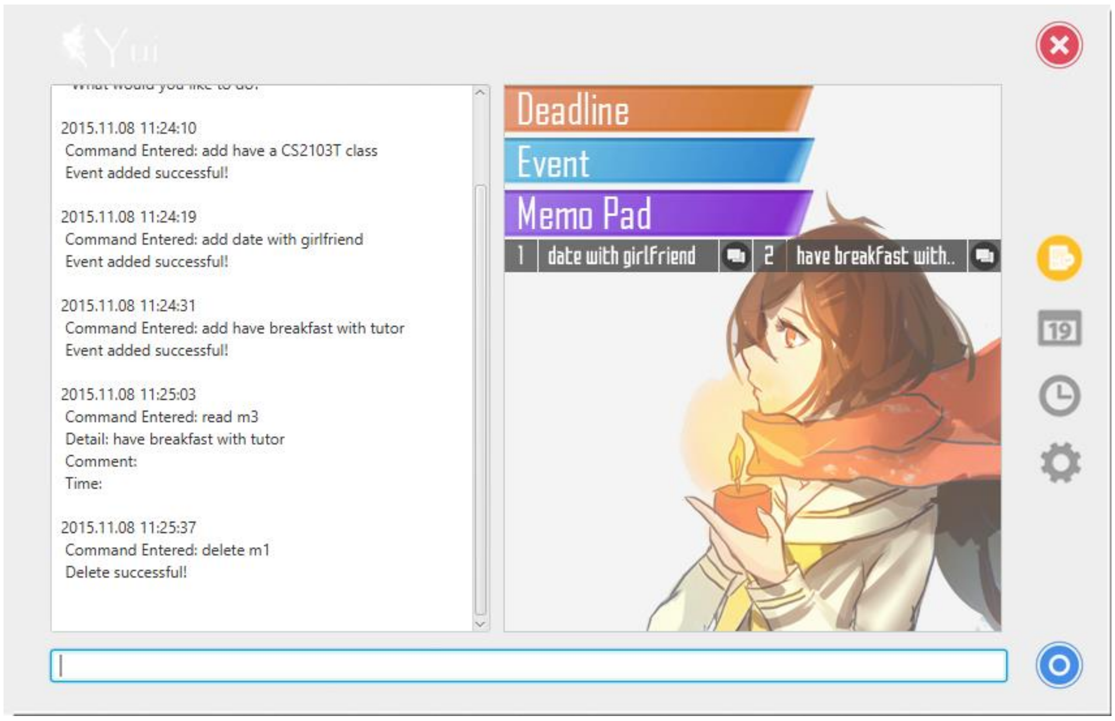


```
2015.11.08 11:23:56
Hello, my master. Welcome back.
This is Yui!  <(^v^)/
The events of today is shown on the right
-What would you like to do?

2015.11.08 11:24:10
Command Entered: add have a CS2103T class
Event added successful!

2015.11.08 11:24:19
Command Entered: add date with girlfriend
Event added successful!

2015.11.08 11:24:31
Command Entered: add have breakfast with tutor
Event added successful!

2015.11.08 11:25:03
Command Entered: read m3
Detail: have breakfast with tutor
Comment:
Time:
```

Figure 3.2 Response after typing in "read m3"

3.3 Delete events

If you wish to delete events that are no longer applicable, type "`delete <event type and index>`". That event will no longer be in your list. For example, you can "`delete m1`".

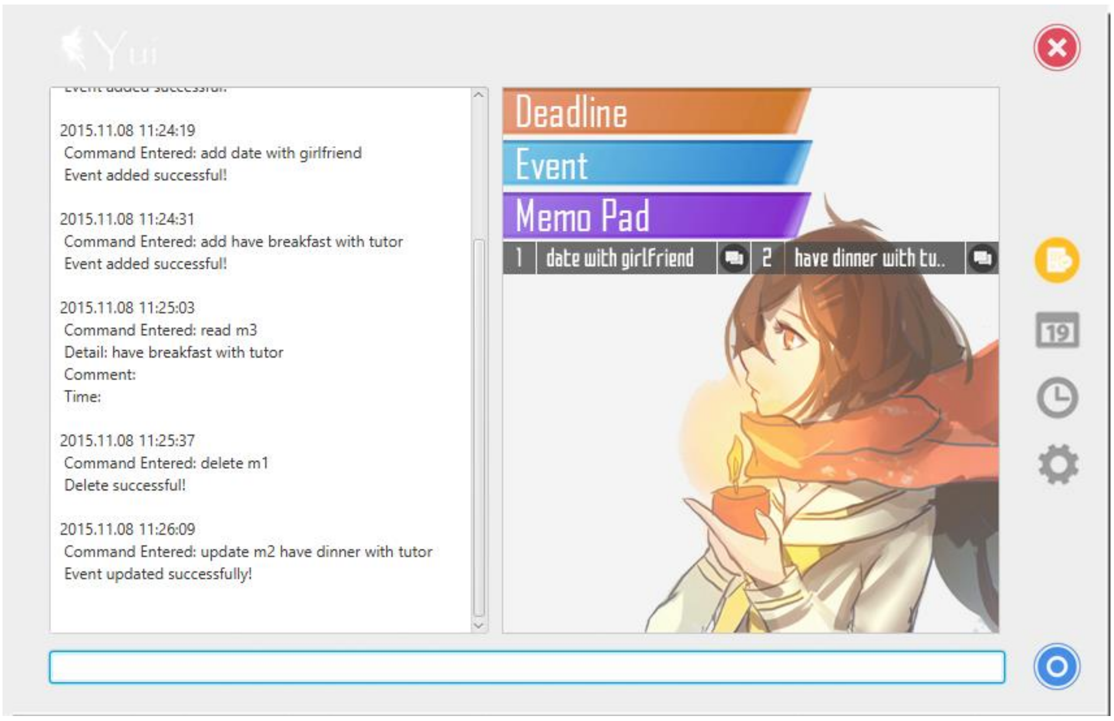


```
-What would you like to do?

2015.11.08 11:24:10
Command Entered: add have a CS2103T class
Event added successful!

2015.11.08 11:24:19
Command Entered: add date with girlfriend
Event added successful!

2015.11.08 11:24:31
Command Entered: add have breakfast with tutor
Event added successful!

2015.11.08 11:25:03
Command Entered: read m3
Detail: have breakfast with tutor
Comment:
Time:

2015.11.08 11:25:37
Command Entered: delete m1
Delete successful!
```

Figure 3.3. Response after deleting an event

3.4 Update events

If you realize that you have made a mistake in the event you keyed in, you can always replace it by typing "`update <event type and index> <event>`". We assume you only made minor mistakes so we will only change the name and time of event. If you made lots of mistakes, it is probably a better idea to delete that event then re-enter. For example, you can change "breakfast" to "dinner" in m2.

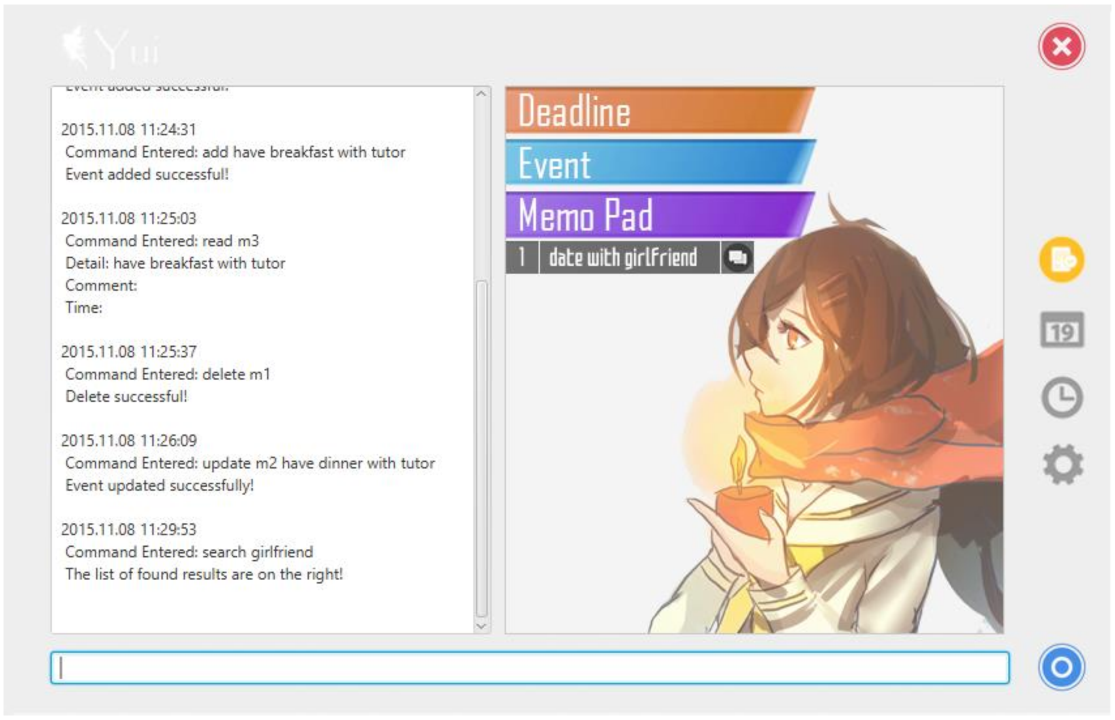


```
2015.11.08 11:24:19
 Command Entered: add date with girlfriend
 Event added successful!

2015.11.08 11:24:31
 Command Entered: add have breakfast with tutor
 Event added successful!

2015.11.08 11:25:03
 Command Entered: read m3
 Detail: have breakfast with tutor
 Comment:
 Time:

2015.11.08 11:25:37
 Command Entered: delete m1
 Delete successful!

2015.11.08 11:26:09
 Command Entered: update m2 have dinner with tutor
 Event updated successfully!
```

Figure 3.4. Response after updating an event

3.5 Search events

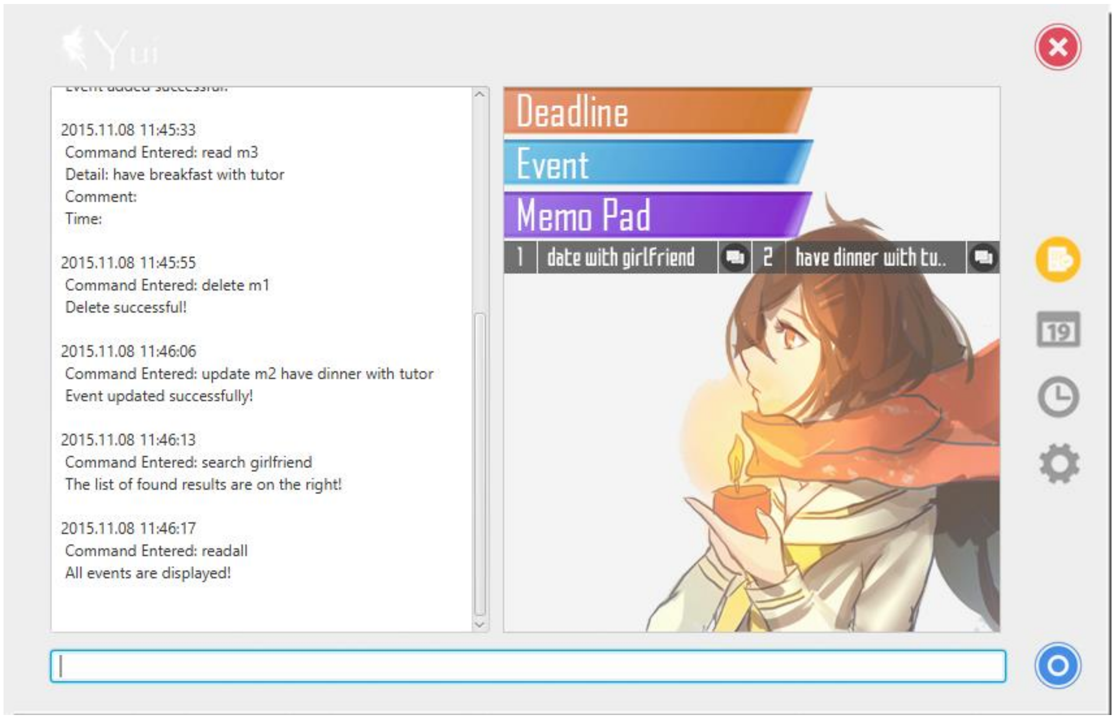
To look for events with specific details, type in "`search <keyword>`". You may find this function useful when you have a lot of events in your list and you can only remember a small part of the event you are looking for. For example, you can try "`search girlfriend`" just to see what will happen. You can refer to the figure on next page.



```
Event added successful!

2015.11.08 11:24:31
 Command Entered: add have breakfast with tutor
 Event added successful!

2015.11.08 11:25:03
 Command Entered: read m3
 Detail: have breakfast with tutor
 Comment:
 Time:

2015.11.08 11:25:37
 Command Entered: delete m1
 Delete successful!

2015.11.08 11:26:09
 Command Entered: update m2 have dinner with tutor
 Event updated successfully!

2015.11.08 11:29:53
 Command Entered: search girlfriend
 The list of found results are on the right!
```

Figure 3.5. Search result of “girlfriend”

Note that if you still have a long list after your first search, you can search again and this timeitwill be searching for events from the results of your first search. If you then want to go back to searching through all the events, you need to type in “`readall`” first to call out all the events.

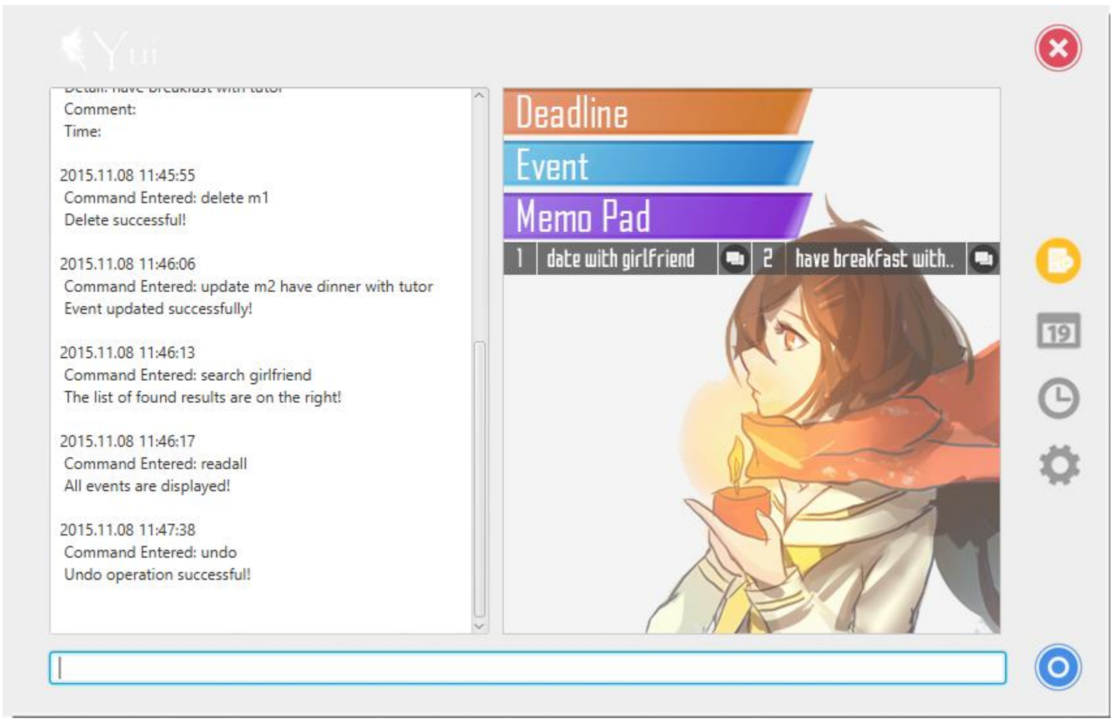


```
Event added successful!

2015.11.08 11:45:33
 Command Entered: read m3
 Detail: have breakfast with tutor
 Comment:
 Time:

2015.11.08 11:45:55
 Command Entered: delete m1
 Delete successful!

2015.11.08 11:46:06
 Command Entered: update m2 have dinner with tutor
 Event updated successfully!

2015.11.08 11:46:13
 Command Entered: search girlfriend
 The list of found results are on the right!

2015.11.08 11:46:17
 Command Entered: readall
 All events are displayed!
```

