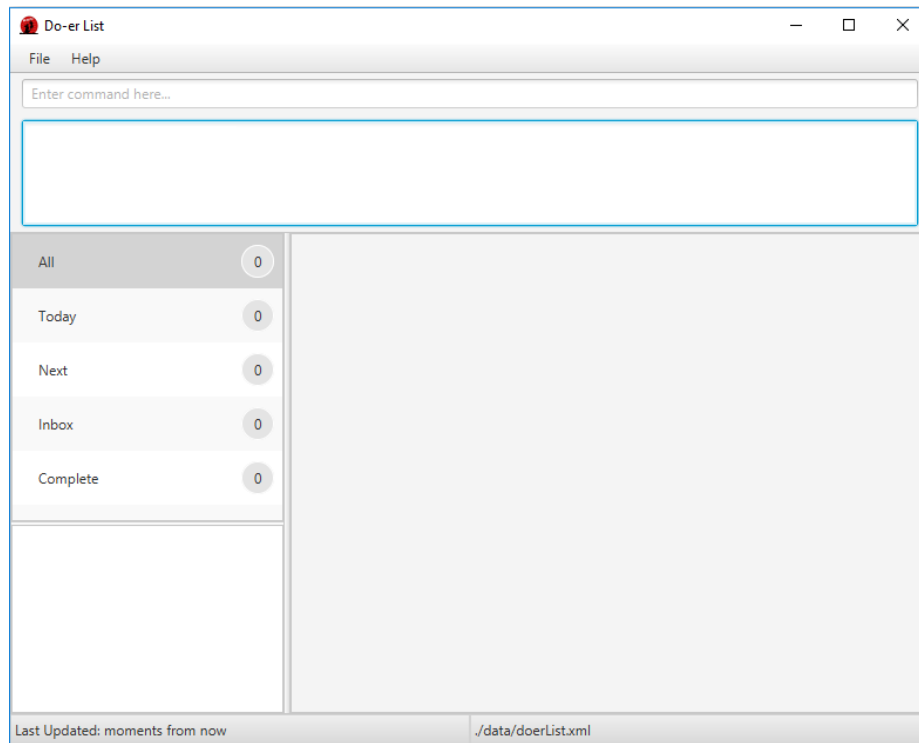
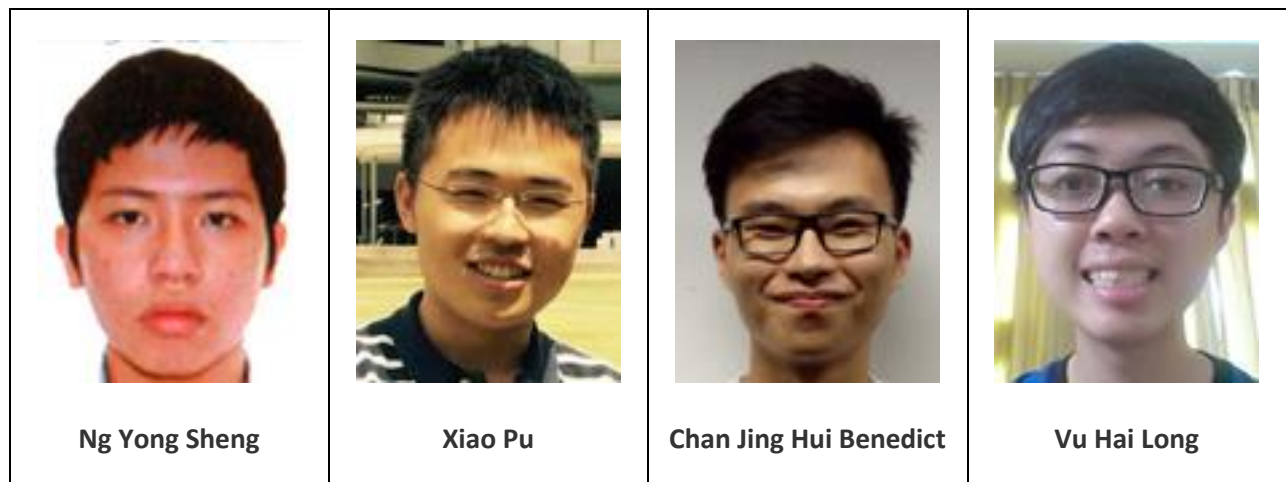


## DO-er LIST



**Supervisor: Si Junke Ashley**



## ***Table of Contents***

---

Acknowledgements .....	4
About Us .....	5
Supervisors.....	5
Project Team.....	6
User Guide.....	10
About Us .....	11
Overview .....	12
Getting Started .....	13
Features .....	15
FAQ .....	33
Command Summary .....	34
Developer Guide .....	35
Setting up .....	37
Design .....	38
Architecture .....	38
UI component.....	41
Logic component .....	43
Model component .....	44
Storage component .....	46

## [W09-C4] [Do-er List]

Common classes .....	47
Implementation.....	48
Logging .....	48
Configuration .....	48
Testing .....	49
Dev Ops.....	51
Build Automation .....	51
Continuous Integration.....	51
Making a Release .....	51
Managing Dependencies .....	51
Appendix A : User Stories.....	52
Appendix B: Use Cases .....	55
Appendix C: Non-Functional Requirements .....	67
Appendix D: Glossary.....	67
Appendix E: Product Survey .....	68
Review of TickTick: .....	68
Review of WunderList.....	69
Review of Trello.....	70
Review of Google Calendar .....	71

## ***Acknowledgements***

---

- Certain aspects of this application were inspired by Marco Jakob's Java FX tutorial.
- This program utilizes Natty Time, Pretty Time, Test FX, Guava and Jackson packages.
- This program is based off Address Book Level 4's sample code.

**Licence:** [MIT](#)

## ***About Us***

---

We are CS2103/T team (Wednesday 09 C4) based in the School of Computing, National University of Singapore.

## ***Supervisors***

---

**Damith C. Rajapakse**



**Role:** Module Coordinator

**Si Junke Ashley**



**Role:** Project Advisor

## ***Project Team***

---

### **Xiao Pu**



- Components in charge of: [UI](#)
  - Aspects/tools in charge of: Testing, Integration, JavaFX
  - Features implemented:
    - [List Tasks](#)
    - [Find Tasks](#)
    - [View Tasks](#)
    - [Delete Tasks](#)
  - Code written: [\[functional code\]](#)[\[test code\]](#)[\[docs\]](#)
  - Other major contributions
    - Design UI with [Yong Sheng](#)
    - Set up Gradle, Travis, Coveralls and Codacy
    - Help other writing UI parts and GUI test cases for the features they implement.
    - Update Developer Guide to match the project at the end
-

## Yong Sheng



- Components in charge of: [UI](#)
  - Aspects/tools in charge of: Guides, Balsamiq, Markdown
  - Features implemented:
    - [Add Recurring Tasks](#)
    - [Help Command](#)
    - [Exit Program](#)
  - Code written: [\[functional code\]](#)[\[test code\]](#)[\[docs\]](#)
  - Other major contributions
    - Design UI with [Xiao Pu](#)
    - Help to design and improve the program interface and guides to be more user-friendly
    - Refactors and spot for errors in code comments
-

## Benedict Chan



- Components in charge of: Storage
  - Features implemented:
    - [Mark Tasks](#)
    - [Unmark Tasks](#)
  - Code written: [\[functional code\]](#)[\[test code\]](#)[\[docs\]](#)
  - Other major contributions
    - Collating the UserGuide.md and DeveloperGuide.md with [Yong Sheng](#)
-



## Vu Hai Long



- Components in charge of: [Logic](#)
  - Aspects/tools in charge of: Testing
  - Features implemented:
    - [Add Tasks](#)
    - [Edit Tasks](#)
    - [Help Function](#)
    - [Task Due](#)
  - Code written: [\[functional code\]](#)[\[test code\]](#)[\[docs\]](#)
  - Other major contributions
    - Do the original refactoring.
- 
-

## ***User Guide***

---

About Us .....	11
Overview.....	12
Getting Started .....	13
Features .....	15
FAQ .....	33
Command Summary.....	34

## ***About Us***

---

Living in the modern and fast-paced world nowadays, we are constantly overwhelmed with tasks to do. Many people face the problem of time management as the traditional methods prove to be little effective. That is what Do-er List aims to solve.

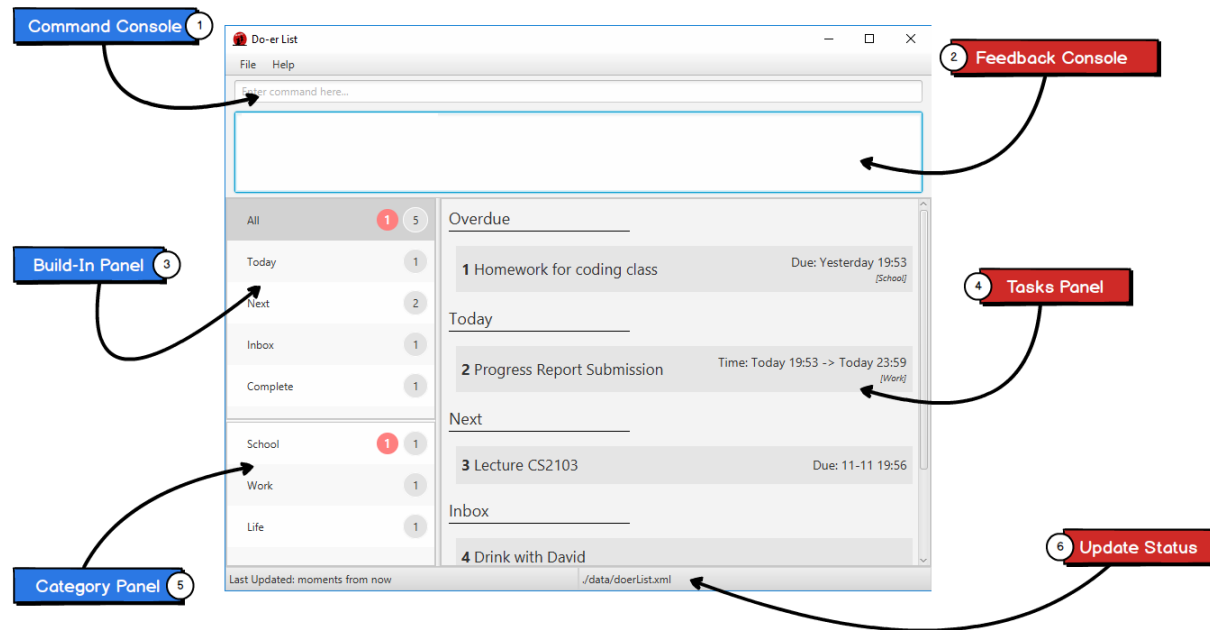
Do-er List is a task manager that is designed for students and office workers. It is a beginner-friendly desktop program that aids you in the planning and completion of your daily tasks. It does not matter if you are planning a big birthday surprise event or recurring task of handling the laundry every now and then, Do-er List is here to resolve your problems.

This user guide aims to allow any user to seamlessly use the product, as intended to. Just follow the instructions as stated and you will get the results you desire.

Eager and excited? Then let us proceed!

## [W09-C4] [Do-er List]

### Overview



Labels	Description
1	Type your commands in the <b>Command Console</b> to execute the desired commands.
2	<b>Feedback Console</b> shows if your command is executed properly or not.
3	View your build-in categories in the <b>Build-In Panel</b> .
4	<b>Tasks Panel</b> will display all the tasks you listed in a panel.
5	<b>Category Panel</b> shows all the custom categories that you have created.
6	View your last update and file storage in <b>Update Status</b> .

## Getting Started

---

Do-er List makes the process of adding, editing or deleting tasks a seamless process. Long gone are the days when you need to type in long and complicated commands or endlessly click on multiple buttons to get what you desire.

All commands have this standard format:

`Command required_fields [optional_fields] ...`

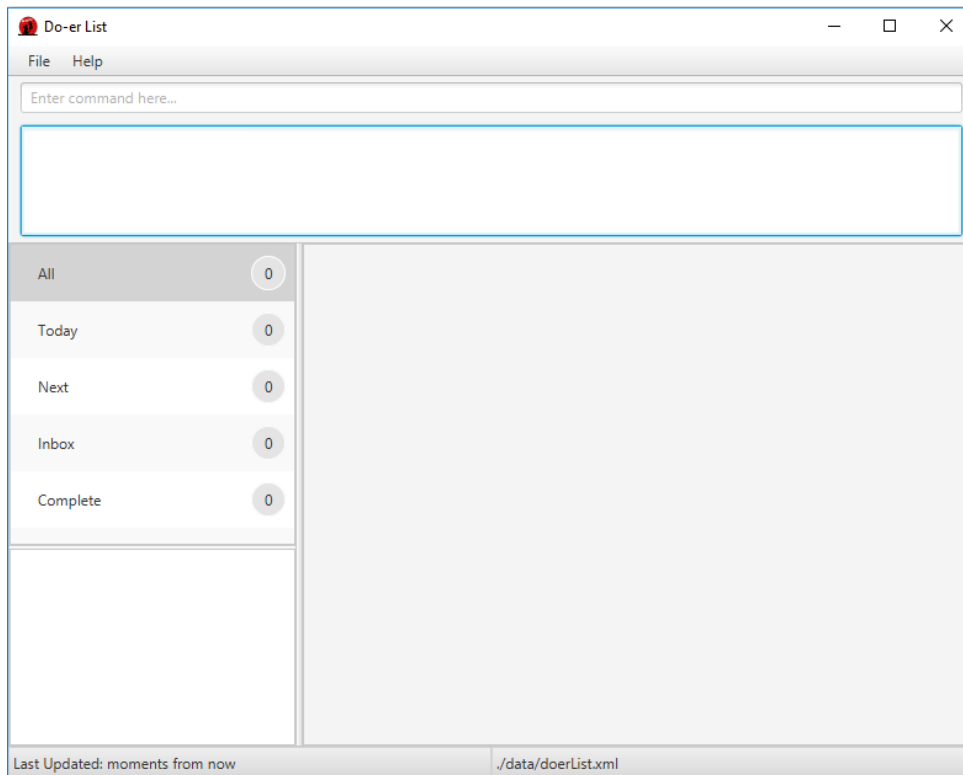
All commands start with a command words, followed by fields that are replaced by your inputs. The fields in the square bracket "[" and "]" are optional. You can choose to not include these fields.

*Please take note of the following:*

1. Ensure you have Java version 1.8.0\_60 or later installed in your Computer.
  - > This app will not work with earlier versions of Java 8.
2. Download the latest `doerlist.jar` from the [releases](#) tab.
3. Copy the file to the folder you want to use as the home folder for the Do-erList.

## [W09-C4] [Do-er List]

4. Double-click the file to start the app. The GUI should appear in a few seconds.



5. Type the command in the command box and press `Enter` to execute it.
- e.g. typing `help` and pressing `Enter` will open the User Guide in a new window.
6. Some example commands you can try:
- `add/t Do post-lecture quiz /s today 10:00 /e tomorrow 12:00 /c CS2103:` adds a task called Do post-lecture quiz to the Do-er List that starts today at 10:00 and ends tomorrow at 13:00 under the category CS2103.
  - `list CS2103:` lists all tasks in **CS2103**
  - `delete 1:` deletes the 1st task shown in the current list
  - `exit:` exits the application
7. Refer to the Features section below for details of each command.

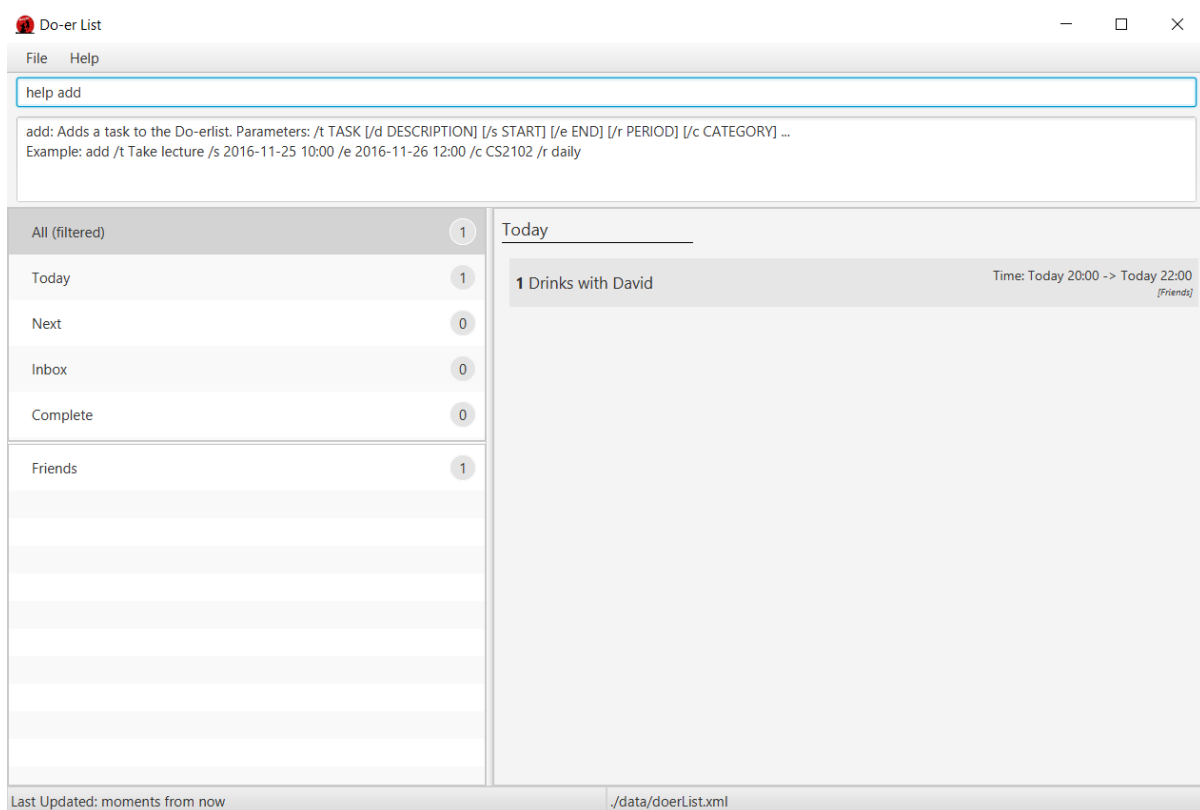
## Features

### Command Format

- Words in **UPPER\_CASE** are the parameters.
- Items in **SQUARE\_BRACKETS** are optional.
- Items with ... after them can have multiple instances.
- If a command has multiple parameters, the order of parameters doesn't matter.

### Viewing help: help

Format: help or help [COMMAND\_NAME]



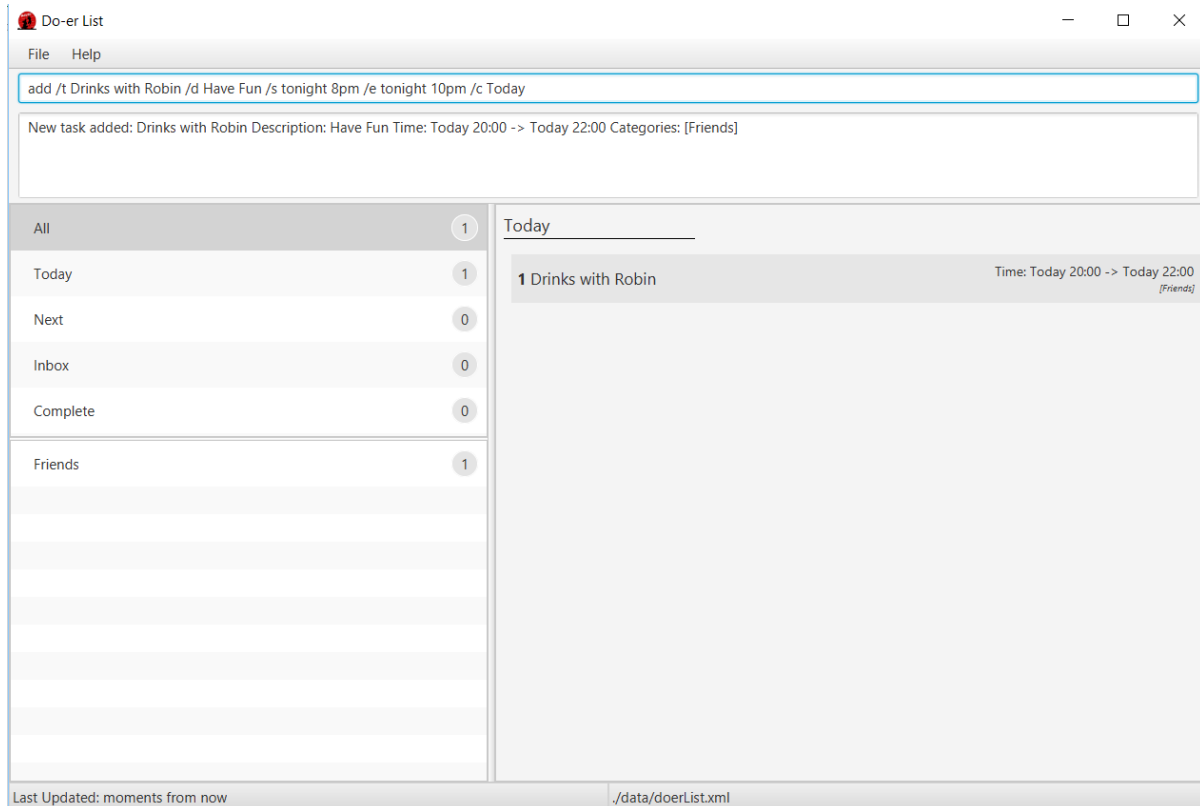
If the **COMMAND\_NAME** is supplied, it will display the instructions of using that command. Help is also shown if you enter an incorrect command e.g. abcd

## [W09-C4] [Do-er List]

### Adding a task: add

Adds a task to the Do-erList

Format: add /t TITLE [/d DESCRIPTION] [/s START] [/e END] [/c CATEGORY]  
[/r RECURRING] ...



- You can choose to have no categories or any number of them in your task.
- The **START** or **END** parameter supports natural language input.
- **RECURRING** intervals can also be added in and supports in natural language input.
- If the **START** date is missing, the Doer-List set it to its current timing and date by default.
- If there is no **START** or **END** parameters, the Doer-list will create task without start date and end date and move it to **Inbox** default category.
- Supported list for natural language for **START** and **END**:
  - next X hours / days / weeks / months
    - X can be any number: 1, 2, 3, ...



## [W09-C4] [Do-er List]

- today
  - tomorrow
  - next week / month
- Supported list for natural language for **RECURRING** tasks:
  - daily
  - weekly
  - monthly
  - yearly

Examples:

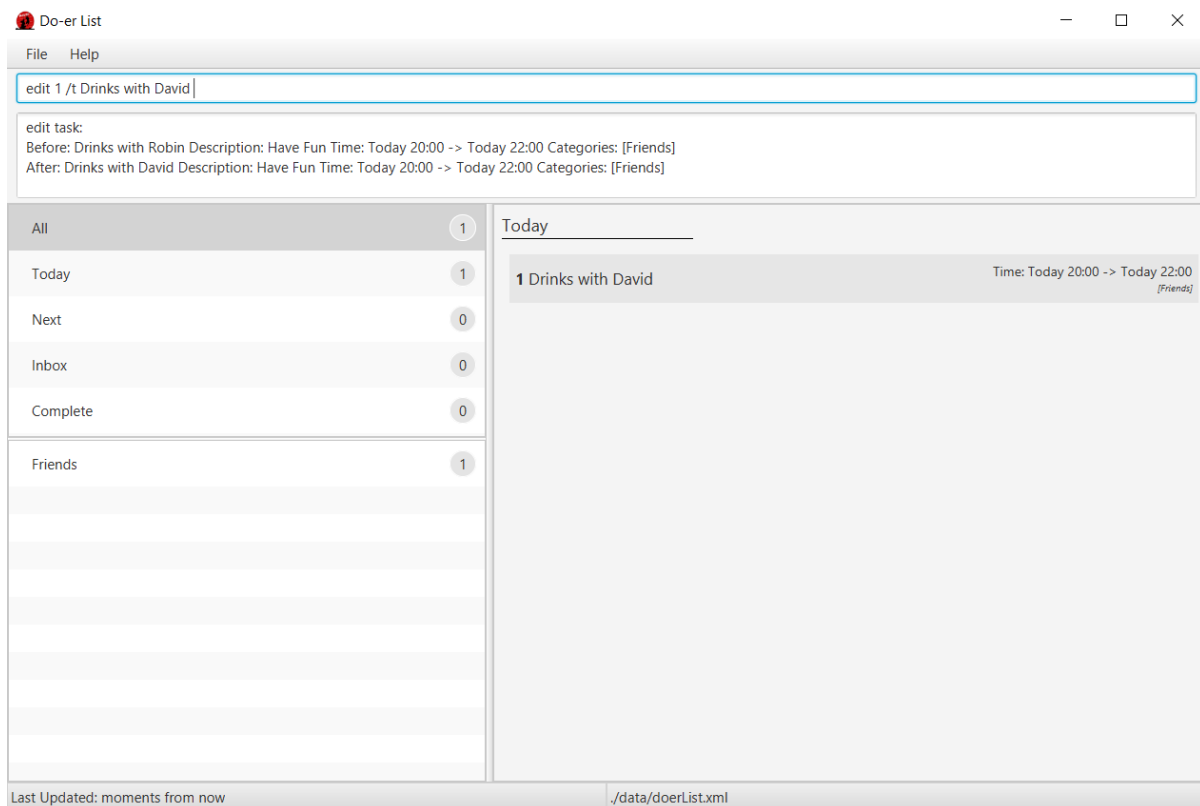
- add /t Weekly Laundry /s 2016-11-23 21:00 /c Chores
- add /t Daily Exercise and Workout!
- add /t Call Mum in Hanoi /d Limit chat timing for overseas charges /s tomorrow 8pm /e tomorrow 10pm /c Optional
- add /t Check Email /s tomorrow 8pm /e tomorrow 10pm /r daily

## [W09-C4] [Do-er List]

### Editing a task: `edit`

Edit an existing task in the Do-er List

Format: `edit INDEX [/t TITLE] [/d DESCRIPTION] [/s START] [/e END] [/c CATEGORY] [/r RECURRING] ...`



Edit an existing task by calling its index. The task's title, description, start date, end date, recurring intervals and categories can be edited.