Figure 3.6. Response of “readall”

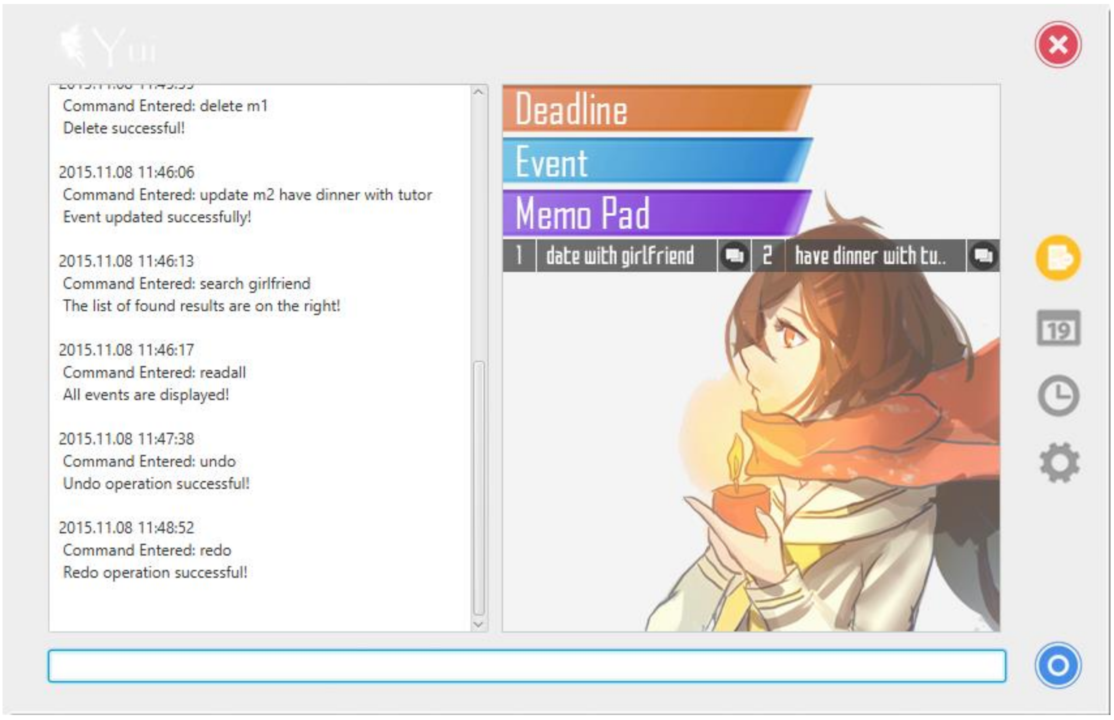
3.6 Undo and Redo

If you wants to undo your last change in the program, type in "undo". Commands such as "read" or "search" will be ignored by "undo" because you did not make any changes to the list. In addition, you cannot "undo" twice in a row! To revert your undo operation, type "redo".



```
Detail: have breakfast with tutor
Comment:
Time:

2015.11.08 11:45:55
Command Entered: delete m1
Delete successful!

2015.11.08 11:46:06
Command Entered: update m2 have dinner with tutor
Event updated successfully!

2015.11.08 11:46:13
Command Entered: search girlfriend
The list of found results are on the right!

2015.11.08 11:46:17
Command Entered: readall
All events are displayed!

2015.11.08 11:47:38
Command Entered: undo
Undo operation successful!
```

Figure 3.7. Response after "undo"

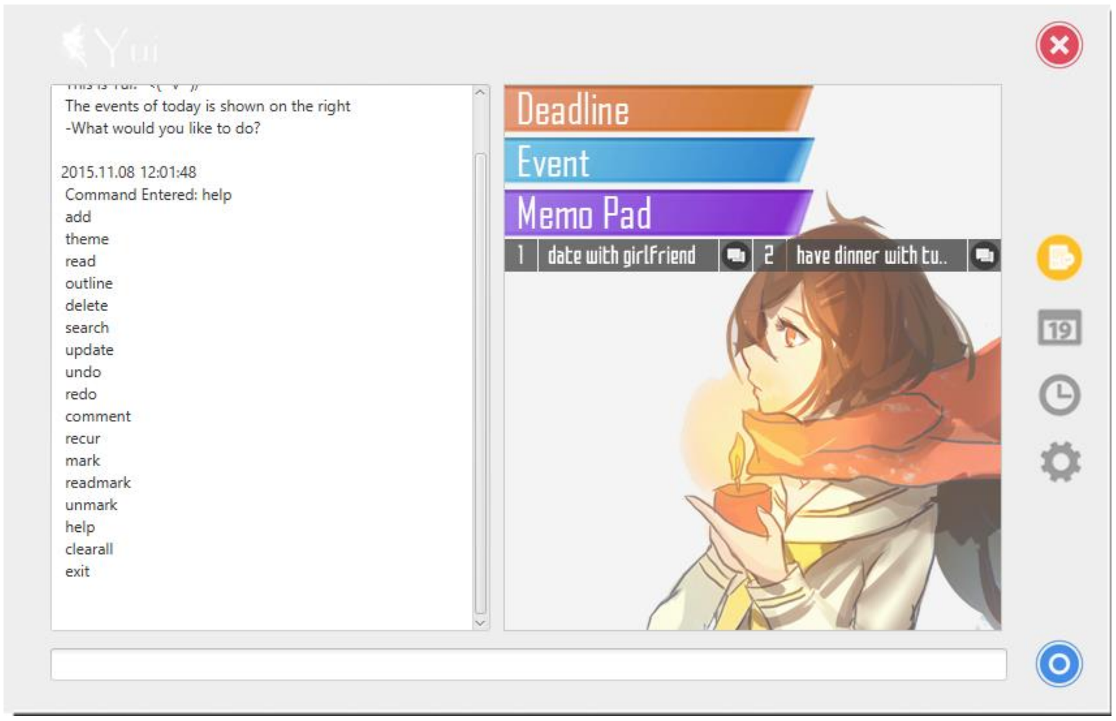


```
2015.11.08 11:45:55
Command Entered: delete m1
Delete successful!

2015.11.08 11:46:06
Command Entered: update m2 have dinner with tutor
Event updated successfully!

2015.11.08 11:46:13
Command Entered: search girlfriend
The list of found results are on the right!

2015.11.08 11:46:17
Command Entered: readall
All events are displayed!

2015.11.08 11:47:38
Command Entered: undo
Undo operation successful!

2015.11.08 11:48:52
Command Entered: redo
Redo operation successful!
```

Figure 3.8. Response after "redo"

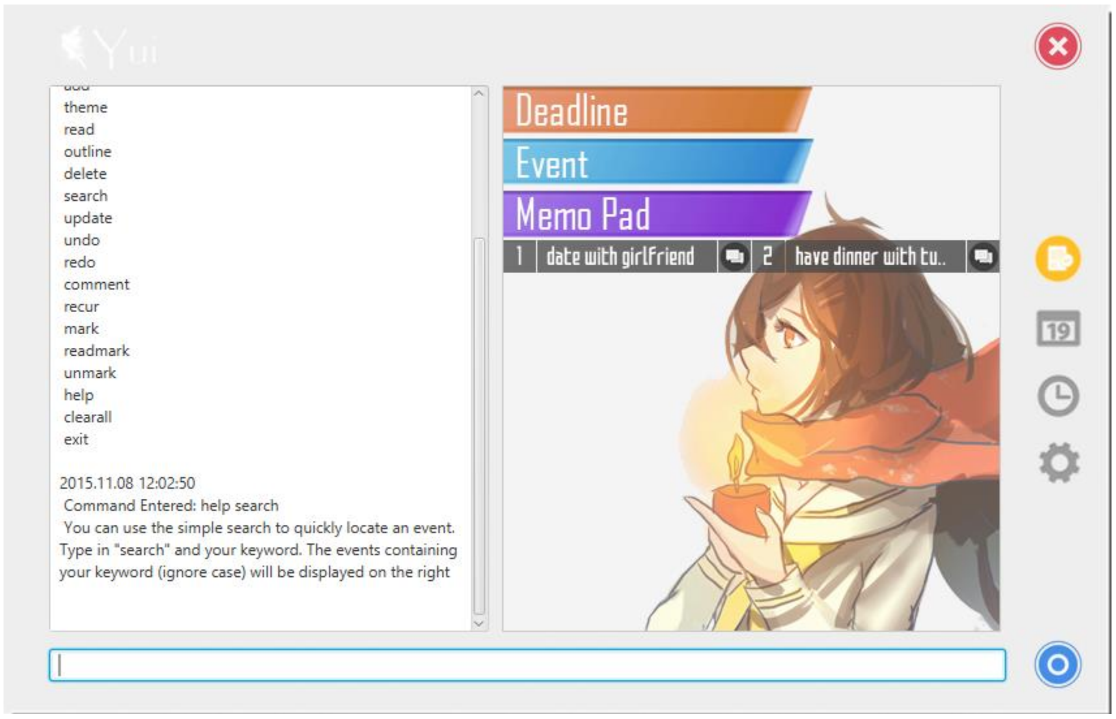
3.7 Help

When you feel lost and do not know what to do, type in “help” to call out the list of all the functions available. To learn more about a function, type in “`help <function name>`”.



```
The events of today is shown on the right
-What would you like to do?

2015.11.08 12:01:48
Command Entered: help
add
theme
read
outline
delete
search
update
undo
redo
comment
recur
mark
readmark
unmark
help
clearall
exit
```

Figure 3.9. Response after “help”



```
theme
read
outline
delete
search
update
undo
redo
comment
recur
mark
readmark
unmark
help
clearall
exit

2015.11.08 12:02:50
Command Entered: help search
You can use the simple search to quickly locate an event.
Type in "search" and your keyword. The events containing
your keyword (ignore case) will be displayed on the right
```

Figure 3.10. Response after “help search”

3.8 Exit

When you want to exit Yui, simply type "`exit`". The date will be saved automatically after each operation so you do not need to worry about losing your list upon closing it!

4 More possibilities

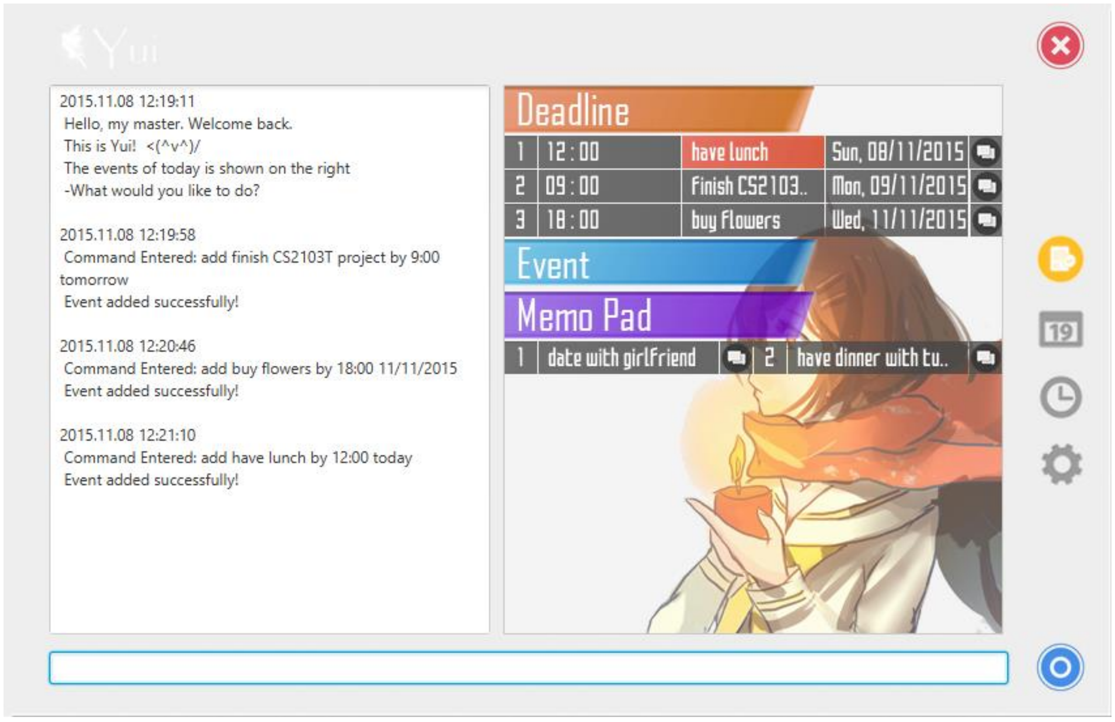
This section gives information on more commands. With special permutations of commands, you can do more amazing stuff. After reading this section, you should be able to use most of the functions we currently provide in Yui. However, feel free to read on section 5 for the extra features we provide for heavy users!

4.1 Add events with deadlines

If the event that you wish to add must be carried out by a specific date and time, type "add `<event>` by <HH:mm> <DD/MM/YY>", where <HH:mm> represents the time and <DD/MM/YY> represents the date. You may also substitute the date with either "today" or "tomorrow" ("tmr"). Similar to "Memo Pad", the "Deadline" event has event type of "d" and you need to refer to them using "d1", "d2", etc.

Examples:

- `add finish CS2103T project by 9:00 tomorrow`
- `add buy flowers by 18:00 11/11/2015`
- `add have lunch by 12:00 today`



```
2015.11.08 12:19:11
 Hello, my master. Welcome back.
 This is Yui!  <(^v^)/
 The events of today is shown on the right
 -What would you like to do?

2015.11.08 12:19:58
 Command Entered: add finish CS2103T project by 9:00
tomorrow
 Event added successfully!

2015.11.08 12:20:46
 Command Entered: add buy flowers by 18:00 11/11/2015
 Event added successfully!

2015.11.08 12:21:10
 Command Entered: add have lunch by 12:00 today
 Event added successfully!
```

Deadline

1	12:00	have lunch	Sun, 08/11/2015
2	09:00	Finish CS2103..	Mon, 09/11/2015
3	18:00	buy flowers	Wed, 11/11/2015

Event

Memo Pad

1	date with girlfriend	2	have dinner with tu..

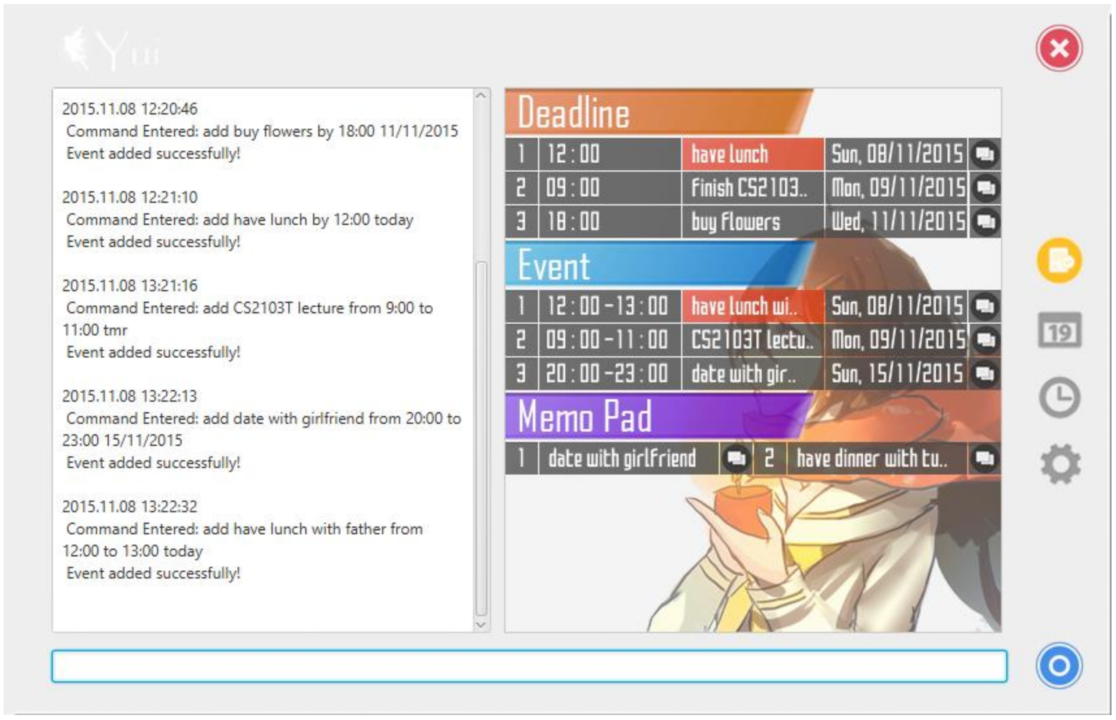
Figure 4.1. Response after adding three deadline events

4.2 Add events with start and end times

Type “`add <event> from <HH:mm> to <HH:mm> <DD/MM/YY>`” to add a task that starts and ends at a specific time. Like events with deadlines, you may substitute the date with “today” or “tomorrow”. These events are assigned an event type of “e” and should be referred to using “e1”, “e2”, etc.

Examples:

- `add CS2103T lecture from 9:00 to 11:00 tmr`
- `add date with girlfriend from 20:00 to 23:00 15/11/2015`
- `add have lunch with father from 12:00 to 13:00 today`



```
2015.11.08 12:20:46
 Command Entered: add buy flowers by 18:00 11/11/2015
 Event added successfully!

2015.11.08 12:21:10
 Command Entered: add have lunch by 12:00 today
 Event added successfully!

2015.11.08 13:21:16
 Command Entered: add CS2103T lecture from 9:00 to
11:00 tmr
 Event added successfully!

2015.11.08 13:22:13
 Command Entered: add date with girlfriend from 20:00 to
23:00 15/11/2015
 Event added successfully!

2015.11.08 13:22:32
 Command Entered: add have lunch with father from
12:00 to 13:00 today
 Event added successfully!
```

Deadline

1	12 : 00	have lunch	Sun, 08/11/2015
2	09 : 00	Finish CS2103..	Mon, 09/11/2015
3	18 : 00	buy Flowers	Wed, 11/11/2015

Event

1	12 : 00 - 13 : 00	have lunch wi..	Sun, 08/11/2015
2	09 : 00 - 11 : 00	CS2103T lectu..	Mon, 09/11/2015
3	20 : 00 - 23 : 00	date with gir..	Sun, 15/11/2015

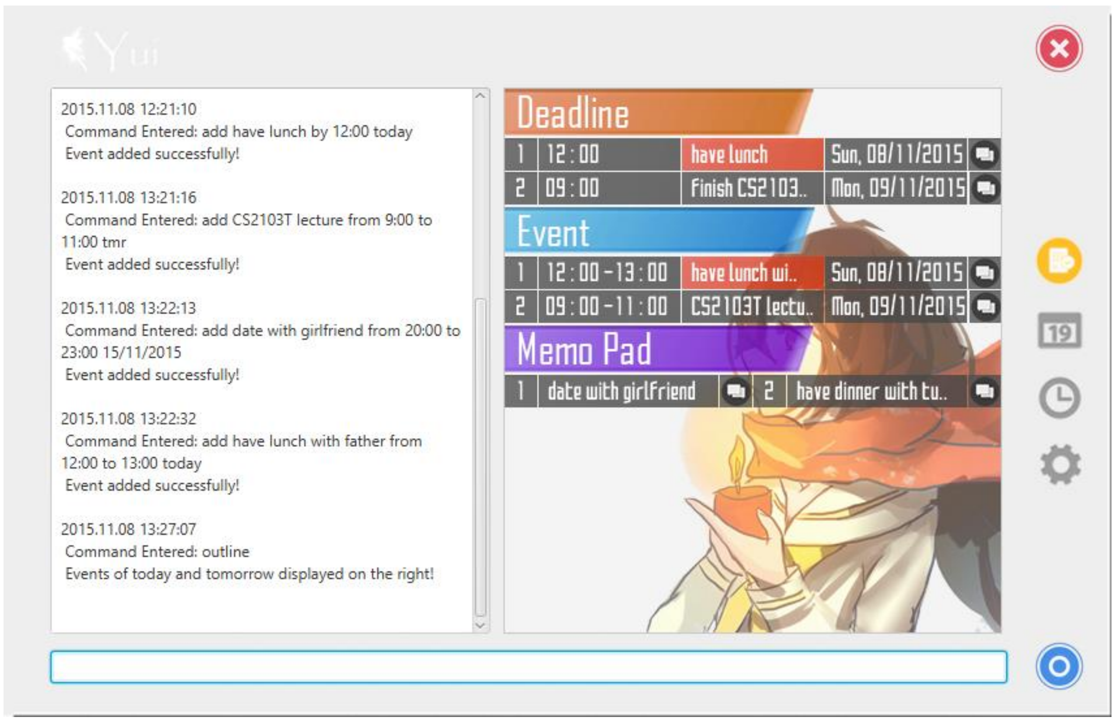
Memo Pad

1	date with girlFriend	2	have dinner with tu..

Figure 4.2. Response after adding three deadline

4.3 Outline

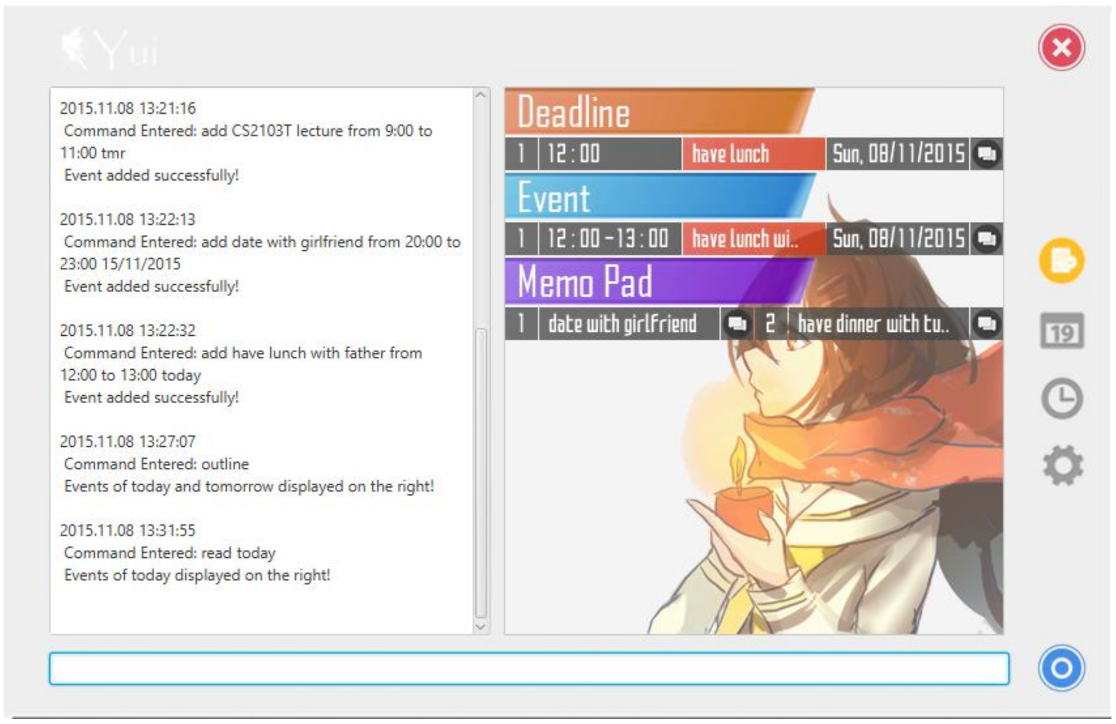
To display the events that need to be done today or tomorrow, type in “`outline`”. Alternatively, you can type in “`read today`” or “`read tomorrow`” (“`read tmr`”) to check the events that occur on today or tomorrow respectively.



```
2015.11.08 12:21:10
 Command Entered: add have lunch by 12:00 today
 Event added successfully!

2015.11.08 13:21:16
 Command Entered: add CS2103T lecture from 9:00 to
11:00 tmr
 Event added successfully!

2015.11.08 13:22:13
 Command Entered: add date with girlfriend from 20:00 to
23:00 15/11/2015
 Event added successfully!

2015.11.08 13:22:32
 Command Entered: add have lunch with father from
12:00 to 13:00 today
 Event added successfully!

2015.11.08 13:27:07
 Command Entered: outline
 Events of today and tomorrow displayed on the right!
```

Yui
Deadline
1 | 12 : 00 | have lunch | Sun, 08/11/2015
2 | 09 : 00 | Finish CS2103.. | Mon, 09/11/2015
Event
1 | 12 : 00 - 13 : 00 | have lunch wi.. | Sun, 08/11/2015
2 | 09 : 00 - 11 : 00 | CS2103T lectu.. | Mon, 09/11/2015
Memo Pad
1 | date with girlfriend | 2 | have dinner with tu..
19

Figure 4.3. Response after "outline"



```
2015.11.08 13:21:16
 Command Entered: add CS2103T lecture from 9:00 to
11:00 tmr
 Event added successfully!

2015.11.08 13:22:13
 Command Entered: add date with girlfriend from 20:00 to
23:00 15/11/2015
 Event added successfully!

2015.11.08 13:22:32
 Command Entered: add have lunch with father from
12:00 to 13:00 today
 Event added successfully!

2015.11.08 13:27:07
 Command Entered: outline
 Events of today and tomorrow displayed on the right!

2015.11.08 13:31:55
 Command Entered: read today
 Events of today displayed on the right!
```

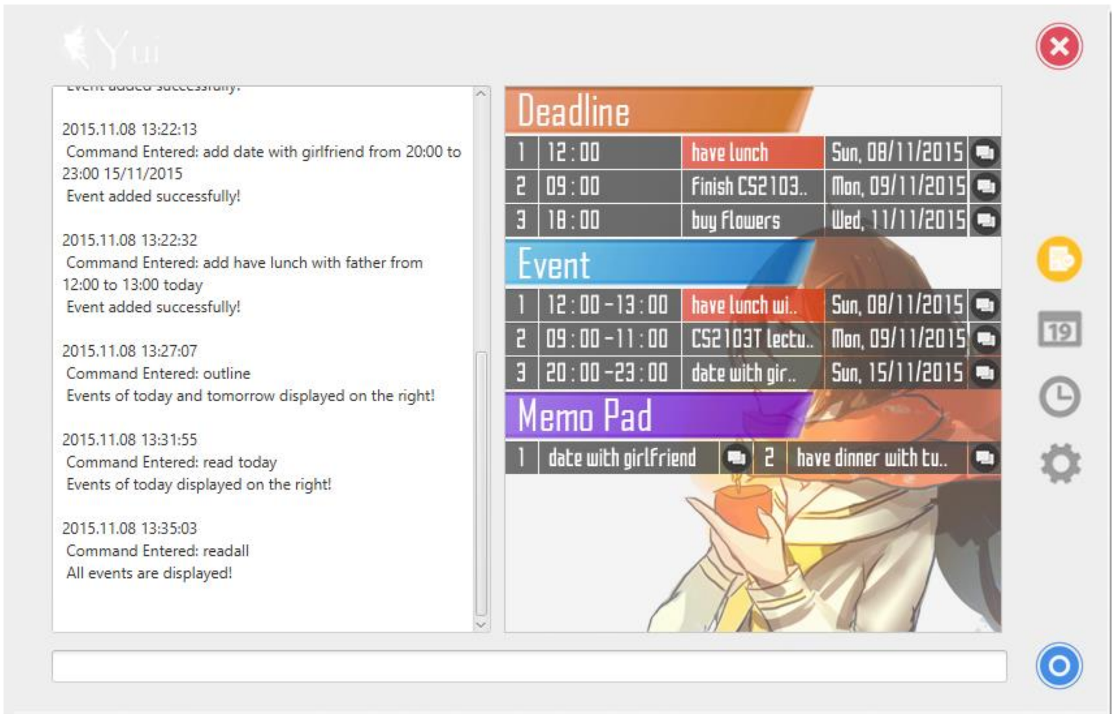
Yui
Deadline
1 | 12 : 00 | have lunch | Sun, 08/11/2015
Event
1 | 12 : 00 - 13 : 00 | have lunch wi.. | Sun, 08/11/2015
Memo Pad
1 | date with girlfriend | 2 | have dinner with tu..
19

Figure 4.4. Response after "outline"

Note that for each of these three commands, all the events in "Memo Pad" will always be shown so that you can choose to finish some of the "Memo Pad" if you think you have much free time.

4.4 Mark as 'complete'/'done'

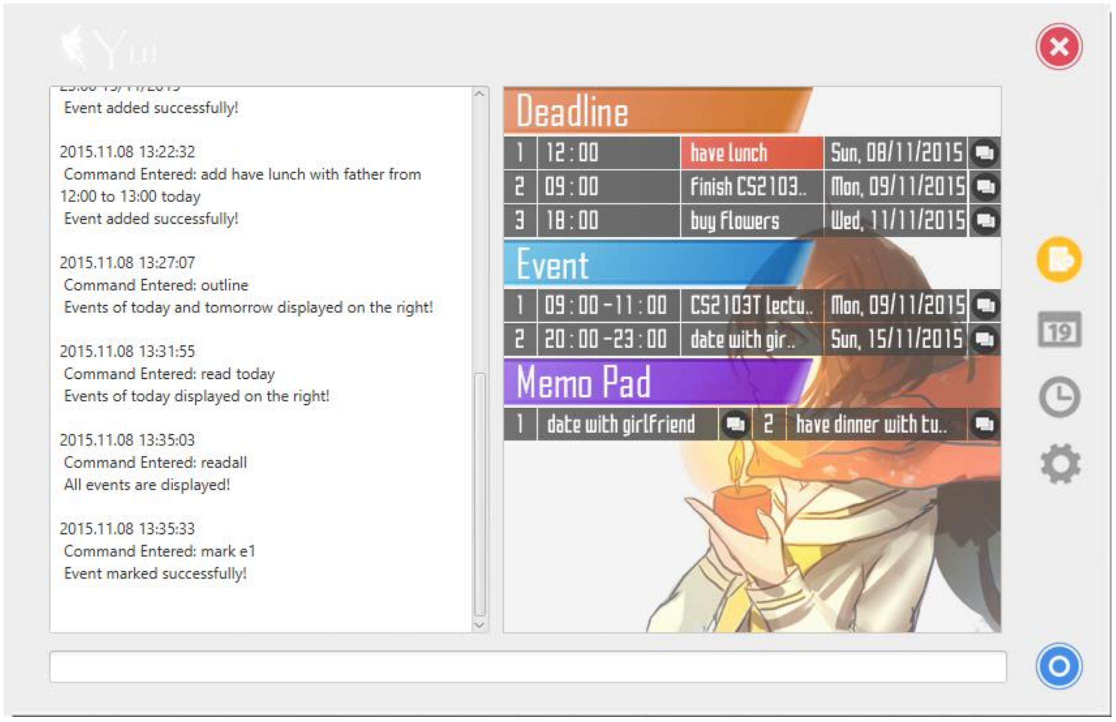
When you have finished an event, type "`mark <event type and index>`" to set it as completed/done. The event will no longer appear in your list.



```
Event added successfully!

2015.11.08 13:22:13
Command Entered: add date with girlfriend from 20:00 to
23:00 15/11/2015
Event added successfully!

2015.11.08 13:22:32
Command Entered: add have lunch with father from
12:00 to 13:00 today
Event added successfully!

2015.11.08 13:27:07
Command Entered: outline
Events of today and tomorrow displayed on the right!

2015.11.08 13:31:55
Command Entered: read today
Events of today displayed on the right!

2015.11.08 13:35:03
Command Entered: readall
All events are displayed!
```

```
Deadline
1 | 12:00         | have lunch     | Sun, 08/11/2015
2 | 09:00         | Finish CS2103..| Mon, 09/11/2015
3 | 18:00         | buy flowers    | Wed, 11/11/2015
Event
1 | 12:00-13:00   | have lunch wi..| Sun, 08/11/2015
2 | 09:00-11:00   | CS2103T lectu..| Mon, 09/11/2015
3 | 20:00-23:00   | date with gir..| Sun, 15/11/2015
Memo Pad
1 | date with girlfriend | 2 | have dinner with tu..
```