Examples:

- `edit 2 /t Do the Laundry /c Chores /r Daily`
- `edit 3 /c Do Homework`
- `edit 3 /s tomorrow 23 00`

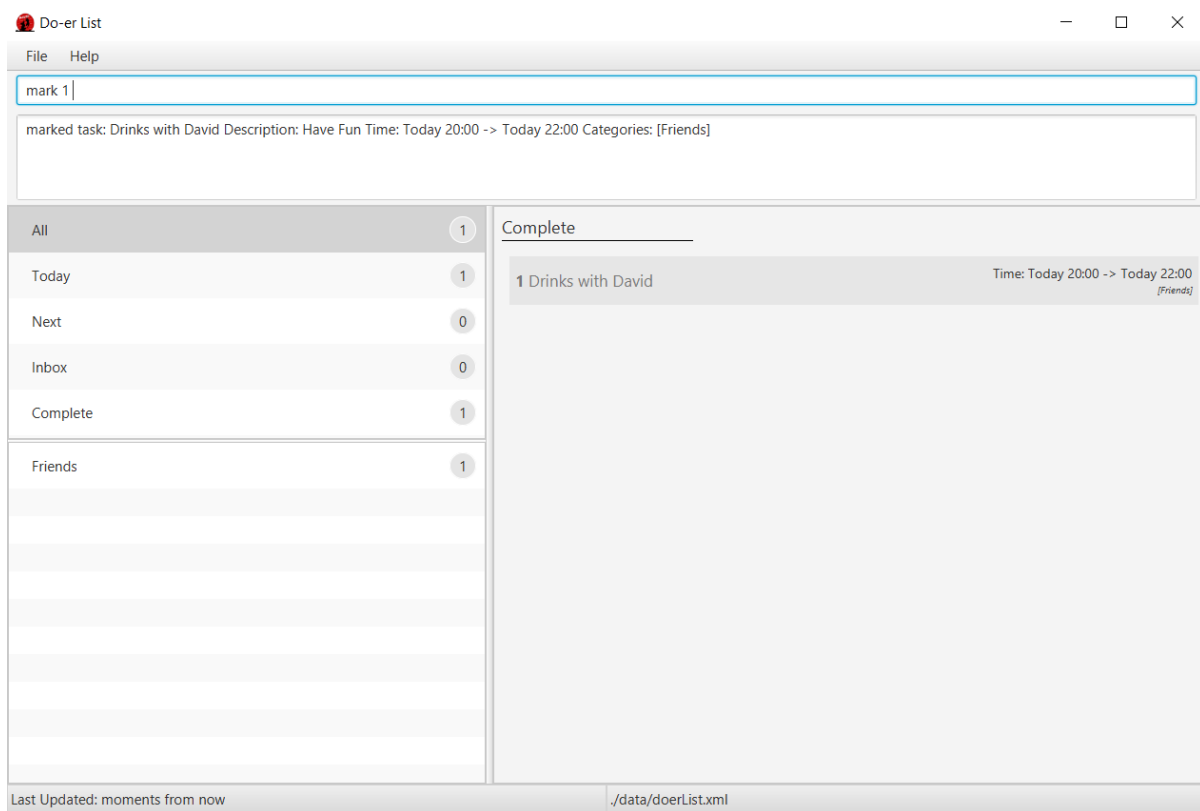
Attributes that are not supplied will not be updated

## [W09-C4] [Do-er List]

### Mark task as done: mark

Marks a certain task as done in the Doer-list.

Format: mark TASK\_NUMBER



Mark task TASK\_NUMBER as done. The task must exist in the Doer-list.

Examples:

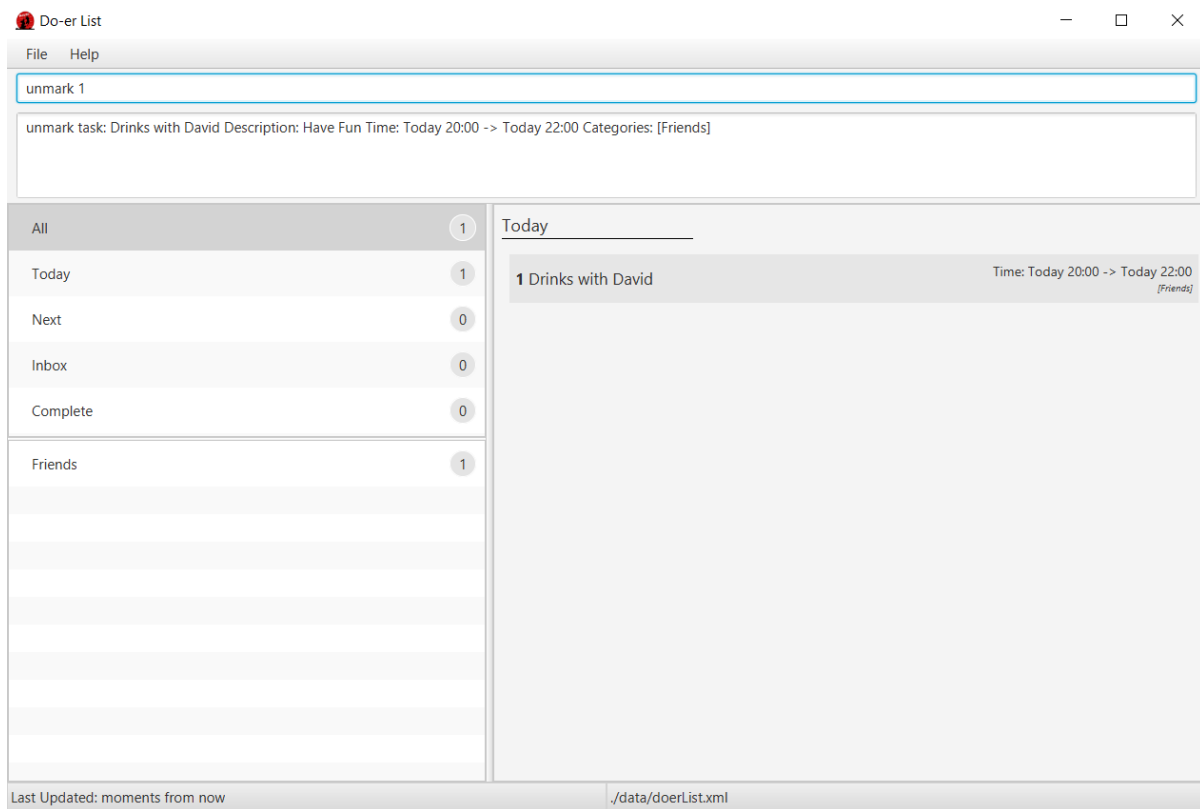
- mark 5
  - Returns task number 5 as done.
  - If a recurring task is mark is done, it will automatically update its start and end date.

## [W09-C4] [Do-er List]

### Unmark task as done: unmark

Marks a certain task as undone in the Doer-list.

Format: unmark TASK\_NUMBER



Mark task TASK\_NUMBER as undone. The task must exist in the Doer-list.

Examples:

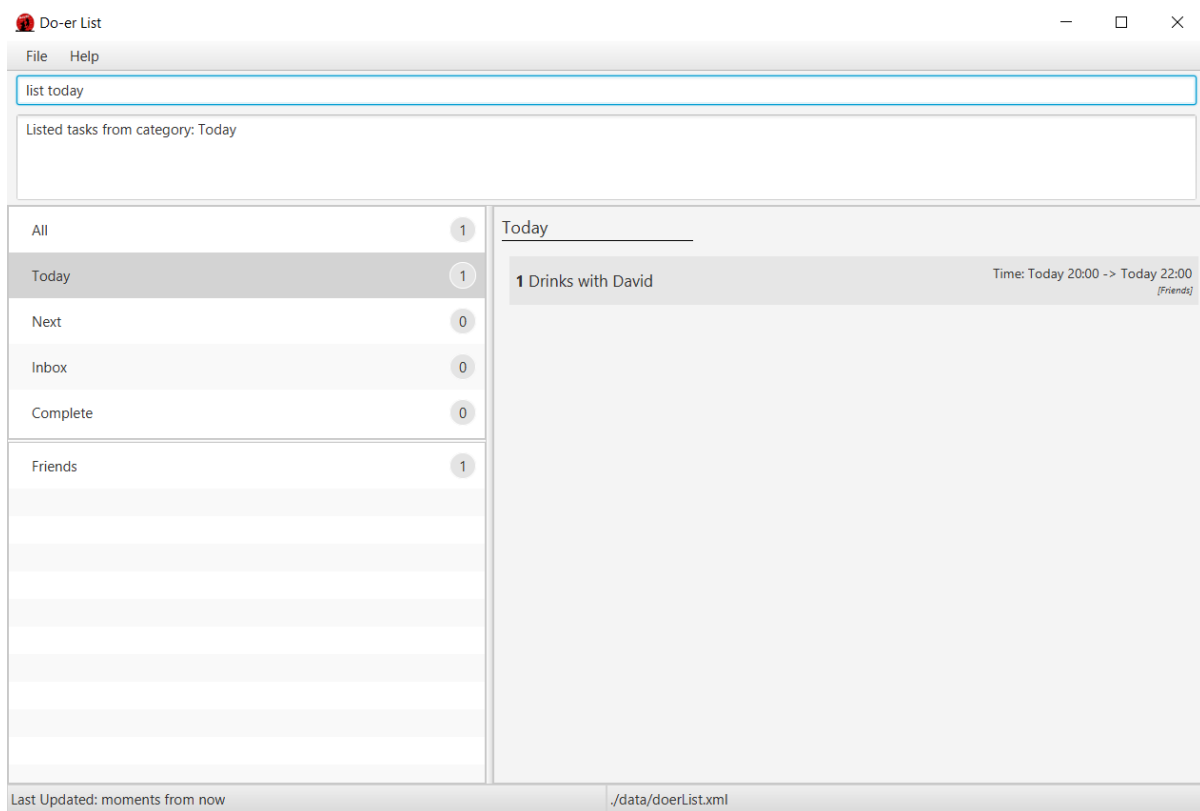
- unmark 5
  - Returns task number 5 as undone.
  - If a recurring task is mark is unmarked, it will automatically subtract its start and end date by its given recurring interval.

## [W09-C4] [Do-er List]

### Listing tasks in a certain category: `list`

Shows a list of all tasks in the Doer-list under the specific category.

Format: `list [CATEGORY]`



- If the CATEGORY or DATE parameter is not supplied, then it will list all tasks.
- Some build-in categories: All Today Next Inbox and Complete can be also listed.

Examples:

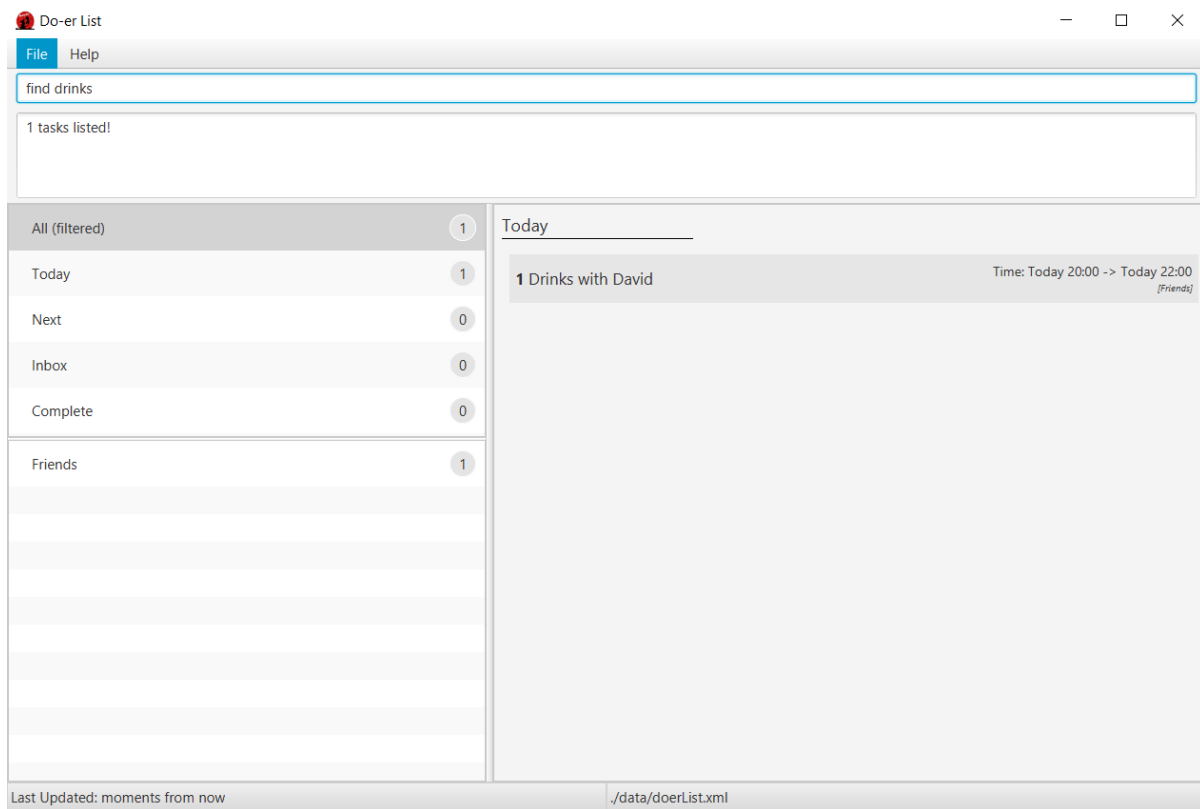
- `list`
- `list Friends`
- `list today`
- `list next`

## [W09-C4] [Do-er List]

### Finding all tasks containing any keyword in their name: `find`

Finds tasks whose names contain any of the given keywords.