Figure 4.5. Status before mark

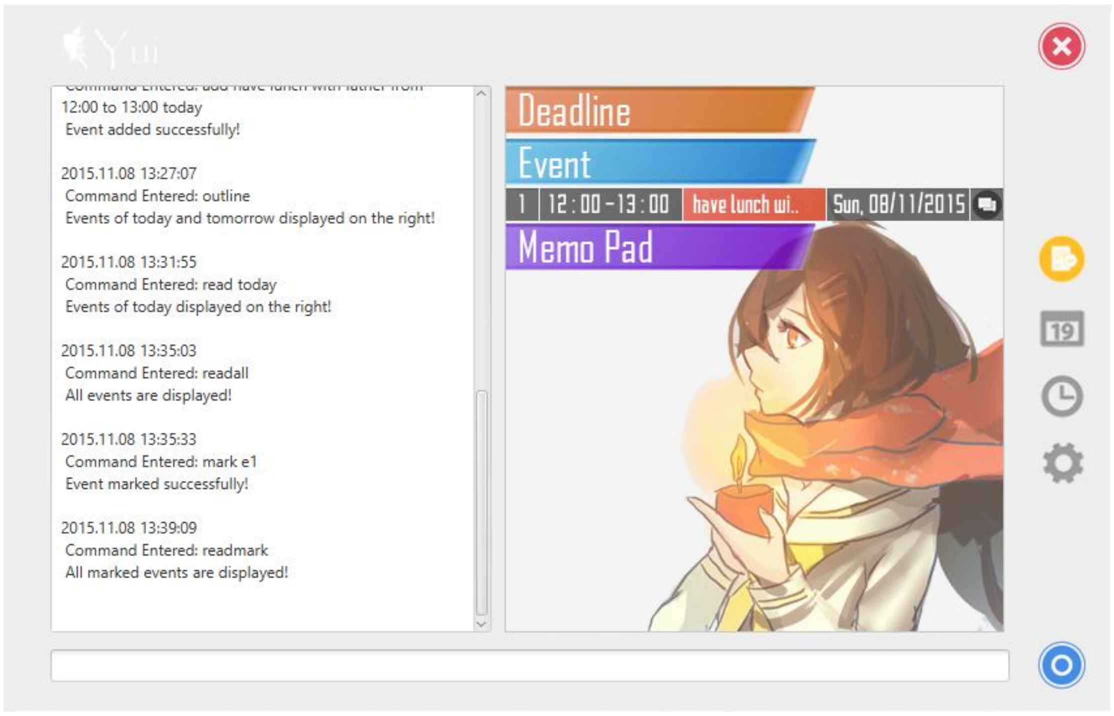


```
23:00 15/11/2015
Event added successfully!

2015.11.08 13:22:32
Command Entered: add have lunch with father from
12:00 to 13:00 today
Event added successfully!

2015.11.08 13:27:07
Command Entered: outline
Events of today and tomorrow displayed on the right!

2015.11.08 13:31:55
Command Entered: read today
Events of today displayed on the right!

2015.11.08 13:35:03
Command Entered: readall
All events are displayed!

2015.11.08 13:35:33
Command Entered: mark e1
Event marked successfully!
```

```
Deadline
1 | 12:00         | have lunch     | Sun, 08/11/2015
2 | 09:00         | Finish CS2103..| Mon, 09/11/2015
3 | 18:00         | buy flowers    | Wed, 11/11/2015
Event
1 | 09:00-11:00   | CS2103T lectu..| Mon, 09/11/2015
2 | 20:00-23:00   | date with gir..| Sun, 15/11/2015
Memo Pad
1 | date with girlfriend | 2 | have dinner with tu..
```

Figure 4.6. Response after "mark e1"

You can "undo" the "mark" operation. Alternatively, you can type in "readmark" to check all the marked events. You can then "unmark" an event you accidentally marked to bring it back to your list.



```
12:00 to 13:00 today
Event added successfully!

2015.11.08 13:27:07
Command Entered: outline
Events of today and tomorrow displayed on the right!

2015.11.08 13:31:55
Command Entered: read today
Events of today displayed on the right!

2015.11.08 13:35:03
Command Entered: readall
All events are displayed!

2015.11.08 13:35:33
Command Entered: mark e1
Event marked successfully!

2015.11.08 13:39:09
Command Entered: readmark
All marked events are displayed!
```

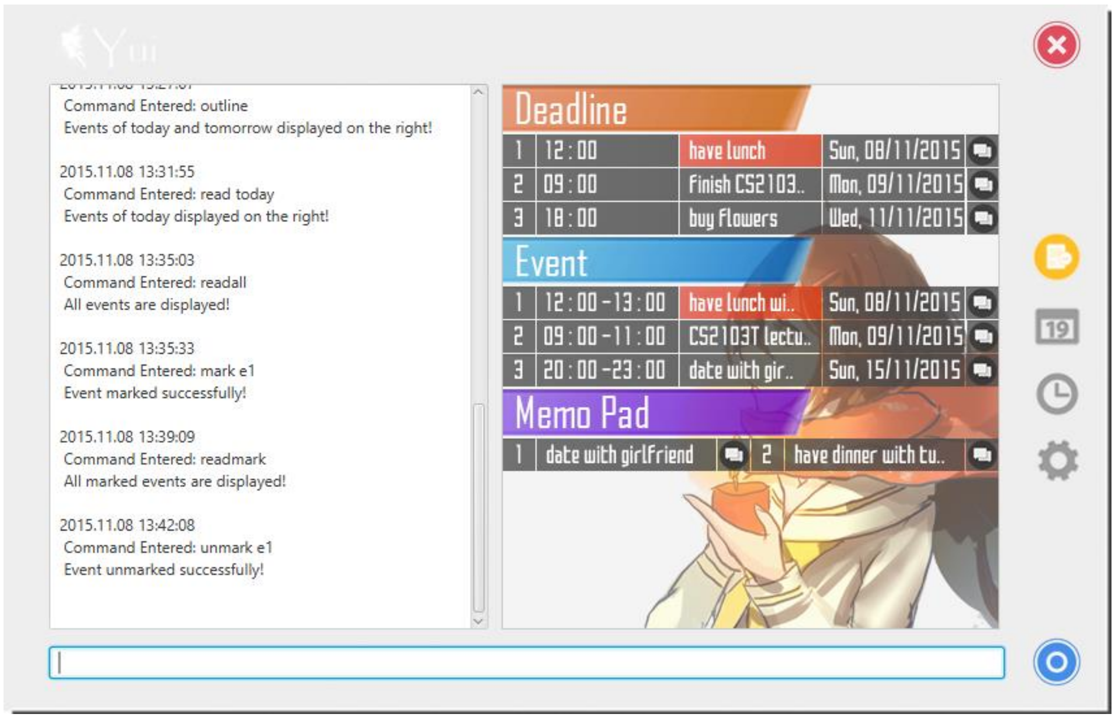
Deadline

Event

1	12:00-13:00	have lunch wi..	Sun, 08/11/2015

Memo Pad

Figure 4.7. Response after "readmark"



```
Command Entered: outline
Events of today and tomorrow displayed on the right!

2015.11.08 13:31:55
Command Entered: read today
Events of today displayed on the right!

2015.11.08 13:35:03
Command Entered: readall
All events are displayed!

2015.11.08 13:35:33
Command Entered: mark e1
Event marked successfully!

2015.11.08 13:39:09
Command Entered: readmark
All marked events are displayed!

2015.11.08 13:42:08
Command Entered: unmark e1
Event unmarked successfully!
```

Deadline

1	12:00	have lunch	Sun, 08/11/2015
2	09:00	Finish CS2103..	Mon, 09/11/2015
3	18:00	buy flowers	Wed, 11/11/2015

Event

1	12:00-13:00	have lunch wi..	Sun, 08/11/2015
2	09:00-11:00	CS2103T lectu..	Mon, 09/11/2015
3	20:00-23:00	date with gir..	Sun, 15/11/2015

Memo Pad

1	date with girlfriend	2	have dinner with tu..

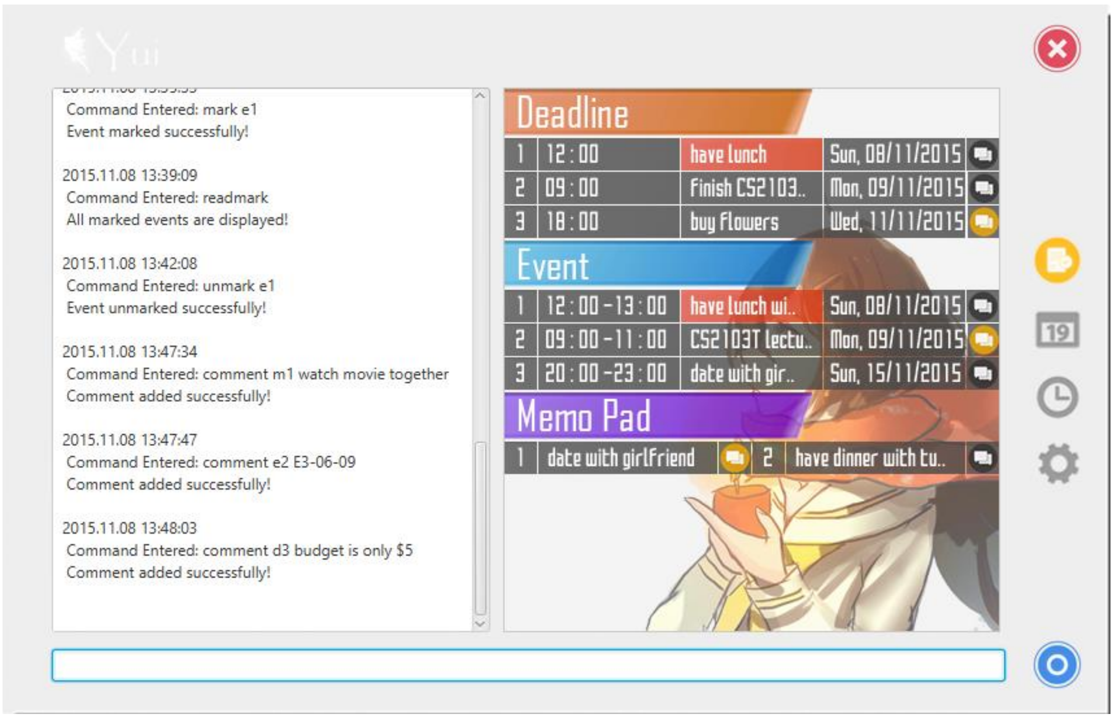
Figure 4.8. Response after "unmark e1"

4.5 Comments

To add comments to events, type in "`comment <event type and index> <comment>`". This is useful if the event has some additional details that you do not want to put into the event title.

Examples:

- `comment m1 watch movie together`
- `comment e2 E3-06-09`
- `comment d3 budget is only $5`



```
Command Entered: mark e1
Event marked successfully!

2015.11.08 13:39:09
Command Entered: readmark
All marked events are displayed!

2015.11.08 13:42:08
Command Entered: unmark e1
Event unmarked successfully!

2015.11.08 13:47:34
Command Entered: comment m1 watch movie together
Comment added successfully!

2015.11.08 13:47:47
Command Entered: comment e2 E3-06-09
Comment added successfully!

2015.11.08 13:48:03
Command Entered: comment d3 budget is only $5
Comment added successfully!
```

Deadline

1	12 : 00	have lunch	Sun, 08/11/2015
2	09 : 00	Finish CS2103..	Mon, 09/11/2015
3	18 : 00	buy flowers	Wed, 11/11/2015

Event

1	12 : 00 - 13 : 00	have lunch wi..	Sun, 08/11/2015
2	09 : 00 - 11 : 00	CS2103T lectu..	Mon, 09/11/2015
3	20 : 00 - 23 : 00	date with gir..	Sun, 15/11/2015

Memo Pad

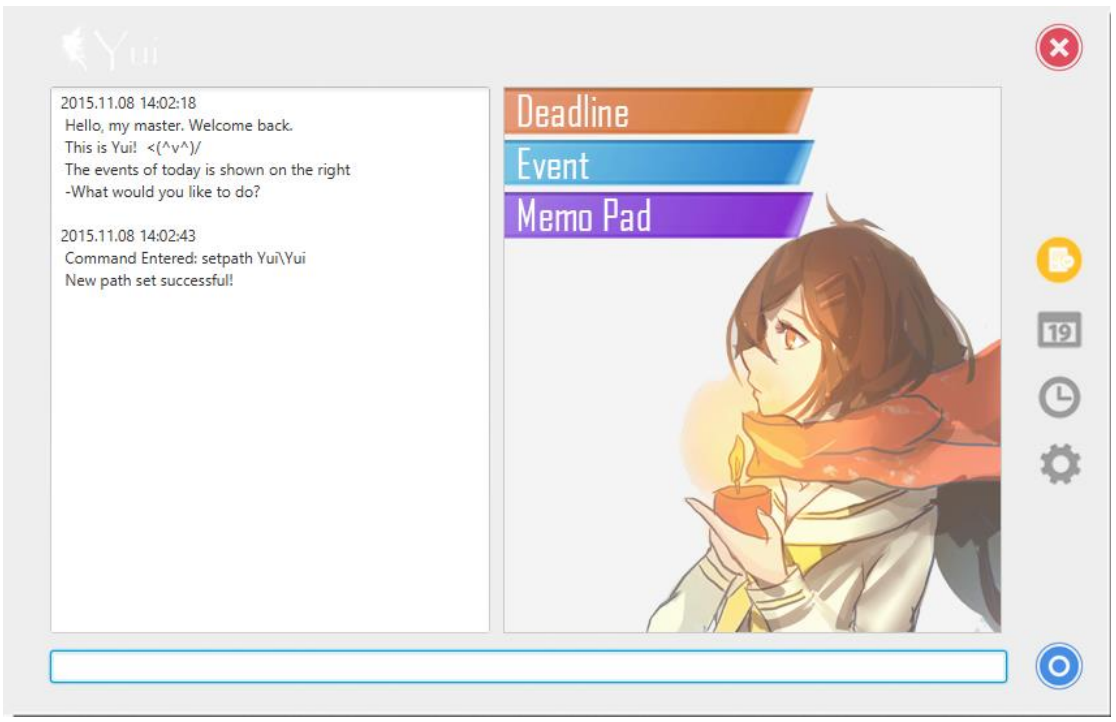
1	date with girlfriend	2	have dinner with tu..

Figure 4.9. Response after adding three comments

Note that for events with comments, the icon on the right hand side of that event will be highlighted. You can always read that event to find out what the comment is.

4.6 Set own path for data

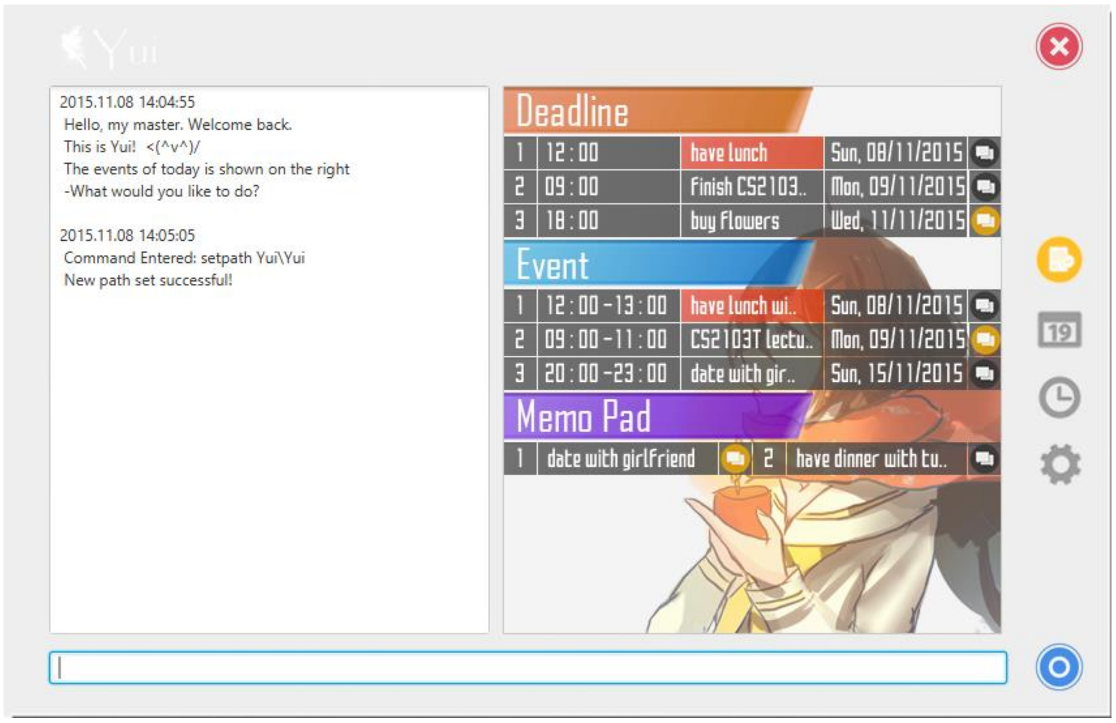
The default location Yui stores your data file is the same location where you put the main app. However, if you want to store it in another place (for example, the folder controlled by your cloud server so that you can use it on another machine), you can type in "`setpath <your directory>\<your data file name>`". Yui will now start to save your list in the new directory. However, if you still need the old list, you need to manually copy the data file to the new directory.



2015.11.08 14:02:18
Hello, my master. Welcome back.
This is Yui! <(^v^)/
The events of today is shown on the right
-What would you like to do?

2015.11.08 14:02:43
Command Entered: setpath Yui\Yui
New path set successful!

Figure 4.10. Response after setting a new path



2015.11.08 14:04:55
Hello, my master. Welcome back.
This is Yui! <(^v^)/
The events of today is shown on the right
-What would you like to do?

2015.11.08 14:05:05
Command Entered: setpath Yui\Yui
New path set successful!

Deadline

1	12 : 00	have lunch	Sun, 08/11/2015
2	09 : 00	Finish CS2103..	Mon, 09/11/2015
3	18 : 00	buy flowers	Wed, 11/11/2015

Event

1	12 : 00 - 13 : 00	have lunch wi..	Sun, 08/11/2015
2	09 : 00 - 11 : 00	CS2103T lectu..	Mon, 09/11/2015
3	20 : 00 - 23 : 00	date with gir..	Sun, 15/11/2015

Memo Pad

1	date with girlfriend	2	have dinner with tu..

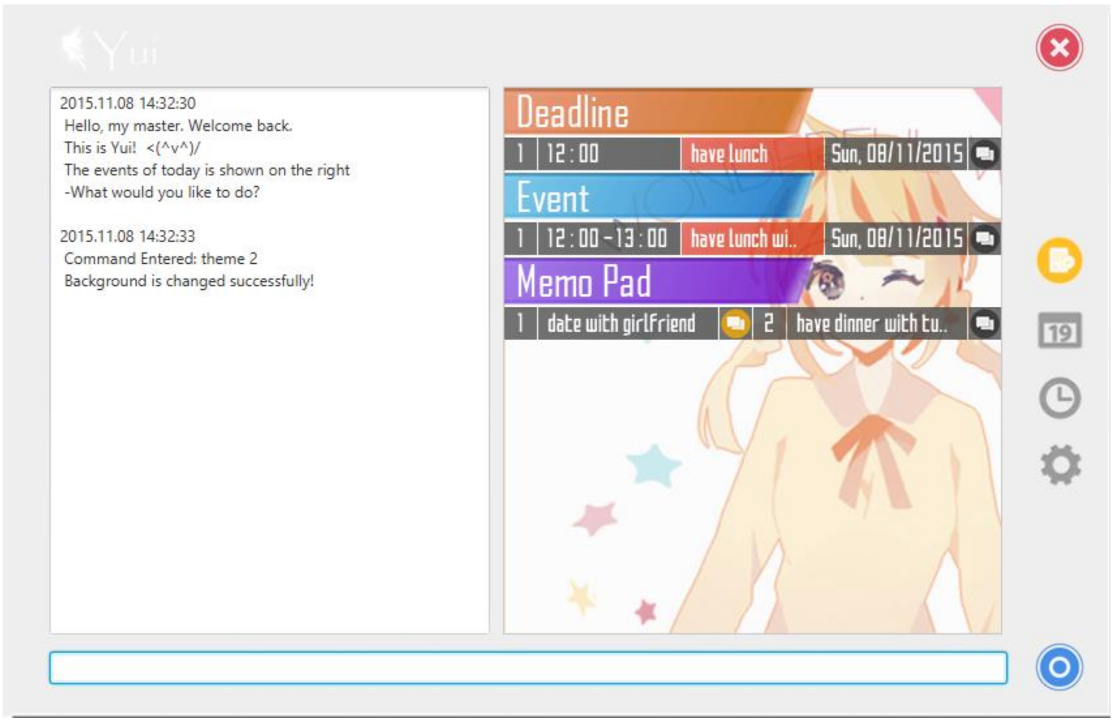
Figure 4.11. Response if file already exists in the new directory

5 Extra features

This section talks about the add-ons and extensions to Yui we have already implemented. You have to learn to use the features but they will definitely help to increase your productivity if you are a heavy user!

5.1 Changing theme

If you feel bored about the current theme, you can type in "`theme 2`". This will change the picture on the right hand side to another picture we provided for you. Alternatively, you can find your own picture and put it into the user.dir folder so that you can use your own picture as the theme. Note that you have to crop it into size 383*418 and name it myTheme.png. You can then go back to Yui and type in "`theme my theme`" to change it to your theme. Note that you can always reset the theme to default using "`theme 1`" or "`theme default`".



```
2015.11.08 14:32:30
 Hello, my master. Welcome back.
 This is Yui!  <(^v^)/
 The events of today is shown on the right
 -What would you like to do?

2015.11.08 14:32:33
 Command Entered: theme 2
 Background is changed successfully!
```

Deadline

1	12:00	have lunch	Sun, 08/11/2015

Event

1	12:00-13:00	have lunch wi..	Sun, 08/11/2015

Memo Pad

1	date with girlfriend	2	have dinner with tu..

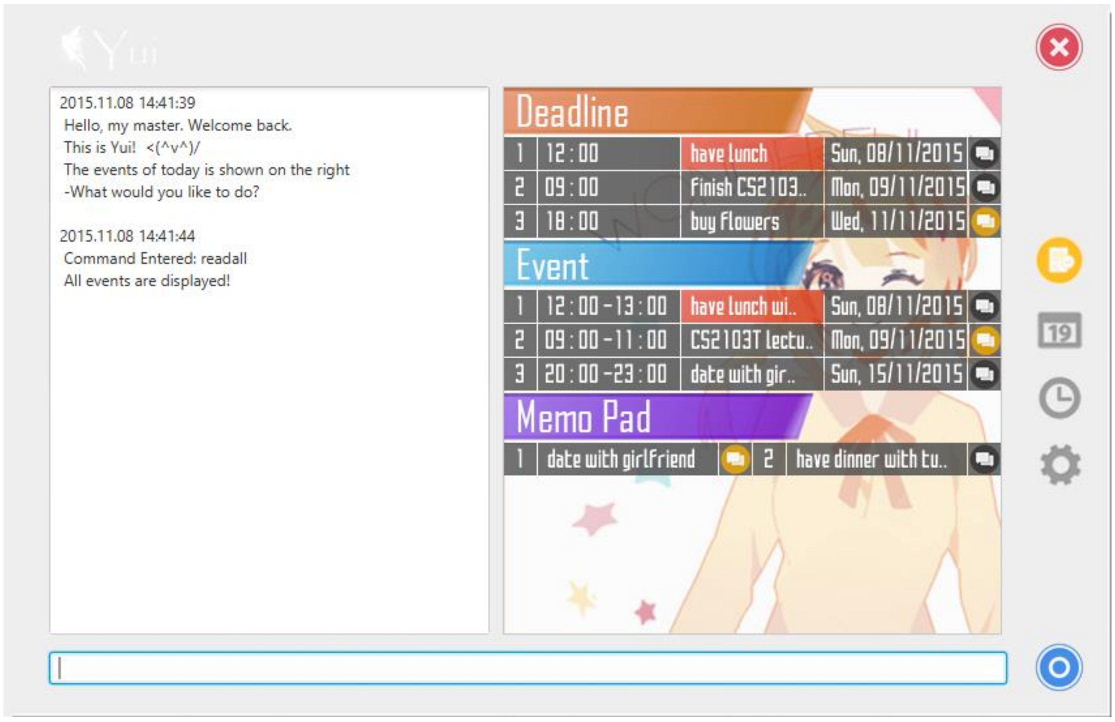
Figure 5.1. Response after "theme 2"

5.2 Recurring task

If you want to set up a task to happen repeatedly every few days, you can use the recurring task function. Simply type in "`recur <event type and index> <period of one iteration (how many days it happens once)> <number of iterations (how many times it will happen)>`". Yui will automatically set the date to the next occurring date if the current time has passed the deadline of a Deadline event, or the end time of an event with start and end time.

Examples:

- `recur d1 1 100`
- `recur e2 7 13`



```
2015.11.08 14:41:39
Hello, my master. Welcome back.
This is Yui!  <(^v^)/
The events of today is shown on the right
-What would you like to do?

2015.11.08 14:41:44
Command Entered: readall
All events are displayed!
```

Deadline

1	12:00	have lunch	Sun, 08/11/2015
2	09:00	Finish CS2103..	Mon, 09/11/2015
3	18:00	buy flowers	Wed, 11/11/2015

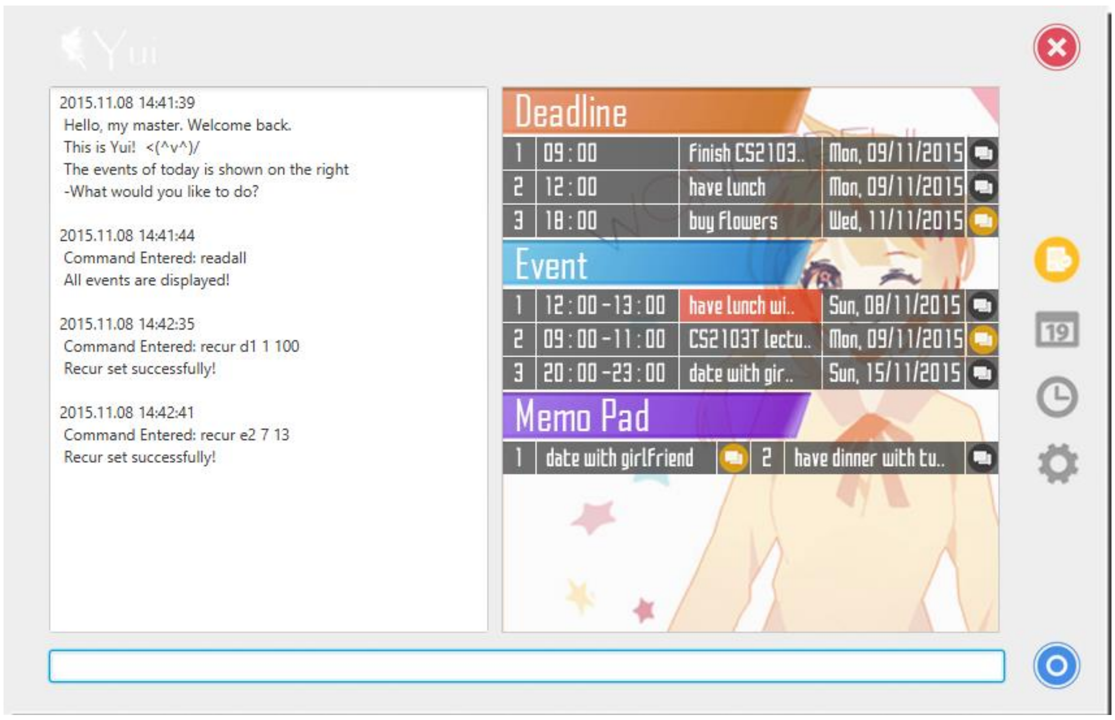
Event

1	12:00-13:00	have lunch wi..	Sun, 08/11/2015
2	09:00-11:00	CS2103T lectu..	Mon, 09/11/2015
3	20:00-23:00	date with gir..	Sun, 15/11/2015

Memo Pad

1	date with girlfriend	2	have dinner with tu..

Figure 5.2. Status before recurring



```
2015.11.08 14:41:39
Hello, my master. Welcome back.
This is Yui!  <(^v^)/
The events of today is shown on the right
-What would you like to do?

2015.11.08 14:41:44
Command Entered: readall
All events are displayed!

2015.11.08 14:42:35
Command Entered: recur d1 1 100
Recur set successfully!

2015.11.08 14:42:41
Command Entered: recur e2 7 13
Recur set successfully!
```

Deadline

1	09:00	Finish CS2103..	Mon, 09/11/2015
2	12:00	have lunch	Mon, 09/11/2015
3	18:00	buy flowers	Wed, 11/11/2015

Event

1	12:00-13:00	have lunch wi..	Sun, 08/11/2015
2	09:00-11:00	CS2103T lectu..	Mon, 09/11/2015
3	20:00-23:00	date with gir..	Sun, 15/11/2015

Memo Pad

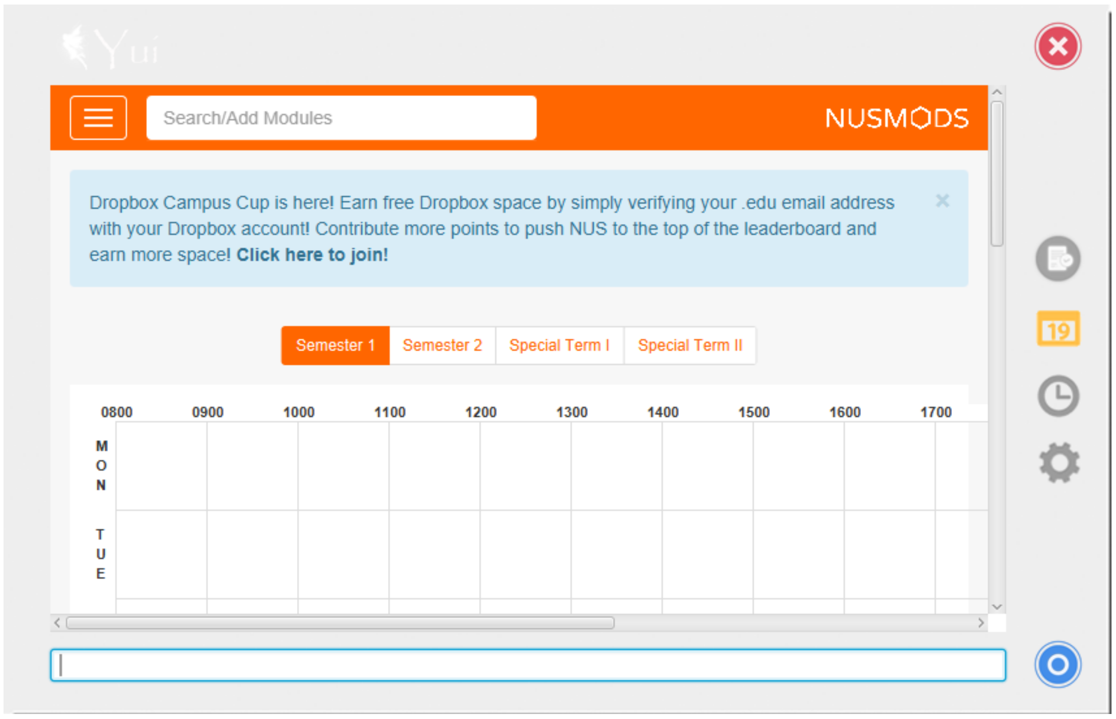
1	date with girlfriend	2	have dinner with tu..

Figure 5.3. Response after setting two recurring events

Note that the time for d1 has passed, hence the date has changed. Yui has also rearranged the events accordingly, so the deadline is now shown after another event which occurs earlier. The time for e2 has not reached and it remains the same. In addition, a “Memo Pad” event does not have time and hence cannot recur.

5.3 NUSMODS

If you are an NUS student, you can type in “`nusmods`” to quick check your timetable. However, this function only works when you are online. The right hand side icon will change to show which mode you are in. To go back to the main screen, type in “`todolist`”.