Format: `find KEYWORD [MORE_KEYWORDS]`



- The search is not case sensitive.
  - lecture will match LecTure
- The order of the keywords does not matter.
  - go to lecture will match Lecture go to
- Title and Description is searched.

## [W09-C4] [Do-*er* List]

- All data in the Do-*er* List matching at least one keyword will be returned
  - lecture will match have lecture

Examples:

- find david

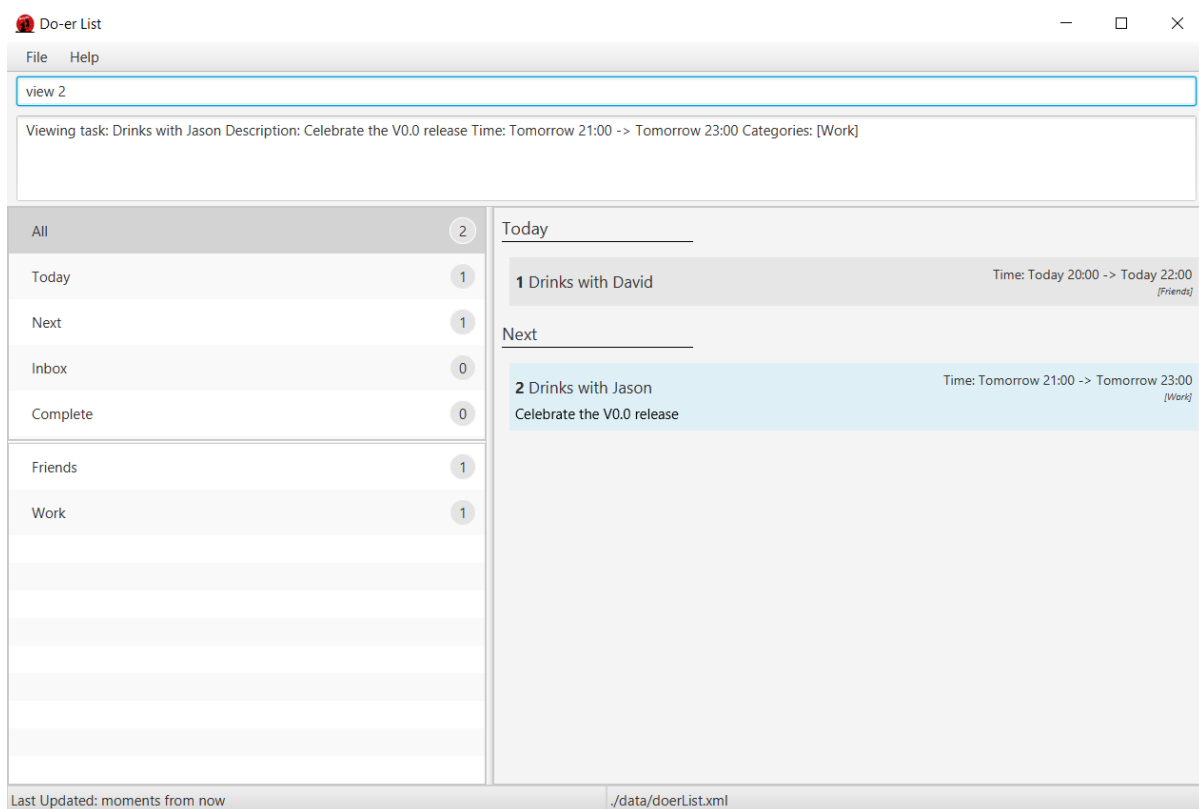
Returns Drinks with David

## [W09-C4] [Do-er List]

### View a task: `view`

Views the task identified by the index number used in the task panel.

Format: `view INDEX`



- Views the details of the task at the specified `INDEX`.
- The index refers to the index number shown in the most recent listing.  
The index **must be a positive integer** 1, 2, 3, ...

Examples:

- `list`  
`view 2`  
Views the 2nd task in the Doer-list.



## [W09-C4] [Do-*er* List]

- `find David`  
`view 1`

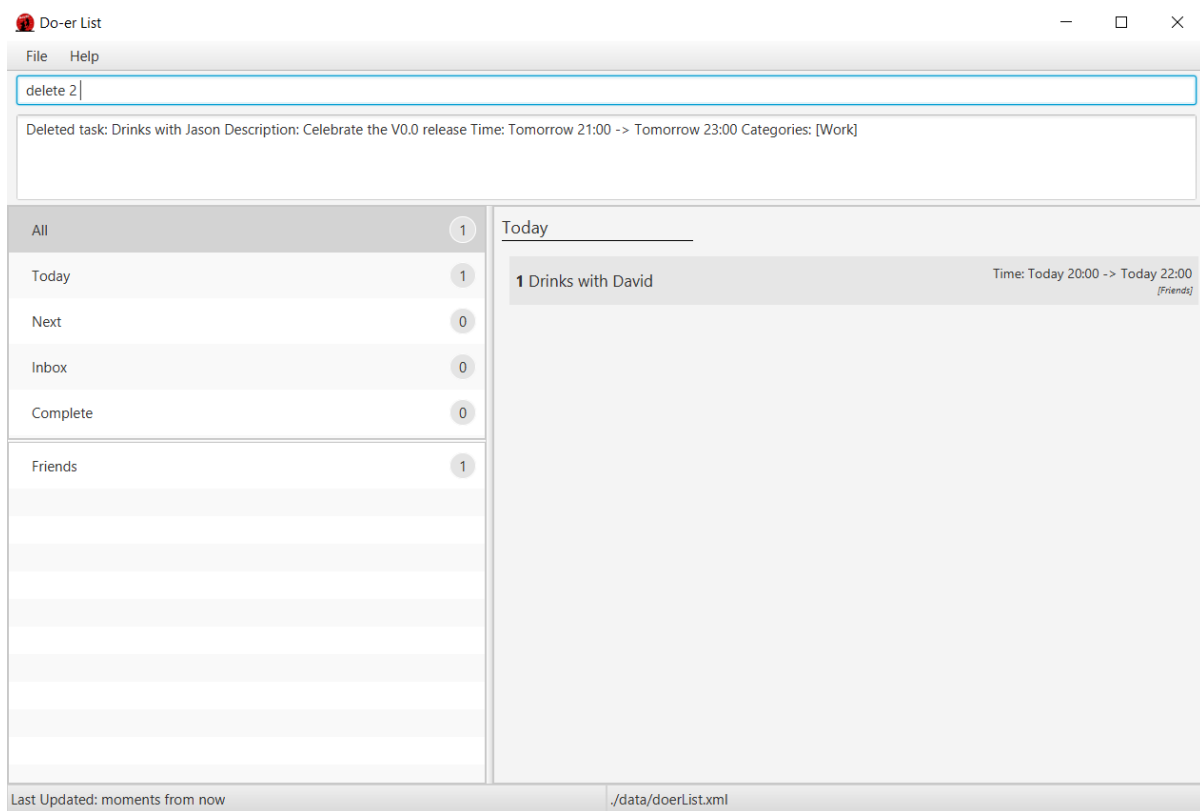
Views the 1st task in the results of the `find` command.

## [W09-C4] [Do-er List]

### Deleting a task: `delete`

Deletes the specified task / event from the Do-erList.

Format: `delete INDEX`



- Deletes the task / event at the specified INDEX.
- The index refers to the index number shown in the most recent listing.  
The index **must be a positive integer** 1, 2, 3, ...

### Examples:

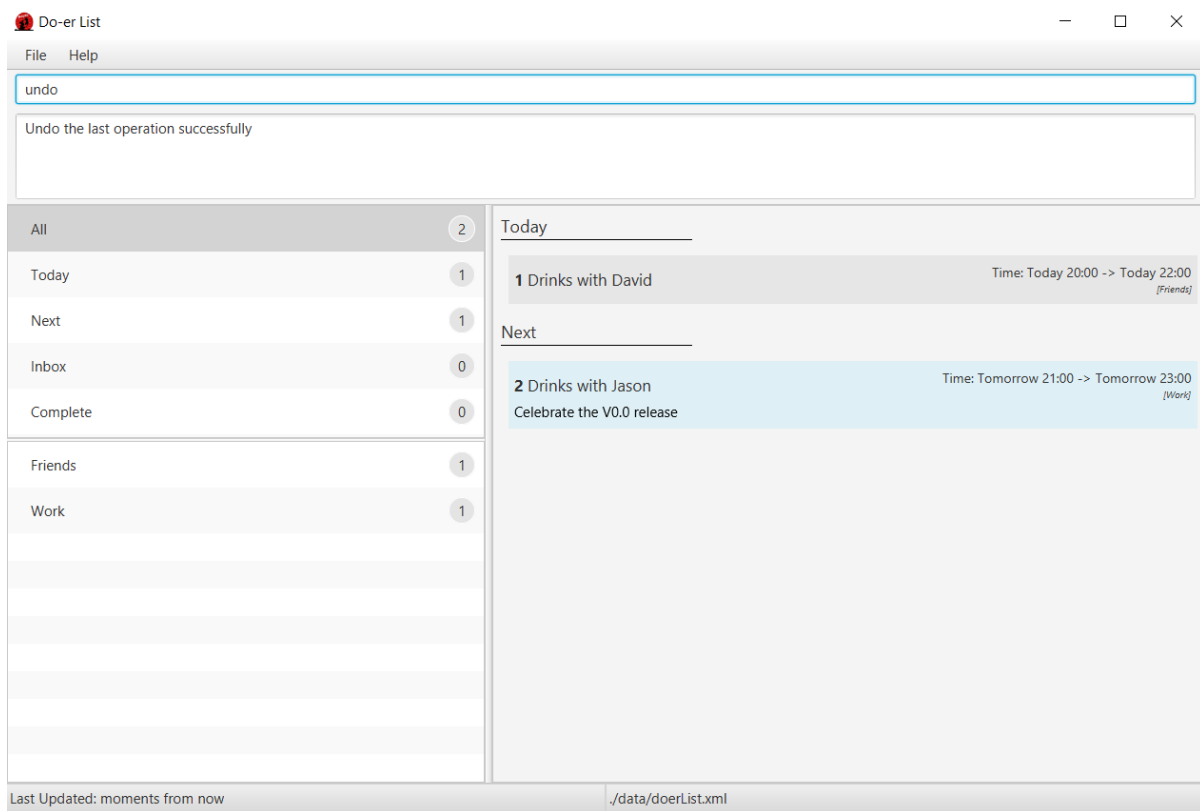
- `list`  
`delete 2`  
Deletes the 2nd task / event in the Do-erList.
- `find David`  
`delete 1`  
Deletes the 1st task / event in the results displayed by using the `find` command.