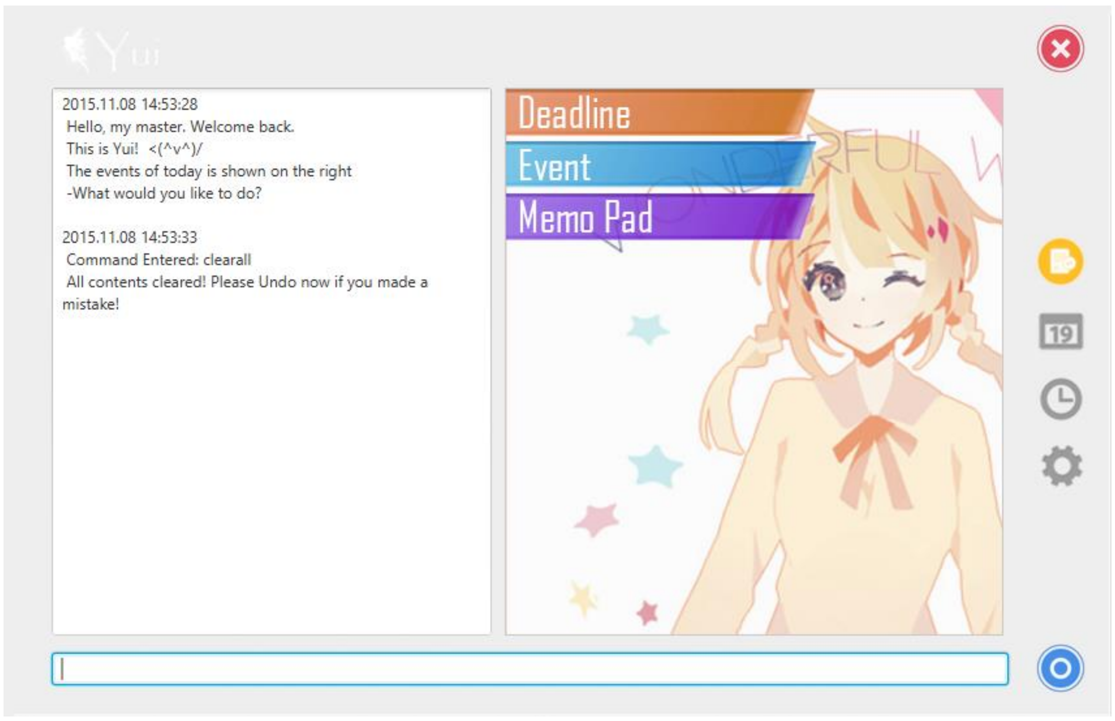


Dropbox Campus Cup is here! Earn free Dropbox space by simply verifying your .edu email address with your Dropbox account! Contribute more points to push NUS to the top of the leaderboard and earn more space! **Click here to join!**

Figure 5.4. Response after “nusmods”

5.4 Clearall

When you want to clear all the items in your list, just delete the data file. Alternatively, you can type in “`clearall`” to clear all the data. However, please “undo” at once if you made a mistake, otherwise you will not be able to get the file back!


2015.11.08 14:53:28
Hello, my master. Welcome back.
This is Yui! <(^v^)/
The events of today is shown on the right
-What would you like to do?

2015.11.08 14:53:33
Command Entered: clearall
All contents cleared! Please Undo now if you made a mistake!

Figure 5.5. Response after “clearall”

5.5 Hotkey

To make it easier for you to use Yui, we have set several hotkeys to increase your productivity.
You can use " ↑ " or " ↓ " arrows to move the list on the right up or down.
Press "Alt" and " ↑ " or " ↓ " arrows to move the message on the left up or down.
Press "Ctrl" and "H" to hide Yui.
Press "Ctrl" and "Y" to show Yui".

6 Help Sheet

You can find below a list of all the commands we provided as your reference. Alternatively, you can use "help" command in Yui to check the detailed format on using these commands!

Command	Description
add <event> <XX:XX> <DD/MM/YY>	Add a new event with deadline to the to-do list
add <event>	Add a new event without details
add <event> <XX:XX> <XX:XX> <DD/MM/YY>	Add a new event with specific do-time
mark	Mark event as done
comment <Your Comment>	Comment on users' current event
readall	Show all events
read today	Show all events falls on today
read tomorrow (tmr)	Show all events falls on tomorrow
setpath	set a new path to store user's list
update <type and index> <new content>	Update the details of users' events
undo	Undo the previous operation
redo	Redo the undo operation
delete <type and index>	Delete the event
help	Call the help menu to view all the commands available
outline	Show today and tomorrow outlines to users
recur <days> <times>	Create recurring tasks
search <Key Word>	Search for the event in your list
nusmods	Go to NUSMods website
clearall	delete everything from the list
exit	Exit Yui

7 Contact Us

If you ever encounter problems when using Yui, feel free to contact our developing team through email. The followings are the email addresses:

Liu Hanyang (A0133992@u.nus.edu)

Mao Chenyang (A0127142@u.nus.edu)

Le Nguyen (A0127821@u.nus.edu)

Justin Erh (A0125622@u.nus.edu)

8 Appendix A: User stories. As a user, I …

8.1 [Likely]

(For some functions in the user stories, we have not figured out a smart implementation. To avoid ambiguity, we did not put them in the user guide. However, we will update them in future versions.)

ID	I can … (i.e. Functionality)	so that I can … (i.e. Value)
create	add new event	record tasks that I want to do some day
read	display all events that I have created	read all the events I have created
undo	undo the last command	delete my last wrong input easily
redo	revert the undo command	redo my last command that I accidentally undo
delete	delete an event	delete events that I no longer want to keep track of
edit	edit information from existing event	modify or add more information to an event
search	search and display specific event	search of specific event easily
extraCommand	have various command words to execute the same command	input command in a shorter form or have an easier to remember command
readToday	display only events from that day	read only the events I have to do by

		today
comment	add extra comment for an event	add more extra information on an event
reminderType	have at least 3 set of basic reminder type for low, medium, high priority	assign various events with a more favorable reminder settings easily
customReminder	reminder type can be customized	further customize the reminder setting if I need to
import/export	store and transfer event data between 2 devices	transfer my event list between my laptop and desktop
mark	mark an event as done	turn off reminder for that event
help	display available commands	find the command input i need if I forget/ do not know
caseInsensitive	input command, search event without the need of correct case	not worry about case sensitivity for long commands and event names
recurTask	input recurring events	do not have to type in the same recurring event multiple times
clearUI	have a clear UI	see my events more clearly

8.2 [Unlikely]

ID	I can … (i.e. Functionality)	so that I can … (i.e. Value)
multilingual	input command and event name in Chinese/ different languages	have support for user who are not familiar with English
emoji	input emoji	express more feelings and expression toward certain events
specialSyntax	allow command input from special syntax	fasten the process of inputting command

9 Appendix B: Non Functional Requirements

1. The software should run on any device with Java Environment.
2. The software can respond to user input faster than a blink of the eye.
3. The software can be launched quickly (less than a second).
4. The software works perfectly even without Internet connection

10 Appendix C: Product survey

Product: Google Calendar **Documented by**: Le Nguyen Strengths: Mouse centric, 'drag and drop' design Offer easy color code for different events Friendly GUI Weaknesses: Only provide "1 hour before" reminder, user needs to set up reminder setting if they want to have a more specific reminder Not keyboard centric, lack good input command from keyboard Lack support for multi-day event (event that last between 10pm-2am next day) Limited offline functionality
Product: Google Keep **Documented by**: Mao Chenyang Strengths: Works both online and offline, fast response Captures events with different needs well Can be used without Internet Can mark something as done and archive it

Weaknesses: Cannot add quickly (does not support quick add) Short cut is not easy to set (no build-in setting of short cut)
Product: Google Calendar **Documented by**: Liu Hanyang Strengths: Support different languages Have a clear UI, which is convenient for users to use. With an account, users can check their calendars in any devices linked to the Internet. It is easy to share your calendar to friends It will send emails to users to remind what is going to happen It is easy to search the events in the calendar. Weaknesses: Must be used online, which people without Internet cannot use. It is linked with an account. If users lose their account, it cannot be used any more. If users only the website version, they may miss some reminder. Information on the Internet is not so safe.

Product: Fantastical 2 **Documented by:** Justin Erh

Strengths:

Natural language to enter events and reminders

Supports time zones

Available also for mobile devices – calendars can sync using a cloud service

GUI looks neat and clean

Reminder list (separate from events)

Localized in various (though not all) languages

Weaknesses:

Mac only - not for Windows

Not free and expensive

Mobile version is separate, requires extra payment

No Web version, just offline (with cloud sync) application

Developer Guide

Liu Hanyang (A0133992X)

Mao Chenyang (A0127142R)

Le Nguyen (A0127821J)

Justin Erh (A0125622N)

Contents

1 Introduction of Yui

Yui is an advanced task management software. It can be used to record your upcoming tasks and give you reminders when necessary. Users are able to control and use all the functionalities in this software just by using keyboard commands, such as adding tasks, searching tasks and setting the data path.

This developer guide serves to help new incoming developers who would like to understand the internal design of Yui and hope to extend the program with more advanced functions which they deem useful.

We will start with the overall architecture of Yui (Section 2) and proceed to functions of the major components of Yui (**Section 3**). After having a brief idea about Yui, you should refer to respective sub-sections in existing APIs (Section 4), coding demonstration (**Section 5**) and testing framework (**Section 6**) to make your code fit into Yui.

Now, let's get started!

2 Architecture

Here is a general view of the design of Yui. There are four major components in Yui: UI, Logic, Parser and Storage. Each component has its own interface. The main functions of each component is shown as follow:

a. UI Component: Controls the appearance of Yui
b. Logic Component: Handles entered user commands and modifies the event list
c. Parser Component: Parses the commands received and returns information to Logic
d. Storage Component: Saves and loads data files

You can refer to Figure 1 below for a better understanding.

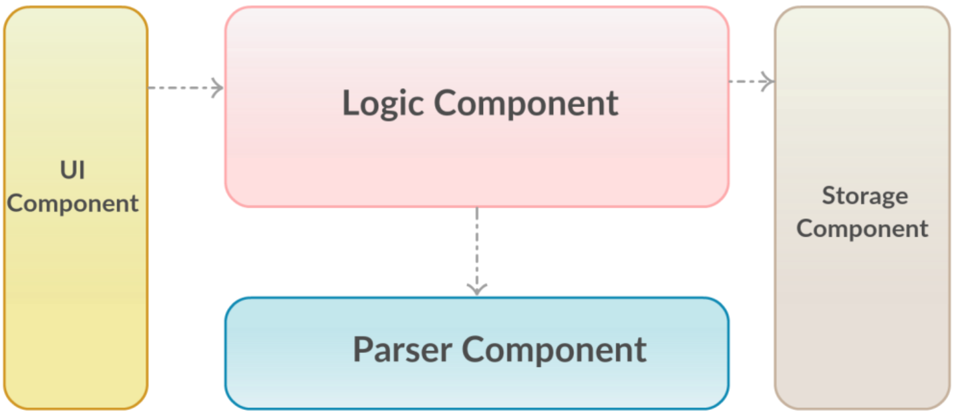


Figure 1: Software architecture of Yui

3 Components

3.1 Commands Flow

In Yui, all the operations are initiated by users. When users type in a new command in the user interface, it will be sent to the Logic components. The Logic component will then send the command to Parser component first and receive essential information from the Parser. After that, it will send the feedback message to the UI and save the event data into Storage component at the same time. The Storage component will save the data received into local file. It will also read data from the local file and pass it to the Logic component if the Logic component request to "show" the data. This process is reflected in the following diagram:

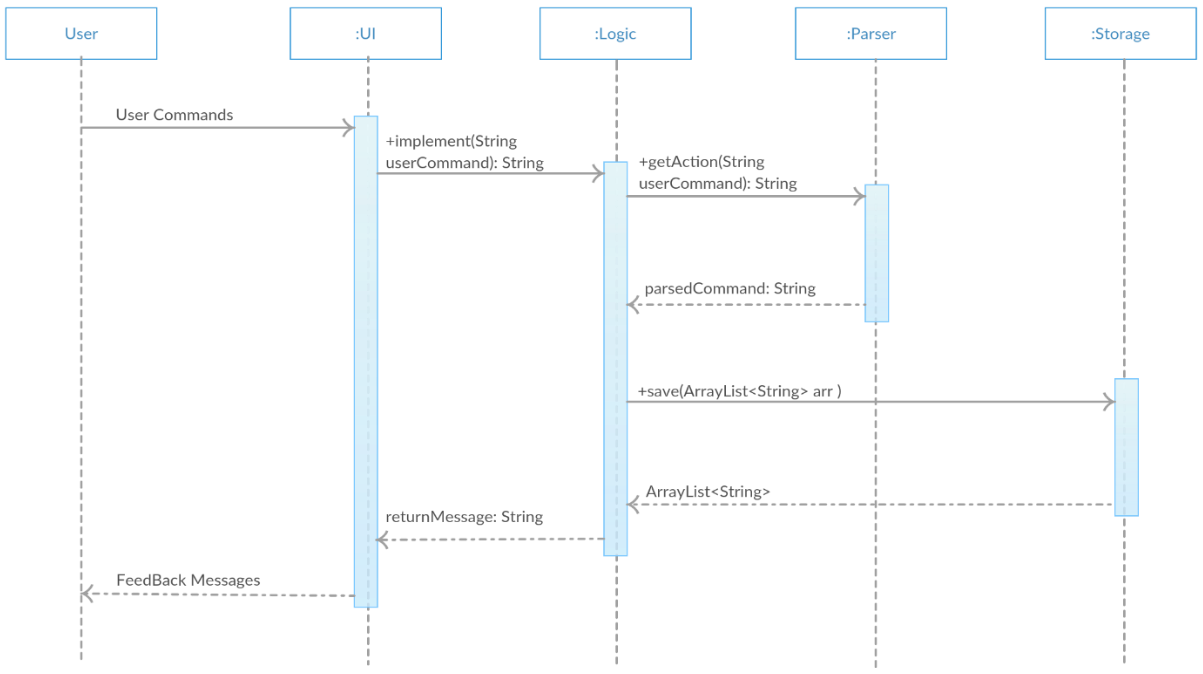


Figure 2: Sequence diagram for Yui to deal with new commands

3.2 UI

If you want to modify the interface or methods, you can go through this section.

GUI component is mainly written with the JavaFx API. We assume you are familiar with JavaFx. If not, you can refer to http://docs.oracle.com/javase/8/javase-clienttechnologies.htm to learn about JavaFx.

3.2.1 Classes Structure

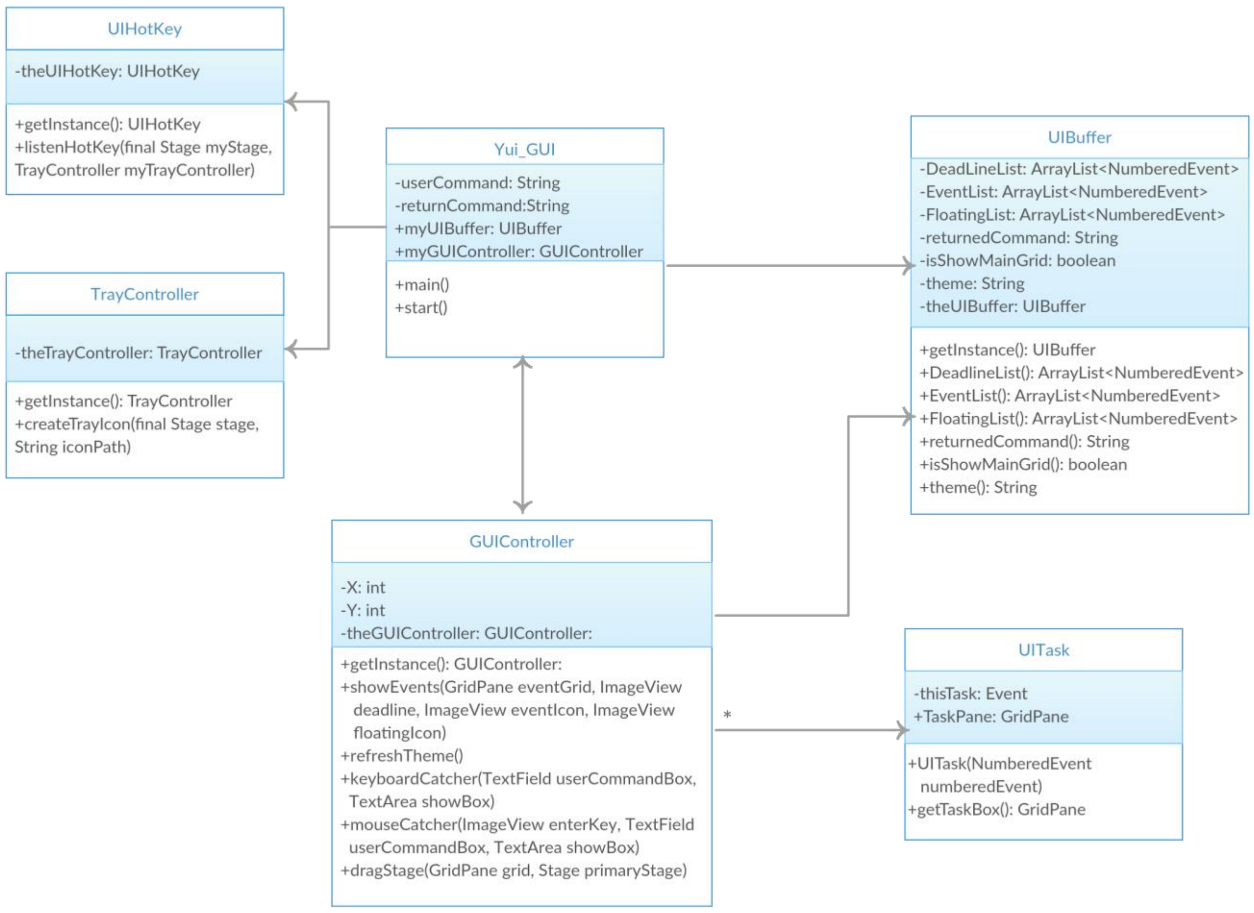

UIHotKey
-theUIHotKey: UIHotKey
+getInstance(): UIHotKey
+listenHotKey(final Stage myStage, TrayController myTrayController)
TrayController
-theTrayController: TrayController
+getInstance(): TrayController
+createTrayIcon(final Stage stage, String iconPath)
Yui_GUI
-userCommand: String
-returnCommand:String
+myUIBuffer: UIBuffer
+myGUIController: GUIController
+main()
+start()
UIBuffer
-DeadLineList: ArrayList<NumberedEvent>
-EventList: ArrayList<NumberedEvent>
-FloatingList: ArrayList<NumberedEvent>
-returnedCommand: String
-isShowMainGrid: boolean
-theme: String
-theUIBuffer: UIBuffer
+getInstance(): UIBuffer
+DeadlineList(): ArrayList<NumberedEvent>
+EventList(): ArrayList<NumberedEvent>
+FloatingList(): ArrayList<NumberedEvent>
+returnedCommand(): String
+isShowMainGrid(): boolean
+theme(): String
GUIController
-X: int
-Y: int
-theGUIController: GUIController:
+getInstance(): GUIController:
+showEvents(GridPane eventGrid, ImageView deadline, ImageView eventIcon, ImageView floatingIcon)
+refreshTheme()
+keyboardCatcher(TextField userCommandBox, TextArea showBox)
+mouseCatcher(ImageView enterKey, TextField userCommandBox, TextArea showBox)
+dragStage(GridPane grid, Stage primaryStage)
*
UITask
-thisTask: Event
+TaskPane: GridPane
+UITask(NumberedEvent numberedEvent)
+getTaskBox(): GridPane

Figure 3: Class diagram for UI component

3.2.2 Yui_GUI

This main class mainly describes how to draw the UI with codes. If you want to change the appearance of GUI, you just need to modify this class first. It gets data from `GUIController` and `UIBuffer` and shows it in the command box and to-do list box.

The methods `main()` and `start()` are just for launching the graphical interface.

3.2.3 GUIController

This class is used to control the appearance of UI with commands. It usually receives the handled results from `UIBuffer` and uses them to alternate the appearance of UI.

It can control the command box, the appearance of to-do list box and the position of the main stage. Also, it can capture the mouse and keyboard motion of the users. Then, it will handle the motion and give the appropriate response. For future details, you can refer Section 4 and 5.1.

3.2.4 UIBuffer

This class is used as a buffer for the data passed from the Logic component. It will save the received data first and then pass it to the `GUIController` and `Yui_GUI`. It is the interface which is used to connect the UI with Logic components.

3.2.5 UITask

This class is the constructor for a single task shown in the to-do list box. It will be called by the `GUIController` and save the single task data into this constructor.

It determines the type of the task, and then apply different structures for deadline, event, or floating task (named as memo pad in user guide). It will return the task box to the `GUIController` for showing.

3.2.6 UIHotKey

This class is used to set the global hot key for `Yui_GUI`. It only has one method called by Yui_GUI which is used to listen to the global hot key in UI. This class uses JIntellitype to carry out its functions.

If you want to learn how to install or use `JIntellitype`, you can refer to https://code.google.com/p/jintellitype/ .

3.2.7 TrayController

This class is used to add a tray icon and set actions when the stage of UI is hidden. It is called by `Yui_GUI` and set a tray icon. It will pop up a message when the UI is hidden.

3.3 Logic

This component is used to process the commands passed from UI.

3.3.1 Classes Structure

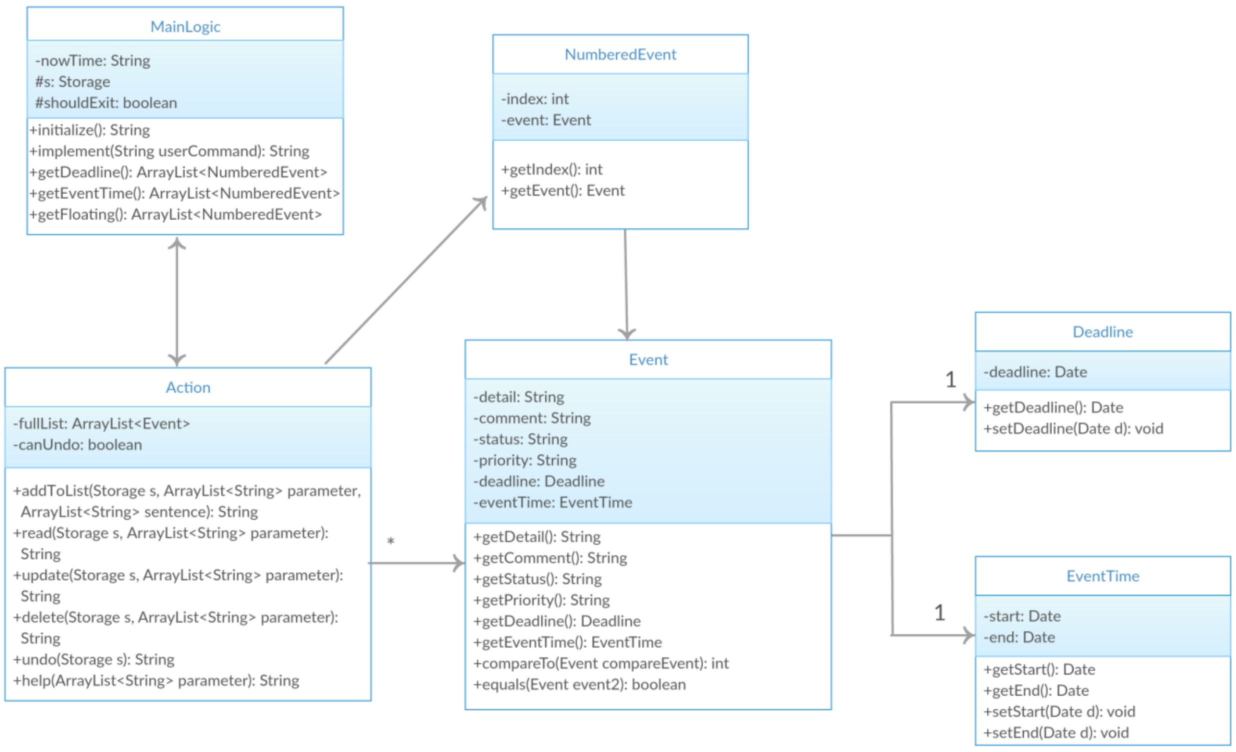

MainLogic
-nowTime: String
#s: Storage
#shouldExit: boolean
+initialize(): String
+implement(String userCommand): String
+getDeadline(): ArrayList<NumberedEvent>
+getEventTime(): ArrayList<NumberedEvent>
+getFloating(): ArrayList<NumberedEvent>
NumberedEvent
-index: int
-event: Event
+getIndex(): int
+getEvent(): Event
Action
-fullList: ArrayList<Event>
-canUndo: boolean
+addToList(Storage s, ArrayList<String> parameter, ArrayList<String> sentence): String
+read(Storage s, ArrayList<String> parameter): String
+update(Storage s, ArrayList<String> parameter): String
+delete(Storage s, ArrayList<String> parameter): String
+undo(Storage s): String
+help(ArrayList<String> parameter): String
*
Event
-detail: String
-comment: String
-status: String
-priority: String
-deadline: Deadline
-eventTime: EventTime
+getDetail(): String
+getComment(): String
+getStatus(): String
+getPriority(): String
+getDeadline(): Deadline
+getEventTime(): EventTime
+compareTo(Event compareEvent): int
+equals(Event event2): boolean
1
Deadline
-deadline: Date
+getDeadline(): Date
+setDeadline(Date d): void
1
EventTime
-start: Date
-end: Date
+getStart(): Date
+getEnd(): Date
+setStart(Date d): void
+setEnd(Date d): void

Figure 4: Class diagram for Logic component

3.3.2 MainLogic

This is the main class of the Logic component. It is used to initialize the storage or read the existing file and implement the user's commands.

This class will call `Parser` first when receiving a new command, and get the parsed command to `implement`.

The method `implement` will decide which operation to call. It has a switch to choose `add, show, delete, search, update, undo, redo, exit,` etc.

3.3.3 Action

This class is used to carry out various functions. When parsed commands are passed to `Action`, it will deal with the data saved. It will be called by `MainLogic` and return the messages to show operations whether are successful or errors are detected.

3.3.4 Event

This is an object that records the event details. It covers `detail, comment, status, priority, deadline` and `eventTime.` Also, it has methods to show these properties.

3.3.5 NumberedEvent

This is an object which contains both an index and an `Event`. It is usually used to tag index from the `Event` list that is retrieved from the `Storage`.

3.3.6 EventTime

This is an object used to record the start date and end date for the event. It also has methods to return these properties.

3.3.7 Deadline

This is an object that records the deadline date of the event, covering the date and time of the deadline.

3.4 Parser

3.4.1 Class Structure

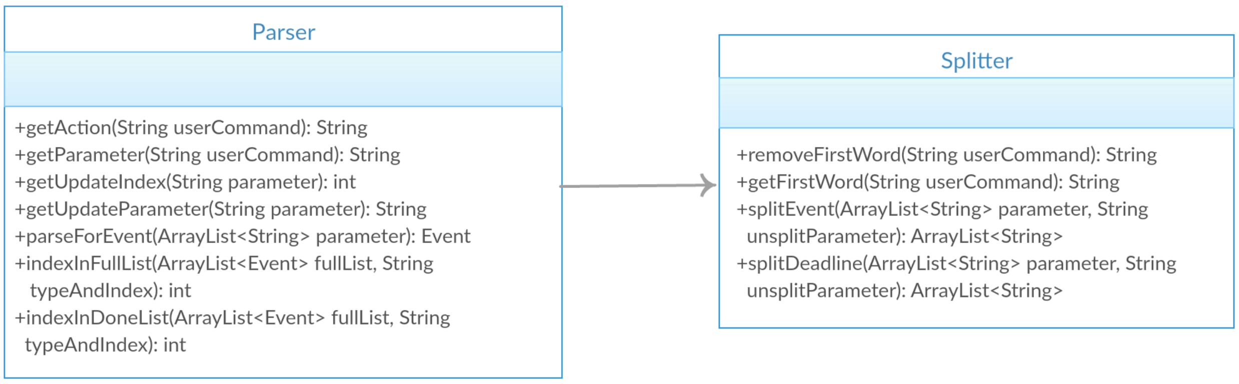

Parser
+getAction(String userCommand): String
+getParameter(String userCommand): String
+getUpdateIndex(String parameter): int
+getUpdateParameter(String parameter): String
+parseForEvent(ArrayList<String> parameter): Event
+indexInFullList(ArrayList<Event> fullList, String typeAndIndex): int
+indexInDoneList(ArrayList<Event> fullList, String typeAndIndex): int
Splitter
+removeFirstWord(String userCommand): String
+getFirstWord(String userCommand): String
+splitEvent(ArrayList<String> parameter, String unsplitParameter): ArrayList<String>
+splitDeadline(ArrayList<String> parameter, String unsplitParameter): ArrayList<String>

Figure 5: Class diagram for Parser component

3.4.2 Parser

This class is used to deal with the commands passed from the Logic component. It will pass the commands it receives to the `Splitter` and returns the results passed from `Splitter` to the Logic. It will parse a command as action and parameter.

This is the interface that connects Logic and Parser components.

3.4.3 Splitter

This class is used to separate the commands passed from the `Parser` into the action and any parameters associated with it. Then, it will return the two parts back to the `Parser`.

For example, for command “`add CS2101 tomorrow`”, the splitter will parse it and return “`add`” `and` “`CS2101 tomorrow`” to the `Parser`.

3.5 Storage

3.5.1 Class Structure

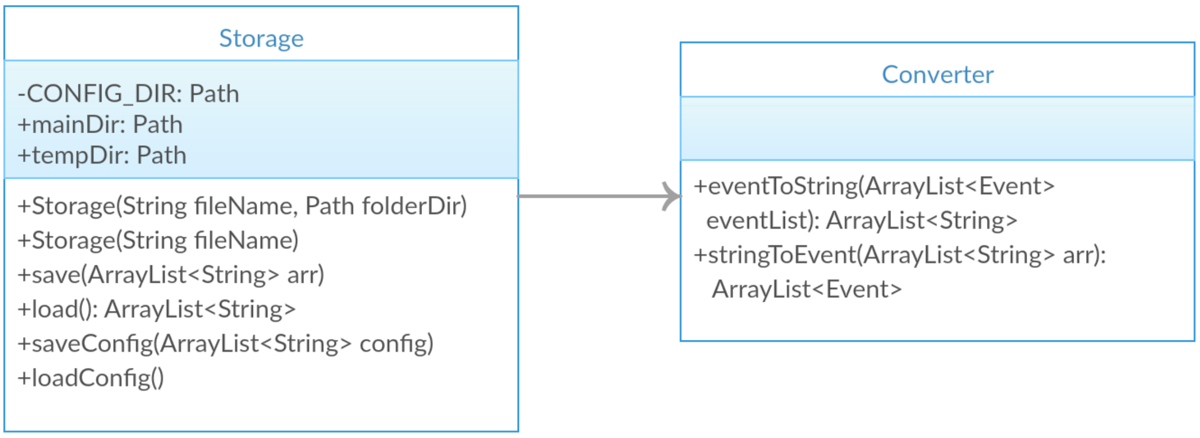

Storage
-CONFIG_DIR: Path
+mainDir: Path
+tempDir: Path
+Storage(String fileName, Path folderDir)
+Storage(String fileName)
+save(ArrayList<String> arr)
+load(): ArrayList<String>
+saveConfig(ArrayList<String> config)
+loadConfig()
Converter
+eventToString(ArrayList<Event> eventList): ArrayList<String>
+stringToEvent(ArrayList<String> arr): ArrayList<Event>

Figure 6: Class diagram for Storage component

3.5.2 Storage

This class is used to save and load the data file. It saves the ArrayList passed from the Logic component, and read the local data file as an ArrayList to pass the tasks to Logic component.

Also, it will load the "`config`" file which saves the data path, and the theme number saved when starting the software. It saves the "`config`" set by the users as well.

3.5.3 Converter

This class is used to convert the Event object to String or String to Event object. When the Storage saves or loads files, it will call the `Converter` to convert the data between String and Event.

4 How to use APIs

After reading the previous section, you should now have a basic understanding of the architecture and components of Yui. You can now start to work on code level and advanced features. Each component has set of public APIs (Application Programming Interfaces), which can be used in other components' classes.

In this section, we will show you some important API in each component. For all the APIs, you can refer the API library at the end of manual.