## [W09-C4] [Do-er List]

### Undo the most recent operation: undo

Undo the most recent operation which modify the data in the Doer-list

Format: undo

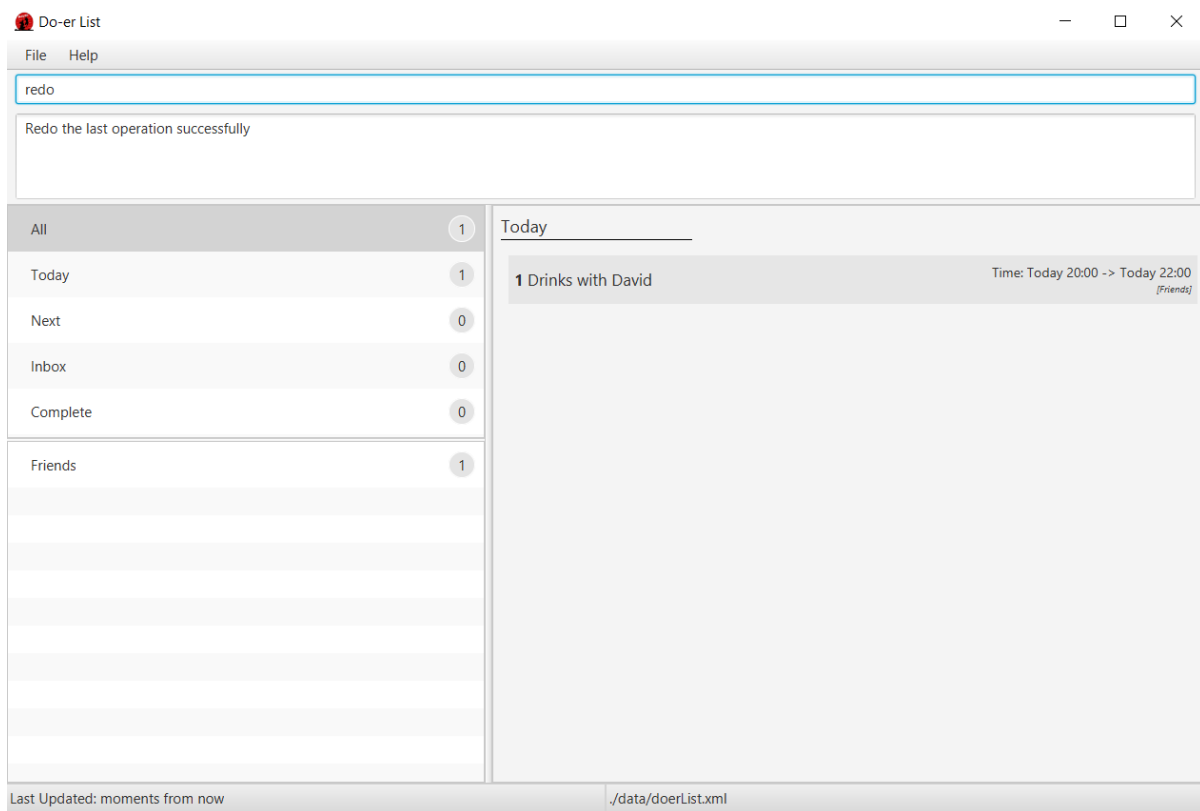


## [W09-C4] [Do-er List]

### Redo the most recent operation: redo

Redo the most recent undo

Format: redo



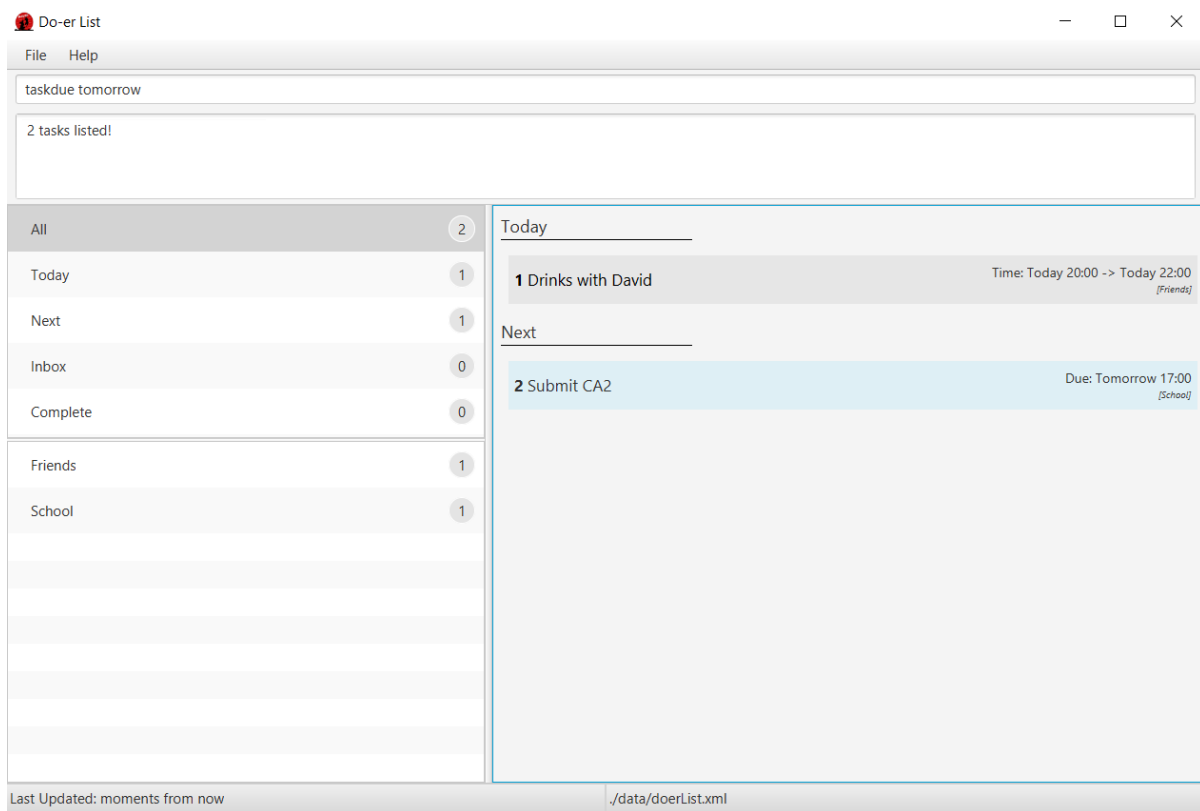
An undo command must first be used before redo command can be executed.

## [W09-C4] [Do-er List]

### Find all tasks due: `taskdue`

Finds all tasks due on and before the date specified in the Doer-list.

Format: `taskdue` `END_DATE`



- Finds all tasks due on and before `END_DATE`.
- The date can be in natural language\*.
- Supported list for natural language\*:
  - next X hours / days / weeks / months
    - X can be any number: 1, 2, 3, ...
  - today
  - tomorrow
  - next week / month

## [W09-C4] [Do-*er* List]

Examples:

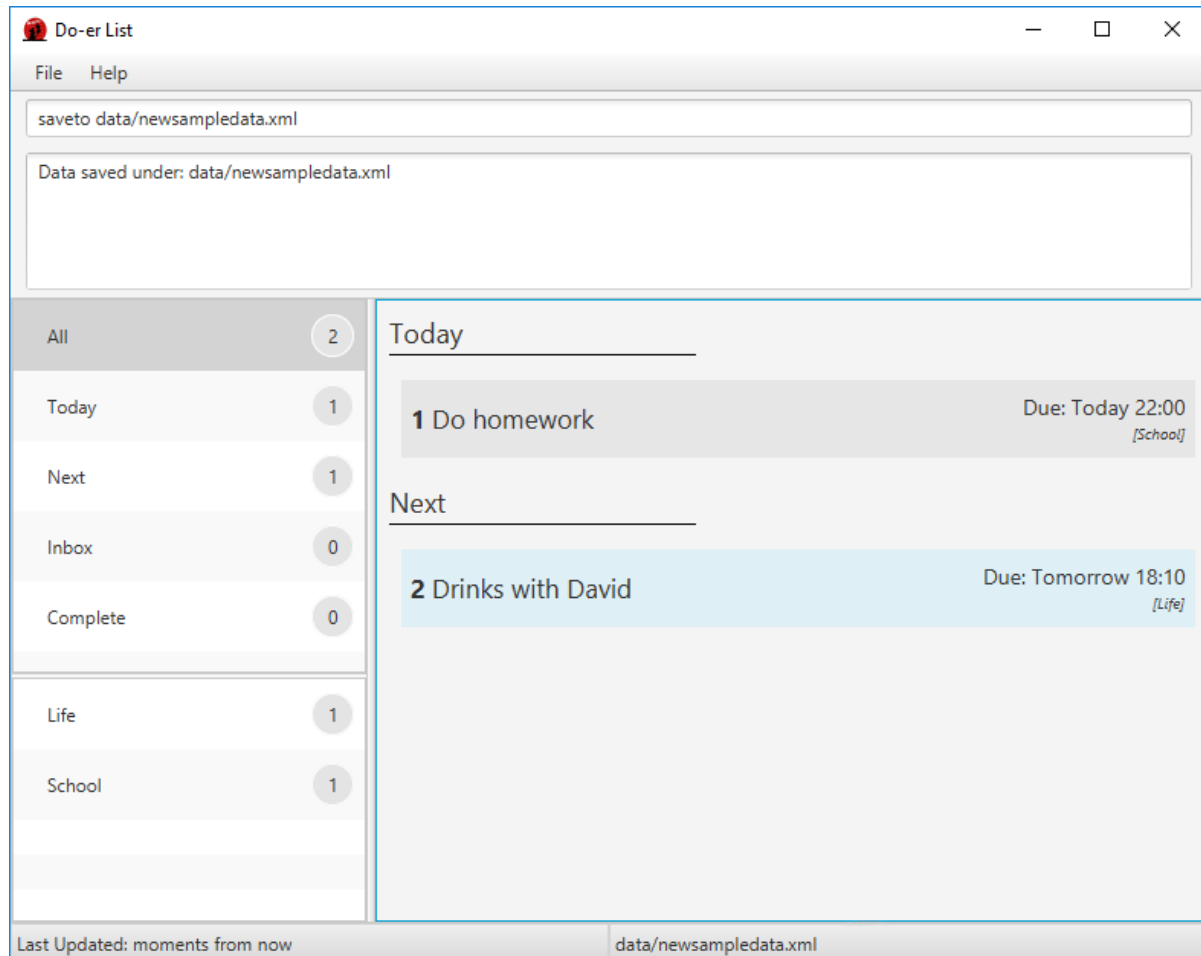
- `taskdue tomorrow`
- `taskdue next 5 hours`
- `taskdue 2016-11-11 21:03`

## [W09-C4] [Do-er List]

### Changing your save location: saveto

Save the data into a new file and location

Format: saveto NEW\_LOCATION



Examples:

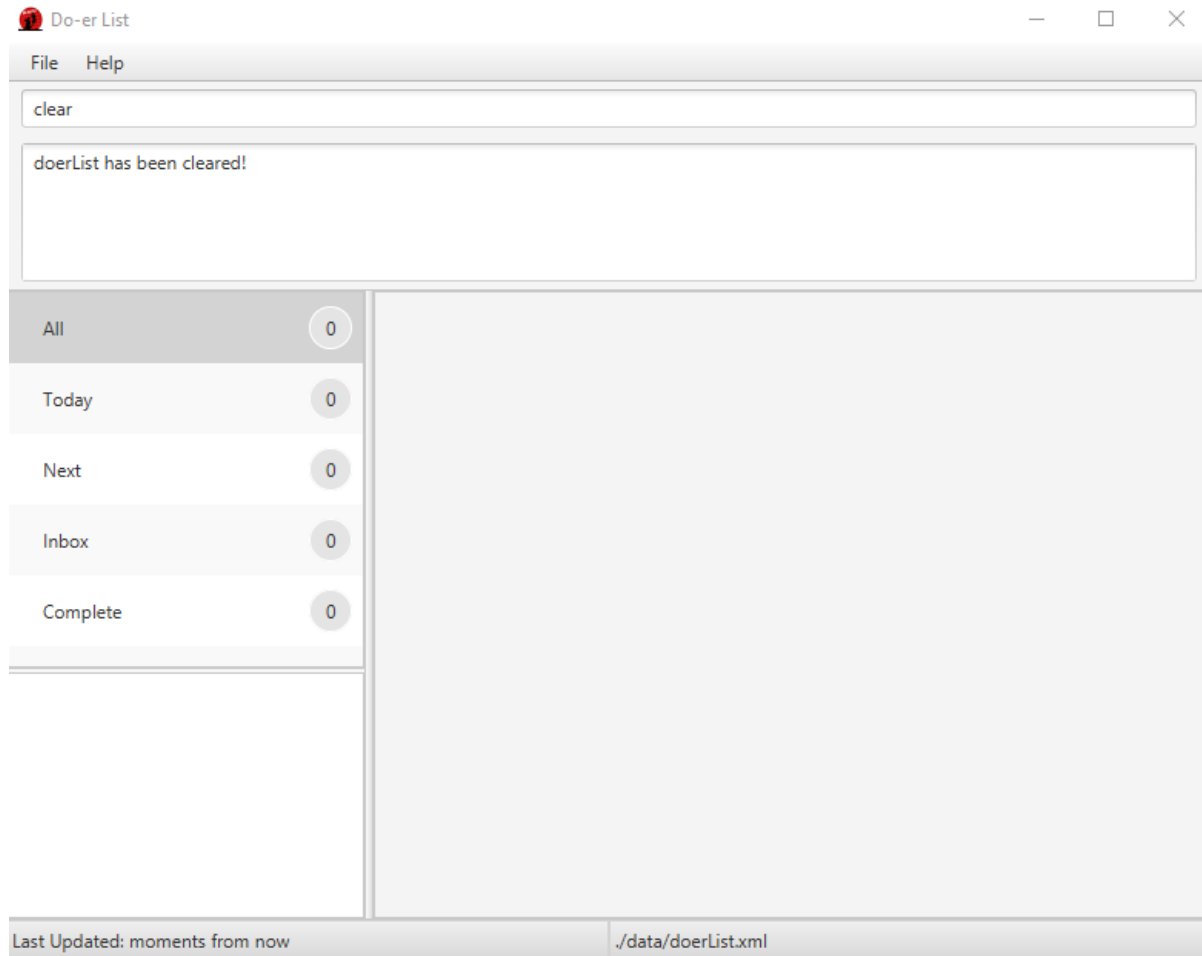
- saveto data/newsampledata.xml

## [W09-C4] [Do-er List]

### Clearing all the tasks: `clear`

Clearing all the tasks

Format: `clear`





## Automatic save

The Doer-list data are saved in the hard disk automatically after any command that changes the data.

## Exiting the program: `exit`

Exits the program.

Format: `exit`

## FAQ

---

**Q:** How do I transfer my data to another computer?

**A:** Install the app in the other computer and overwrite the empty data file it creates with the file that contains the data of your previous Do-er List folder.

**Q:** Running "doerlist.jar" gives an error or does not seem to work.

**A:** You will need to install the latest version of Java. Refer to the installation guide [here](#).

**Q:** Where is the save button for me to save my schedule in this program?

**A:** Your data are saved in the hard disk automatically after any command that changes the data as aforementioned [here](#) in the guide. There is no need for you to save it manually.

## Command Summary

Command	Format
Help	help [COMMAND]
Add	add /t TITLE [/d DESCRIPTION] [/s START] [/e END] [/c CATEGORY] [/r RECURRING] ...
Edit	edit INDEX [/t TITLE] [/d DESCRIPTION] [/s START] [/e END] [/c CATEGORY] [/r RECURRING] ...
Mark Done	mark TASK_NUMBER
Mark Undone	unmark TASK_NUMBER
List	list [CATEGORY]
Find	find KEYWORD [MORE_KEYWORDS]
View	view INDEX
Delete	delete INDEX
Undo	undo
Redo	redo
Task Due	taskdue END_DATE
Save	saveto NEW_LOCATION
Exit	exit

## ***Developer Guide***

---

Developer Guide.....	35
Setting up.....	37
Design.....	38
Architecture .....	38
UI component.....	41
Logic component .....	43
Model component .....	44
Storage component .....	46
Common classes .....	47
Implementation.....	48
Logging.....	48
Configuration.....	48
Testing .....	49
Dev Ops.....	51
Build Automation .....	51
Continuous Integration.....	51
Making a Release .....	51
Managing Dependencies .....	51
Appendix A : User Stories .....	52

## [W09-C4] [Do-er List]

Appendix B: Use Cases .....	55
Appendix C: Non-Functional Requirements.....	67
Appendix D: Glossary .....	67
Appendix E: Product Survey .....	68
Review of TickTick: .....	68
Review of WunderList.....	69
Review of Trello.....	70
Review of Google Calendar .....	71

## Setting up

---

### Prerequisites

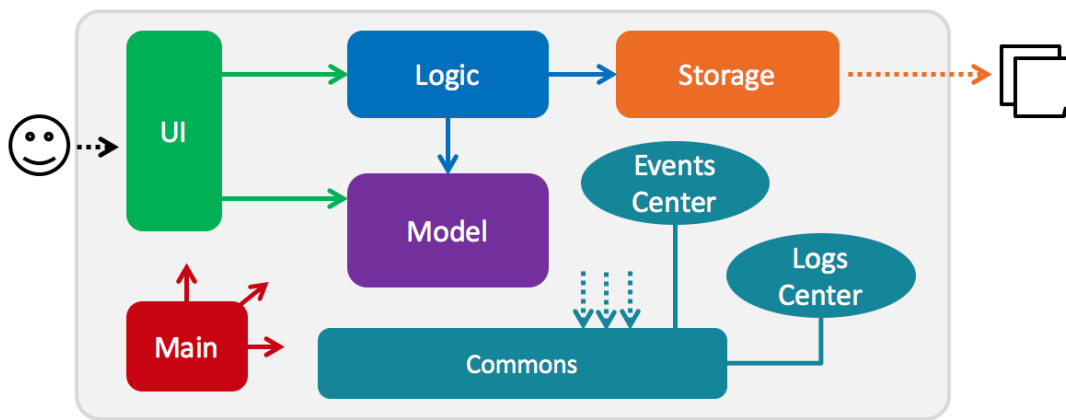
1. **JDK 1.8.0\_60** or later  
Having any Java 8 version is not enough.  
This application will not work with earlier versions of Java 8.
2. **Eclipse** IDE
3. **e(fx)clipse** plugin for Eclipse (Do the steps 2 onwards given in the link below)  
[<http://www.eclipse.org/efxclipse/install.html#for-the-ambitious>]
4. **Buildship Gradle Integration** plugin from the Eclipse Marketplace

### Importing the project into Eclipse

1. Fork this repository, and clone the fork to your computer.
2. Open Eclipse (Note: Ensure you have installed the **e(fx)clipse** and **buildship** plugins as given in the prerequisites above).
3. Click File > Import.
4. Click Gradle > Gradle Project > Next > Next.
5. Click Browse, then locate the project's directory.
6. Click Finish.
  - If you are asked whether to 'keep' or 'overwrite' config files, choose to 'keep'.
  - Depending on your connection speed and server load, it can even take up to 30 minutes for the set up to finish. (This is because Gradle downloads library files from servers during the project set up process.)
  - If Eclipse auto-changed any settings files during the import process, you can discard those changes.

## Design

### Architecture



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

`Main` has only one class called `MainApp`. It is responsible for,

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

`Commons` represents a collection of classes used by multiple other components. Two of those classes play important roles at the architecture level.

- `EventsCenter` : This class (written using [Google's Event Bus library](#)) is used by components to communicate with other components using events (i.e. a form of *Event Driven* design)
- `LogsCenter` : Used by many classes to write log messages to the application's log file.

## [W09-C4] [Do-er List]

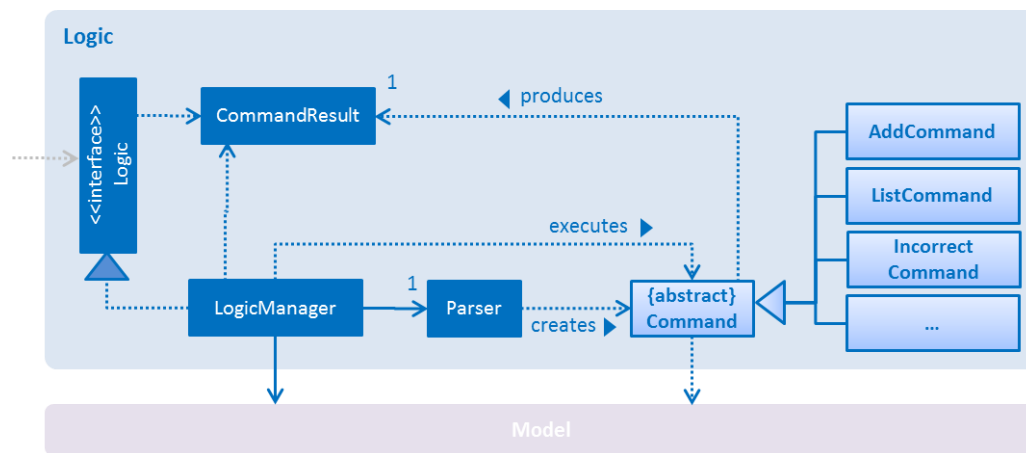
The rest of the application consists four components.

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

Each of the four components

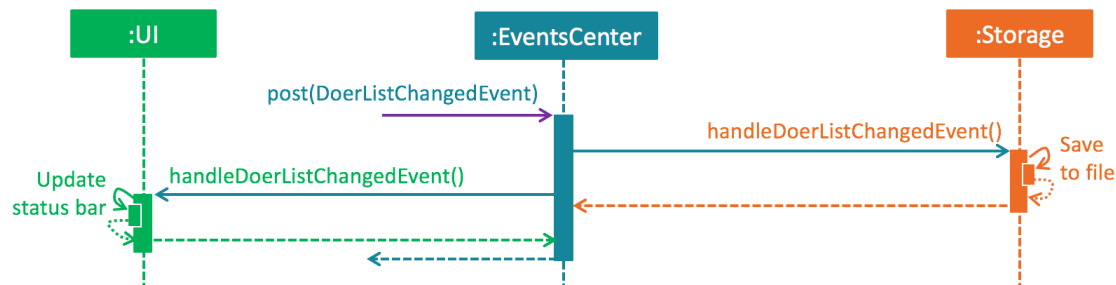
- Defines its *API* in an interface with the same name as the Component.
- Exposes its functionality using a {Component Name}Manager class.

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



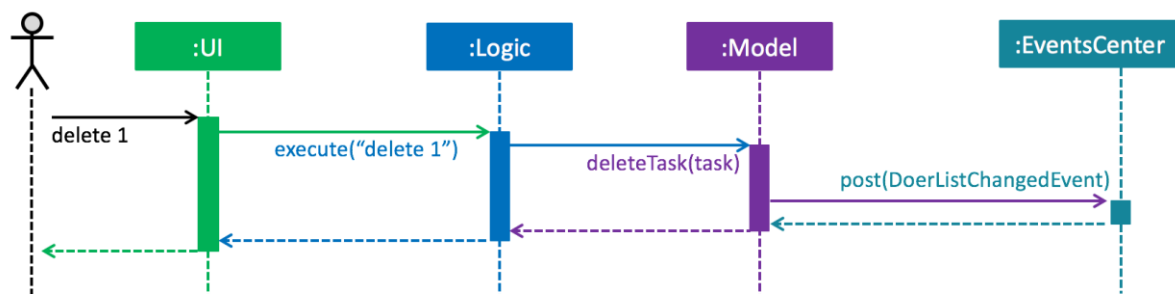
The *Sequence Diagram* below shows how the components interact for the scenario where the user issues the command `delete 3`.

## [W09-C4] [Do-er List]



Note how the `Model` simply raises a `DoerListChangedEvent` when the Do-er List data are changed, instead of asking the `Storage` to save the updates to the hard disk.

The diagram below shows how the `EventsCenter` reacts to that event, which eventually results in the updates being saved to the hard disk and the status bar of the UI being updated to reflect the 'Last Updated' time.

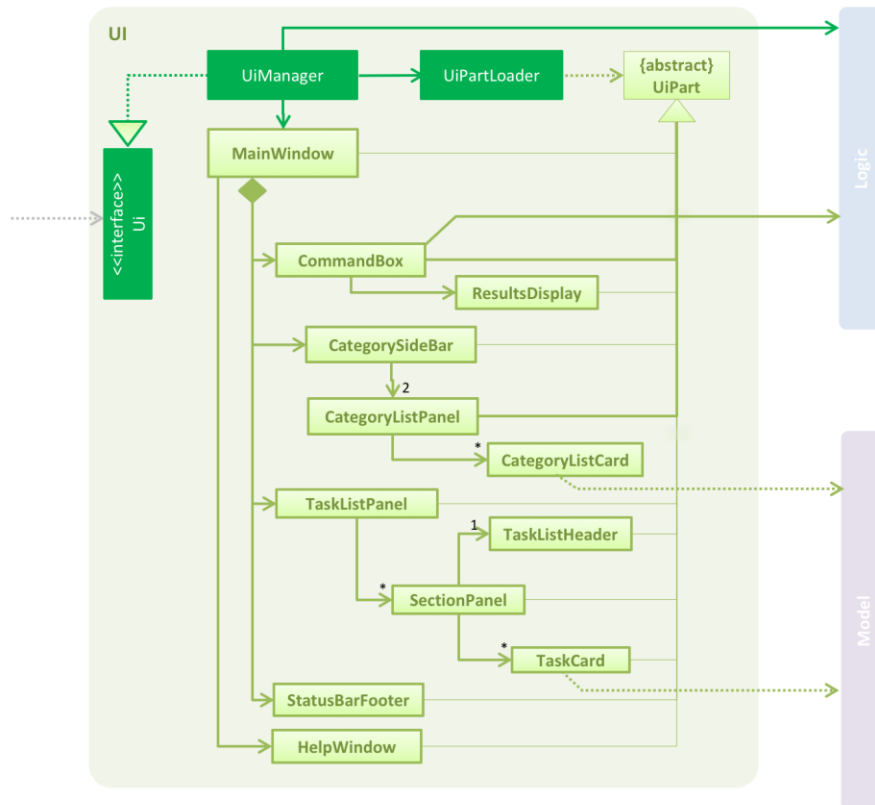


Note how the event is propagated through the `EventsCenter` to the `Storage` and `UI` without `Model` having to be coupled to either of them. This is an example of how this Event Driven approach helps us reduce direct coupling between components.

The sections below give more details of each component.



## UI component



### API : [Ui.java](#)

The UI consists of a **MainWindow** that is made up of parts e.g. **CommandBox**, **ResultDisplay**, **TaskListPanel**, **StatusBarFooter** etc. All these, including the **MainWindow**, inherit from the abstract **UiPart** class and they can be loaded using the **UiPartLoader**.

The UI component uses JavaFx UI framework. The layout of these UI parts are defined in matching **.fxml** files that are in the **src/main/resources/view** folder.