For UI developer,

you can use the APIs of GUI components through its controller class, `GUIController.java`. The following are the APIs of `GUIController`.

return type	description
void	showEvents(GridPane eventGrid, ImageView deadline, ImageView eventIcon, ImageView floatingIcon) refresh the tasks shown in the to-do list box after called
void	refreshTheme() refresh the background image of the to-do list box after called
void	keyboardCatcher(TextField userCommandBox, TextArea showBox) catch the keyboard motion of the users and make responses to both UIBuffer and GUI
void	mouseCatcher(ImageView enterKey, TextField userCommandBox, TextArea showBox) catch the mouse motion of enterKey of the users and send responses to both UIBuffer and GUI
void	dragStage() catch the mouse motion to the stage, and change the coordinate of the stage

For Logic developer,
you can use the APIs of Logic components through its main class, `ToDoList.java`. The following are the APIs of `ToDoList`.

return type	description
static String	initialize() initialize the storage file and path, then returns the welcome message
static String	implement() handle the commands received from the UI and return the corresponding messages
static ArrayList<String>	getFloating() get the list of floating events
static ArrayList<String>	getDeadline() get the list of events with deadlines
static ArrayList<String>	getEventTime() get the list of events with start time and end time

Note

To use the logic, you must call initialize() in GUI components before any other operations that handle commands.

For Parser developer,
you need to extract the required details from the commands that have been input by the user. The following is the public APIs of the `Parser class`.

return type	description
static String	getAction(String userCommand) parse the command and return the inputted action
static String	getParameter(String userCommand) parse the command and get the parameter of the pending event
static Event	parseForEvent(ArrayList<String> parameter) parse the parameters of the user command and save them into the Event object
static Event	getUpdateEvent(ArrayList<Event> fullList, ArrayList<String> parameter) get the Event from the fullList and parse the parameters of the updated user command. The Event will be updated with new parameters and returned.

For Storage developer,
you need to manage saving and loading of both the main file as well as the temporary file (backup of the file before the most recent edit to allow undoing the last operation) . You will also store the directory required for all the actions inside. The following is the public APIs of the `Storage class`.

Constructor
Storage(String filename, Path dire) Initialize storage class, generate a blank file with specified directory (recommended)

return type	description
void	save(ArrayList<String> arr) save the arrayList into the main directory

void	saveConfig(ArrayList<String> config) save the changes of the config list to the config data file
ArrayList<String>	load() read file from the main directory
ArrayList<String>	loadConfig() read the config file and return the config list which covers the save path and theme type

5 Start Coding

At this point, you should now know the overall design of Yui. You should have understood the structures and APIs used in Yui. Therefore, you can now start to extend your own codes! For different components, you can refer to different sections. You can read Section 5.1 for UI writing, 5.2 for Logic writing and 5.3 for Parser writing.

5.1 UI

Your main work is to pass the user commands to the Logic component and show any meaningful feedback on the screen. At the same time, the appearance of the to-do list will also be changed.

In UI component, we have applied the MVC (Model-View-Controller) pattern. The `Yui_GUI` shows the view of the software. The `GUIController` controls `Yui_GUI` and the `UIBuffer` stores the data for both `GUIController` and `Yui_GUI`.

If you want to extend the data needed for UI component, you can refer to `UIBuffer.java`. The data variables are located at the top of the class. For each data variable, there are two methods. The methods starting with "get" are used for getting data from the Logic component. The methods whose name is the same as the variables are used for passing data to `GUIController` and `Yui_GUI`. Then, you can add your new data variables in this format.

If you want to change the appearance of the UI, you can refer to `Yui_GUI.java` and `UITask.java`. `Yui_GUI` is mainly for the stable appearance of the UI. The methods starting with "set" are used for adding different components of the UI. You can change the appearance in this format: write a "set" method first and add it into the "start" method. The `UITask` is used for the appearance of a single task shown in the to-do list. The methods staring with "set" or "build" are used for setting the components in the task box. You can extends it in the similar format.

If you want to extend the motions for the UI, you can refer to `UIHotKey.java` and `GUIController.java`. GUIController controls both the showing of to-do list and the command box. To extend it, you can read the methods `show` and `motionCatcher` and add more functions in them. For `UIHotKey`, you can extends the `listenHotKey` to add more hot keys.

If you want to add more images and text fonts for UI, you can save them into the Image and Fonts folders.

5.2 Logic

Logic are used to process the commands passed from the UI component. Your work are to add more actions in the Logic component and add some parsing methods in the Parser component to support the new functions.

If you want to extend new functions in Logic, you should refer to `MainLogic.java` first. You need to add a new case in the `modify` method. Then, you need to refer to `Action.java`. You need to add new methods in this class to handle the commands passed from `MainLogic` and pass it to Parser. With the parsed commands, you can add other operations.

5.3 Parser

Parser is used to parse commands passed from Logic. Your work are to add more splitter methods in Parser and make it able to handle more flexible commands.

If you want to extend new splitting methods in Parser, you should refer to `Splitter.java`. In this class, you can write more methods to split the user commands into different parameters and pass them back to Logic. The Logic will then handle these parameters.

6 Testing

6.1 Manual Test

You can test the software manually in the System. Here are examples about how you can test with typing in the text box.

Test function	Command in text box	Input Data	Expected Result
Add event	add CS2101 tomorrow	String: "add CS2101 tomorrow"	Return "Event added successfully" and add an event in the data file. The to-do list is also changed.
Read event	Read m1	String: read m1	Return details of the first memo pad.

Delete event	delete m1	String: delete m1	Return “Delete successfully” and delete the event in the data file. The to-do list is also changed.
Undo operation	undo	String: undo	Return “Undo operation successfully” and undo the last operation
Search event	search CS2101	String: search CS2101	Show the tasks contains CS2101 in the to-do list
Redo operation	redo	String: redo	Return “Redo operation successfully” and redo the last undo operation
Help	help	String: help	Return the help list.
Exit the software	exit	String: exit	Exit the software

6.2 Automatic Test

6.2.1 Unit Test (White Box)

We have written the JUnit test for each component. You can refer to “/src/Junit test” and find the tests for them. To run these JUnit, you can choose Run as JUnit Test to run these tests. Here are the JUnit test we have:

Test Name	Function
SUTParser	It is used for the test for Parser component. You can read the different test case to check which methods it is test for this component.
SUTStorage	It is used for the test for Storage component. It can test whether the file is read successfully and whether the data is saved successfully.
ActionTest	It is used for the test for Logic component. It can test the methods in the Action.java. You can read the name of each test case to check which method it is testing.

6.2.2 Systemic Test (Black Box)

We also prepare a systemic test named as `SystemTest.java` for this software. These are JUnit tests which inputs a user’s commands and gets the feedback from the software system. It checks whether the feedback is the same as our expected feedback. However, it is a black box test - if there is a mistake, you will not be able to tell which component the error lies in.

To run the Systemic Test, the method is similar as the Unit Test. Until now, our systemic test can cover 80% of the codes excepted part of the UI.

If you add new functions for Yui, you can add a new test case in this class first. It should be named as test<Function> and contains an input String, feedback information and expected information.

7 Future Work

7.1 Apply Option Function in GUI

In the future, we plan to add an option pane in the GUI. If users have typed in "option", then it will hide the command box and to-do list, and show a new pane at the original position. In this option pane, it should show which theme the users are using, the save path they have selected, and the language that the program has been set to.

7.2 Improve the Parser

Currently, our Parser can only parse commands with specific formats. In the future, we plan to add more methods to make it able to parse more flexible commands.

7.3 Apply Multi-language Support

At the moment, we can only type in Chinese task names. In a future version of the program, we will add support for more languages and add multi-language feedback messages. The user can also type in multi-language commands.

7.4 Apply GUI Test

In this version, we do not have a complete GUI test. We will try to learn to use TextFx to write test cases for GUI.

8 Appendix - API Library

8.1 GUIController API

return type	description
void	showEvents(GridPane eventGrid, ImageView deadline, ImageView eventIcon, ImageView floatingIcon) refresh the tasks shown in the to-do list box after called
void	refreshTheme() refresh the background image of the to-do list box after called
void	keyboardCatcher(TextField userCommandBox, TextArea showBox) catch the keyboard motion of the users and make responses to both UIBuffer and GUI
void	mouseCatcher(ImageView enterKey, TextField userCommandBox, TextArea showBox) catch the mouse motion of enterKey of the users and make responses to both UIBuffer and GUI
void	dragStage() catch the mouse motion to the stage, and change the coordinate of the stage

8.2 UITask API

return type	description
GridPane	getTaskBox() return the task box pane shown in the to-do list box. It contains the information of a single task.

8.3 UIBuffer API

return type	description
void	getFeedBack(String userCommand) pass the userCommand to Logic and save the returned command in UIBuffer
String	returnedCommand() return the returned command saved in UIBuffer

void	getList() get the task list from the Logic and save it in UIBuffer
ArrayList<Numbered Event>	DeadlineList() return the deadline list saved in the UIBuffer
ArrayList<Numbered Event>	EventList() return the event list saved in the UIBuffer
ArrayList<Numbered Event>	FloatingList() Return the floating task list saved in the UIBuffer
void	getTheme() get the theme type from the Logic and save it in UIBuffer
String	theme() return the theme type saved in UIBuffer
void	getIsShowMainGrid() get the boolean controlled the showing of main grid from Logic and save it in the UIBuffer
boolean	isShowMainGrid() return the boolean saved in the UIBuffer

8.4 UIHotKey API

return type	description
void	listenHotKey(final Stage myStage, final TrayController myTrayController) listen to the global hot key and gives a response to the UI showing

8.5 TrayController API

return type	description
void	createTrayIcon(final Stage stage, String iconPath) create the tray icon and make response to the UI showing

8.6 MainLogic API

return type	description
static String	initialize() initialize the storage file and return welcome message

static String	implement() handle the commands received from UI and return the corresponding messages
static ArrayList<String>	getFloating () get the list of floating (memo) events
static ArrayList<String>	getDeadline () get the list of events with deadlines
static ArrayList<String>	getEventTime () get the list of events with start time and end time
static boolean	getIsShow() get the current showing screen, namely nusmods or the todolist
static String	getTheme() get the theme name chosen by user

8.7 Action API

return type	description
static String	addToList(Storage s, ArrayList<String> parameter, ArrayList<String> sentence) add the parameter into the list and return corresponding messages
static void	readAll(Storage s) show the full list on the right hand side panel
static void	exit() exit the software
static String	bground(Storage s, ArrayList<String> parameter) change the background according to the parameter and return corresponding messages
static String	searchKey(Storage s, ArrayList<String> parameter) get the result of the search for a certain parameter and return corresponding messages
static String	update(Storage s, ArrayList<String> parameter) update a certain event, save it into the data file and return corresponding

	messages
static String	deleteEvent(Storage s, ArrayList<String> parameter) delete a certain event from the date file and return corresponding messages
static String	undo(Storage s) undo the last operation
static String	redo(Storage s) redo the last undo operation
static ArrayList<NumberedEvent>	getDeadlineList() get the list of Deadline event from current showing events
static ArrayList<NumberedEvent>	getEventTimeList() get the list of EventTime event from current showing events
static ArrayList<NumberedEvent>	getFloatingList() get the list of Floating event from current showing events
static String	read(Storage s, ArrayList<String> parameter) get the details about a specific event or events on a specific day according to the parameter and return corresponding messages
static String	outline(Storage s, ArrayList<String> parameter) get the list of events that fall on today and tomorrow, and return corresponding messages
static String	comment(Storage s, ArrayList<String> parameter) set comment for the specific event according to the parameter and return corresponding messages
static String	recur(Storage s, ArrayList<String> parameter) set recur for the specific event according to the parameter and return corresponding messages
static String	mark(Storage s, ArrayList<String> parameter) mark the specific event according to the parameter and return corresponding

	messages
static String	nusmods() get the main screen show nusmods and return corresponding messages
static String	todolist() get the main screen show todolist and return corresponding messages
static boolean	getIsShow() return whether it is now showing nusmods
static String	clearAll(Storage s, ArrayList<String> parameter) delete all content in the list and return corresponding messages
static void	readMark(Storage s) have the list of all marked events shown
static String	unmark(Storage s, ArrayList<String> parameter) unmark the specific event according to the parameter and return corresponding messages
static String	setpath(Storage s, ArrayList<String> parameter) set the data storage path according to the parameter and return corresponding messages
static String	help(ArrayList<String> parameter) return the help message according to the parameter given

8.8 Event API

return type	description
String	getDetail() get the details of the event
String	getComment() get the comment of the event
String	getStatus() get the status description of the events

String	getPriority() get the priority (recurring status) of the event
Deadline	getDeadline() get the deadline the event
EventTime	getEventTime() get the start and end time of the event
void	setDetail() set the details of the event
void	setComment() set the comment of the event
void	setStatus() set the status description of the events
void	setPriority() set the priority (recurring status) of the event
void	setDeadline() set the deadline the event
void	setEventTime() set the start and end time of the event
int	compareTo(Event compareEvent) override and returns int according to which Event is earlier
boolean	equals(Event event2) override and returns whether the two Events are the same Event

8.9 Deadline API

return type	description
Date	getDeadline() return the time as a Date
void	setDeadline(Date d) set the time as the input d

8.10 EventTime API

return type	description

`Date`	`getStart()`
	`return the time as a Date`

`Date`	`getEnd()`
	`return the time as a Date`

`void`	`setStart(Date s)`
	`set the time as the input s`

`void`	`setEnd(Date e)`
	`set the time as the input e`

8.11 NumberedEvent API

return type	description

`int`	`getIndex()`
	`return the index of the NumberedEvent`

`Event`	`getEvent()`
	`return the Event stored in the NumberedEvent`

8.12 Parser API

return type	description

`static String`	`getAction(String userCommand)`
	`parse the command and return what the action is`

`static ArrayList<String>`	`getParameter(String userCommand)`
	`parse the command and get the parameter of the pending event`

`static ArrayList<String>`	`getSentence(String userCommand)`
	`parse the command and return the parameter of the pending event. This is used to deal with invalid input`

`static int`	`getUpdateIndex(ArrayList<Event> fullList, ArrayList<String> parameter)`
	`return the index to update within the fullList`

`static Event`	`getUpdateEvent(ArrayList<Event> fullList, ArrayList<String> parameter)`
	`return the Event used to replace the old Event`

`static Event`	`parseForEvent(ArrayList<String> parameter)`
	`return an Event according to the parameter`

static int	indexInFullList(ArrayList<Event> fullList, String typeAndIndex) return the index of that Event in the list that are not marked done
static int	indexInDoneList(ArrayList<Event> fullList, String typeAndIndex) return the index of that Event in the list that are marked done

8.13 Splitter API

return type	description
static String	removeFirstWord(String userCommand) remove the first word and return the remaining String
static String	getFirstWord(String userCommand) return the first word of the userCommand
static ArrayList<String>	splitEvent(ArrayList<String> parameter, String unsplitParameter) return the ArrayList according to the format of an EventTime event
static ArrayList<String>	splitDeadline(ArrayList<String> parameter, String unsplitParameter) return the ArrayList according to the format of a Deadline event

8.14 Storage API

return type	description
void	save(ArrayList<String> arr) save the arrayList of String into the main directory
void	saveE(ArrayList<Event> arr) save the arrayList of Event into the main directory
void	saveConfig(ArrayList<String> config) save the current configuration (theme, data store path)
ArrayList<String>	loadConfig() load and return the current configuration (theme, data store path)
void	saveAs(ArrayList<String> arr, Path dir) save the arrayList of String into the main directory also change main directory to the new Path

void	saveAsE(ArrayList<Event> arr, Path dir) save the arrayList of Event into the main directory also change main directory to the new Path
ArrayList<String>	load() read String file from the main directory
ArrayList<Event>	loadE() read Event file from the main directory
ArrayList<String>	load(Path dir) read String file from the specified directory
ArrayList<Event>	loadE(Path dir) read Event file from the specified directory
ArrayList<String>	loadAs(Path dir) read String file from the specified directory also change main directory to the new Path
ArrayList<Event>	loadAsE(Path dir) read Event file from the specified directory also change main directory to the new Path
Path	mainDir() return the directory for the main file
Path	tempDir() return the directory for the temporary file
void	reset() delete the data file in the main directory

8.15 Converter API

return type	description
static ArrayList<String>	eventToString(ArrayList<Event> eventList) convert the ArrayList of Event to ArrayList of String
static ArrayList<Event>	stringToEvent(ArrayList<String> arr) convert the ArrayList of String to ArrayList of Event