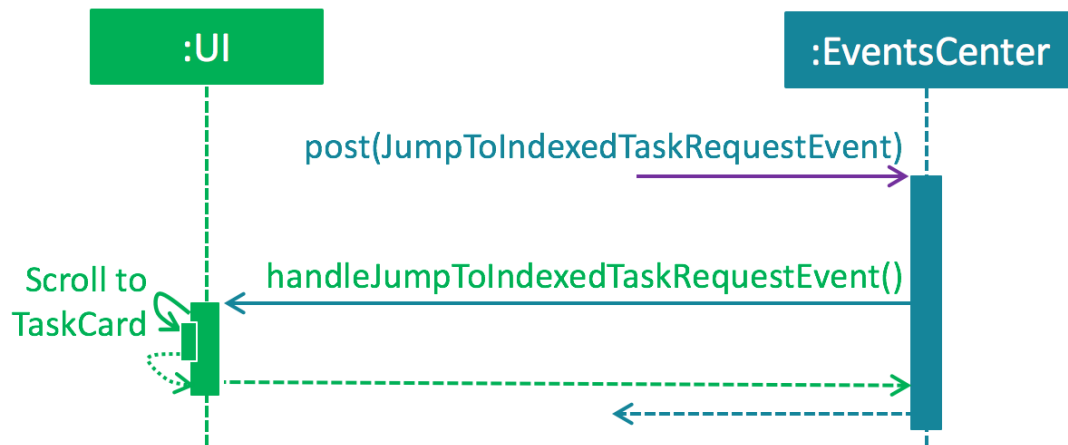
For example, the layout of the **MainWindow** is specified in **MainWindow.fxml**

The UI component,

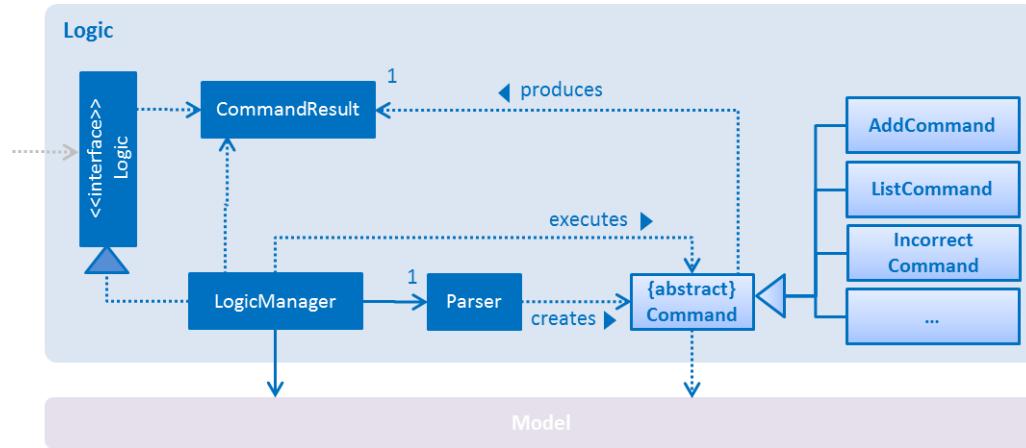
- Executes user commands using the **Logic** component.

## [W09-C4] [Do-er List]

- Binds itself to some data in the `Model` so that the UI can auto-update when data in the `Model` change.
- Responds to events raised from various parts of the application and updates the UI accordingly. For example, the diagram below shows how the UI reacts to the `JumpToIndexedTaskRequestEvent` event.



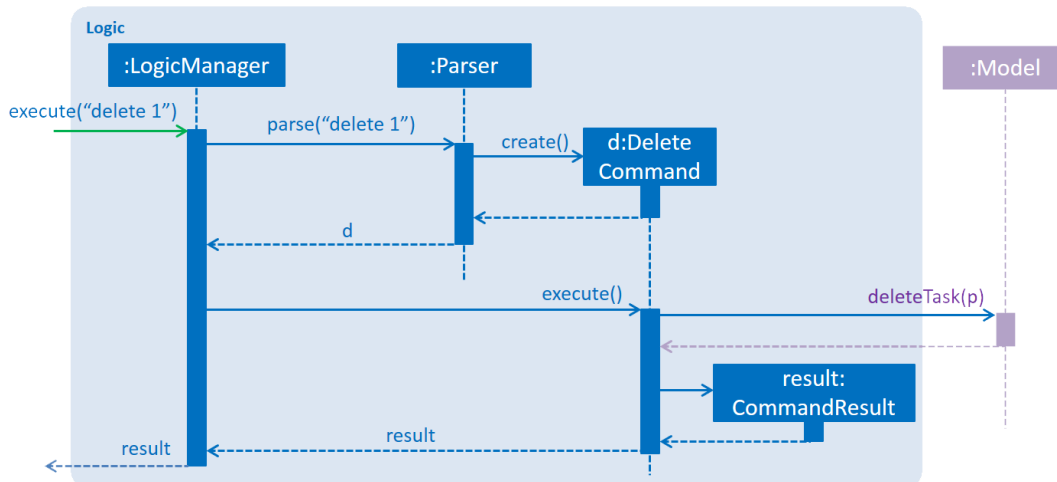
## Logic component



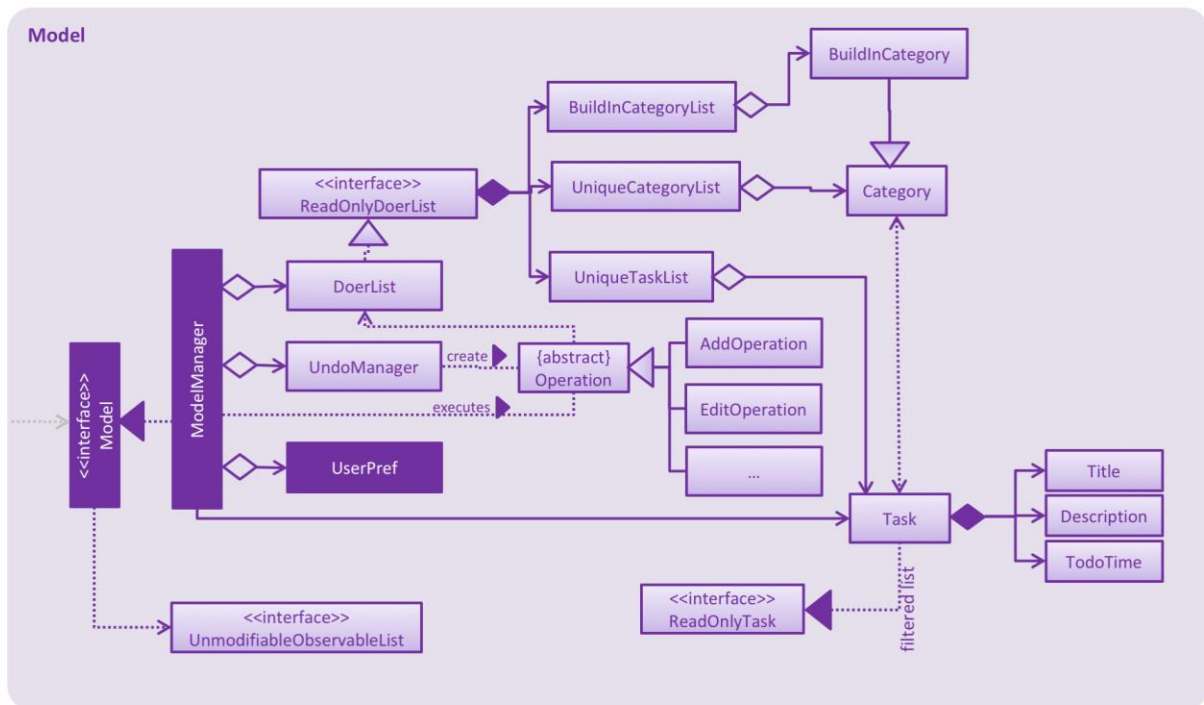
API: `Logic.java`

1. Logic uses the Parser class to parse the user command.
2. This results in a Command object which is executed by the LogicManager.
3. The command execution can affect the Model (e.g. adding a task) and/or raise events.
4. The result of the command execution is encapsulated as a CommandResult object which is passed back to the ui.

Given below is the Sequence Diagram for interactions within the Logic component for the execute ("delete 1") API call.



## Model component



**API :** [Model.java](#)

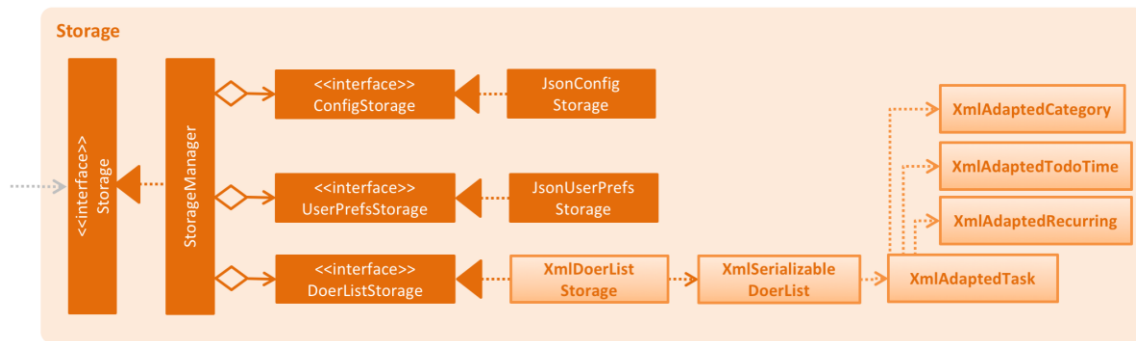
The **Model**,

- stores a **UserPref** object that represents the user's preferences.
- stores the Do-er List data.
- stores a **UndoManager** that records the history operations to Do-er List data.  
For example, when **Model** requests to add a task to Doer List, **UndoManager** will records the opposites operations (delete a task in this case) in its undo stack. Next time, when the **undo** method of the **Model** is called, **Model** will pull the first operation in the **UndoManager** and executes it. Meanwhile, **UndoManager** will push the operation to its redo stack. Therefore, next time when **redo** method is called, the **Model** could pull the first operation in **UndoManager**'s redo stack and executes it.

### **[W09-C4] [Do-*er* List]**

- exposes `UnmodifiableObservableList<ReadOnlyTask>`, `UnmodifiableObservableList<Category>` that can be 'observed' (E.g. the UI can be bound to this list) so that the UI automatically updates when the data in the list changes.
- does not depend on any of the other three components.

## Storage component



### API : [Storage.java](#)

The storage component,

- can save Config objects in json format and read it back
- can save UserPref objects in json format and read it back.
- can save the Do-er List data in xml format and read it back.

## Common classes

Classes used by multiple components are in the `seedu.doerList.commons` package. Some of them are below:

- [EventsCenter.java](#) : The class is responsible for post events and handling events. When an event has been raised, the `EventsCenter` will check whether there is handler for the event and notify the correct handlers.
- [LogsCenter.java](#): The class is responsible for recording the operation histories in execution. The records are useful for programmer in debugging.
- [BaseEvent.java](#): The is the super class for every single event raised by the application.

## ***Implementation***

---

### **Logging**

We are using `java.util.logging` package for logging. The `LogsCenter` class is used to manage the logging levels and logging destinations.

- The logging level can be controlled using the `logLevel` setting in the configuration file. (See [Configuration](#))
- The `Logger` for a class can be obtained using `LogsCenter.getLogger(Class)` which will log messages according to the specified logging level.
- Currently log messages are outputted through `console` and to a `.log` file.

### **Logging Levels**

- **SEVERE:** Critical problem detected which may possibly cause the termination of the application.
- **WARNING:** Can continue, but proceed with caution.
- **INFO:** Information showing the noteworthy actions by the application.
- **FINE:** Details that is not usually noteworthy but may be useful in debugging. e.g. print the actual list instead of just its size.

### **Configuration**

Certain properties of the application can be controlled (e.g App name, logging level) through the configuration file. (default: `config.json`):



## Testing

---

Tests can be found in the `./src/test/java` folder.

### In Eclipse:

If you are not using a recent Eclipse version (i.e. *Neon* or later), enable assertions in JUnit tests as described [here](#).

- To run all tests, right-click on the `src/test/java` folder and choose `Run as > JUnit Test`
- To run a subset of tests, you can right-click on a test package, test class, or a test and choose to run as a JUnit test.

### Using Gradle:

- See [UsingGradle.md](#) for how to run tests using Gradle.

We have two types of tests:

1. **GUI Tests** - These are *System Tests* that test the entire application by simulating user actions on the GUI. These are in the `guitests` package.
2. **Non-GUI Tests** - These are tests that do not involve the GUI. They include:
  - i. *Unit tests* targeting the lowest level methods/classes.  
e.g. `seedu.doerList.commons.UrlUtilTest`
  - ii. *Integration tests* that are checking the integration of multiple code units (those code units are assumed to be working).  
e.g. `seedu.doerList.storage.StorageManagerTest`
  - iii. Hybrids of unit and integration tests. These tests are checking multiple code units as well as how they are connected together.  
e.g. `seedu.doerList.logic.LogicManagerTest`

**Headless GUI Testing:** Thanks to the [TestFX](#) library that we are using, our GUI tests can be run in *headless* mode. In the headless mode, GUI tests do not show up on the screen.

## **[W09-C4] [Do-*er* List]**

That means the developer can do other things on his computer while the tests are running.

See [UsingGradle.md](#) to learn how to run tests in headless mode.

## Dev Ops

---

### Build Automation

See [UsingGradle.md](#) to learn how to use Gradle for build automation.

### Continuous Integration

We use [Travis CI](#) to perform *Continuous Integration* on our projects.

See [UsingTravis.md](#) for more details.

### Making a Release

Here are the steps to create a new release.

1. Generate a JAR file [using Gradle](#).
2. Tag the repository with the version number. e.g. v0.1
3. [Create a new release using GitHub](#) and upload the JAR file your created.

### Managing Dependencies

A project often depends on third-party libraries. For example, Do-er List depends on the [Jackson library](#) for XML parsing. Managing these *dependencies* can be automated using Gradle. For example, Gradle can download the dependencies automatically, which is better than these alternatives.

- a. Include those libraries in the repository (this bloats the repo size)
- b. Require developers to download the necessary libraries manually (this creates extra work for developers)

## Appendix A : User Stories

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

Priority	As a ...	I want to ...	So that I can...
* * *	new user	see the usage instructions of all commands	how to use various commands in the application
* * *	user	create task with title and description	put summary as title and more details in the description
* * *	user	create task without start or end time	record tasks that need to be done without <a href="#">deadlines</a>
* * *	user	create task with start and end time or <a href="#">deadlines</a>	know what task I should be doing or due on a certain timing
* * *	user	edit task's title, description, start time, end time and categories	update it when I want to in the event I need to make some changes instead of creating a new task
* * *	user	view all tasks	have an overview of all the tasks

[W09-C4] [Do-er List]

Priority	As a ...	I want to ...	So that I can...
* * *	user	view a specific task	get more details of the specific task
* * *	user	find a task by title and description	quickly locate the task if I can only remember few words in the title or description
* * *	user	delete tasks	track only the required tasks
* *	user	add tasks to different categories	organise my tasks properly
* *	user	view the tasks under a certain category	examine different tasks under different categories
* *	user	view the tasks are going to happen or due today, tomorrow, next 7 days, etc.	know what is going to happen in the coming day(s)
* *	user	undo the most recent operations	revert the changes when I make a wrong command by mistake
* *	user	redo the most recent	redo the operation after undoing it

## [W09-C4] [Do-er List]

Priority	As a ...	I want to ...	So that I can...
		operations	
* *	user	specify a storage location for data storage	specify my data location to the cloud syncing service provider to prevent data loss
* *	user	mark or unmark the task as <a href="#">done</a>	only need to keep track of the tasks which are needed to be done and archive the finished tasks
*	user	type command parameters in arbitrary order	find the specific order of parameter(s) in the event I forget
*	user	add external ical file to the todo-lists	keep track of tasks created by other applications.
*	user	create recurring tasks	be reminded to do the same task at the same time interval
*	user	view events in Google Calendar	have a better pictorial view of my schedule

## Appendix B: Use Cases

---

(For all use cases below, the **System** is the Do-er List and the **Actor** is the user, unless specified otherwise)

### Use case: Add task

#### MSS

1. User requests to add a task in.
2. System creates task with title, description, start date, end date, recurring interval.
3. The task is moved into the categories according to the supplied parameters.
4. System displays the details of the created task.  
Use case ends.

#### Extensions

- 1a. add command followed by the wrong parameters.
  - 1a1. System indicates the error and displays the correct format for user.  
Use case ends.
- 2a. TITLE is empty string.
  - 2a1. System indicates the error that TASK\_NAME is empty.  
Use case ends.
- 2b. User does not supply START or END parameters.
  - 2b1. Task is created and categorized to INBOX.
  - 2b2. System display the created task.  
Use case resumes from step 2.
- 2c. User does not supply START parameter.
  - 1c1. Task is created with START as today.  
Use case resumes from step 2.
- 2d. System can parse START or END, which are not in standard format.  
Use case resumes from step 2.
- 2e. System is unable to parse START or END, which are not in standard format.

## [W09-C4] [Do-er List]

- 2e1. System indicates the error to user.  
Use case resumes from step 2.
- 2f. START time is after END time
  - 2f1. System indicates the error to user  
Use case ends
- 2g. There is not START time or END time while recurring interval is supplied
  - 2g1. System indicates the error to user  
Use case ends
- 3a. The category does not exist in the system
  - 3a1. System creates a new category  
Use case resumes from step 3
- 3b. The category does not exist in the system
  - 3b1. System creates a new category
  - 3b2. The category name is invalid
  - 3b3. System indicates the error  
Use case ends



## Use case: Edit task

### MSS

1. User types in the command.
2. System finds the task at that index.
3. A new task is created based on new attributes and old attributes. (E.g. title, description, start time, end time, recurring interval, category).
4. System replaces the old task with new task.
5. System displays the details of the newly edited task.  
Use case ends.

### Extensions

- 1a. edit command followed by the wrong parameters.  
1a1. System indicates the error and display the correct format for user.  
Use case ends.
- 2a. edit command is followed by the non-existent INDEX.  
2a1. System indicates the error that the INDEX is non-existent  
Use case ends.
- 3a. System can parse START or END, which are not in standard format.  
Use case resumes from step 2.
- 3b. System is unable to parse START or END, which are not in standard format.  
3b1. System indicates the error to user.  
Use case resumes from step 2.
- 3c. START time is after END time  
3c1. System indicates the error to user  
Use case ends
- 3d. There is not START time or END time while recurring interval is supplied  
3d1. System indicates the error to user  
Use case ends
- 3e. The category does not exist in the system  
3e1. System creates a new category  
Use case resumes from step 3

**[W09-C4] [Do-*er* List]**

- 3f. The category does not exist in the system
  - 3f1. System creates a new category
  - 3f2. The category name is invalid
  - 3f3. System indicates the error
  - Use case ends
- 4a. The replacement of task results in duplication in task list
  - 4a1. System indicates the error
  - Use case ends

### Use case: Delete task

#### MSS

1. User types in the command.
  2. System finds the task at that index.
  3. System deletes the task.
- Use case ends.

#### Extensions

- 1a. delete command is followed by the wrong parameters
  - 1a1. System indicates error and display the correct format to user.Use case ends.
- 1b. delete command is followed by a non-existent INDEX
  - 1b1. System indicates the error in the INDEX is non-existent.Use case ends.

## Use case: List task by category

### MSS

1. User types the command with CATEGORY as parameter.
2. System displays all the task under the CATEGORY.

Use case ends.

### Extensions

- 1a. User does not supply CATEGORY.
  - 1a1. System displays all the tasks.Use case ends.
- 2a. The category does not exist in the system.
  - 2a1. System indicates the error.Use case ends.

## **Use case: Undo Command**

### **MSS**

1. User types in the undo command.
2. System tries to find the last operation that caused a change of data.
3. System undoes the operation.
4. System indicates the change to user.  
Use case ends.

### **Extensions**

- 2a. The last operation which involve the change of the data does not exist  
2a1. System indicates the error.  
Use case ends.

## **Use case: Clear Command**

### **MSS**

1. User types in the command.
2. System deletes all the tasks.  
Use case ends.

## Use case: Help Command

### MSS

1. User types in the command.
2. System finds the details of a command with its parameters.
3. System displays the details.  
Use case ends.

### Extensions

- 1a. help command is followed by the wrong parameters.
  - 1a1. System indicates the error and displays the correct format for the user. Use case ends.
- 1b. help command is followed by no parameter.
  - 1b1. System open the help windows  
Use case ends.

## Use case: View a task

### MSS

1. User types in the view command.
2. System retrieves the task list based on the index parameter in the recent displayed list.
3. System displays the detail of the task.  
Use case ends.

### Extensions

- 1a. view command is followed by the wrong parameters
  - 1a1. System indicates the error  
Use case ends
- 2b. The index is not valid.
  - 2b1. System indicates the errors to user.  
Use cases ends.

## Use case: Find keywords

### MSS

1. User requests to find keywords.
2. System shows the tasks with requested keywords.  
Use case ends.

### Extensions

- 1a. User doesn't supply keywords
  - 1a1. System indicates the errors to use  
Use case ends.

## Use case: Task Due Command

### MSS

1. User requests to find all tasks due by end date.
2. System parse the DATE to standard form
3. System List shows all of the tasks due by end date.  
Use case ends.

### Extensions

- 1a. taskdue command is followed by the wrong parameters
  - 1a1. System indicates the error  
Use case ends.
- 2a. System is able to parse the supplied DUE date to standard format
  - 2a1. Use case resumes from step 2
- 2b. System is not able to parse the supplied DUE date to standard format
  - 2b1. System indicates the error Use case ends.

## Use case: Redo Command

### MSS

1. User types the command.
2. System reverses the changes caused by the most recent undo.  
Use case ends.

### Extensions

- 1a. No recent undo is recorded.
  - 1a1. System indicates the error and shows the error message. Use case ends.



## Use case: Mark Command

### MSS

1. User request to mark task of INDEX done
2. System find the task with INDEX number
3. System marked the task as done.  
Use case ends.

### Extensions

- 1a. mark command is followed by the wrong parameters  
1a1. System indicates the error  
Use case ends
- 2a. System cannot find task with INDEX number.  
2a1. System shows an error message.  
Use case ends.

## Use case: Unmark Command

### MSS

1. User request to mark task of INDEX undone.
2. System find the task with INDEX number
3. System marked the task as undone.  
Use case ends.

### Extensions

- 1a. unmark command is followed by the wrong parameters  
1a1. System indicates the error  
Use case ends
- 2a. The INDEX is invalid.  
2a1. System shows an error message.  
Use case ends.

## Use case: Save Command

### MSS

1. User request to change the location of saving file.
2. System changes the saving location
3. System indicates the changing is successful.  
Use case ends.

### Extensions

- 1a. saveto command is followed by the wrong parameters  
1a1. System indicates the error  
Use case ends
- 2a. The saving path is invalid.  
2a1. System shows an error message.  
Use case ends.

## Use case: Exit Command

### MSS

1. User request to exit the programme
2. System terminated.

## ***Appendix C: Non-Functional Requirements***

---

1. The program should work on any mainstream OS as long as it has Java 1.8.0\_60 or higher installed.
2. It should be able to hold up to 1000 tasks.
3. Automated unit tests and open source code for this program should be readily available.
4. Every operation executed should be logged to the log file.
5. The program should favour DOS style commands over Unix-style commands.
6. The product should not have dependencies on other packages.
7. The software can be launched, without installing, by clicking on the executable file.

## ***Appendix D: Glossary***

---

### **Mainstream OS**

Windows, Linux, Unix and OS-X

### **Deadline**

A time interval with the start day as the day the task created day and the end day represents the date of the deadline.

### **Done**

The built-in category in the system which store all the tasks that are marked as done.

## ***Appendix E: Product Survey***

---

### **Review of [TickTick](#):**

#### **Strengths:**

- Desktop software is provided, so we can launch it quickly without using a browser.
- Task are automatically categorised to Overdue Today Next 7 days and Complete, users can quickly find what they need to do in a certain day.
- User can create their own categories for tasks and allocate tasks to different categories.
- Elegant GUI is provided; the UI is not wordy and icons are quite intuitively.

#### **Weaknesses:**

- Network connection is required. If there is no network connection, the software even cannot be opened.
- The parser for input text can only deal with simple command.
  - E.g. Adding the start time of event. If the command cannot be recognized, it will be automatically added as task title.

## Review of [WunderList](#)

### Strengths:

- Ease of usage is the biggest strength. A user can easily add multiple items just by entering his desired items.
- Apple Watch integration is a nice bonus for Apple Watch owners.
- Slick user interface that allows background customizations.

### Weaknesses:

- The free version has very limited features. Users only get 25 assigns per shared to-do list and 10 background choices.
- A constant network connection is required. If there is no network connection, the software cannot be opened.
- Wunderlist lacks IFTTT integration compared to other to-do list applications.

## Review of [Trello](#)

### Strengths:

- Online/Cloud based program that allows it to be transferrable to other computers.
- Ease to add in notes and description into Trello cards.
- Customizable looks.

### Weaknesses:

- It cannot link up with other calendar software like Google calendar, which makes it hard to keep track of tasks done.
- Limited file attachment size, a problem when it comes to uploading large media files like videos or high quality images.
- The free version is much more limited than the paid version, making certain customisation features difficult to accomplish.

## Review of [Google Calendar](#)

### Strengths:

- Adds different kind of colouring to the schedule.
- Undo addition or deletion of events.
- Create multiple calendar for different purposes.
- GUI is quite intuitive. The formatting is clear and it does not require guidelines.
- Able to use calendar in offline mode.

### Weaknesses:

- Unable to view all deleted events or reminders.
- It does not have command-line inputs to modify the calendar; most operations require a user to click, which can be time-consuming.
- Only accessible via browsers; no desktop application available.