# Le Duc - Developer Guide

1. Setting Up

- Please ensure that you have Java 11 or above installed.

- Download the latest leduc jar file.

- Execute the jar file with command: `[CS2113-T16-1][Le Duc].jar`

2. Design

2.1. Class Diagram

Le Duc main class, called `Duke`, is composed of 4 classes : `Storage`, `Ui`, `Parser`, `TaskList`.

- `Storage` deals with saving and loading files such as the file containing the config or the file containing all the tasks.

- `Ui` deals with the interaction between the user and the program.

- `Parser` given an user's input (through Ui), the Parser will return the corresponding command

- `TaskList` represents the list containing all the tasks.
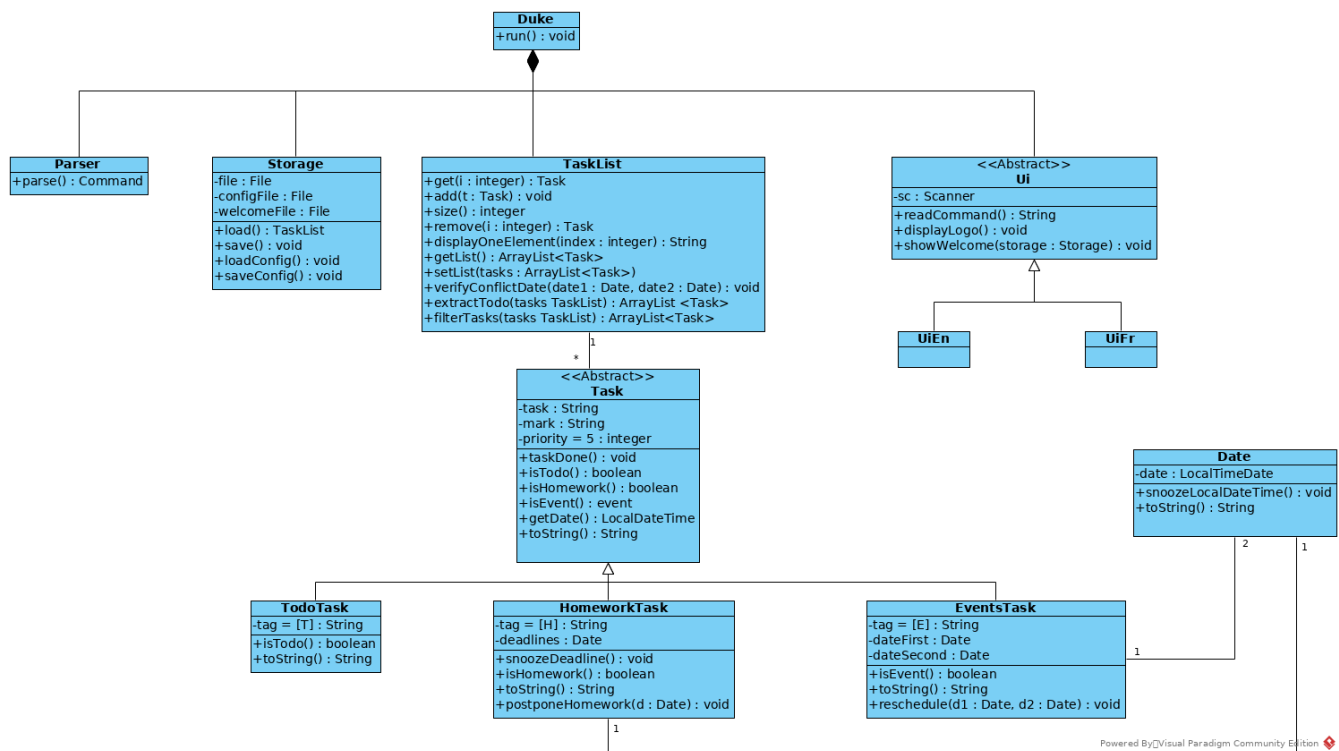
In this class diagram, constructors, getters and setters were not all represented.

The following class diagram will represent the interaction between all classes without being too specific. The class diagram for the abstract class `Command` and his concrete class is not represented because it is only one abstract class connected to a multitude of concrete class. The same goes for the `DukeException` class.
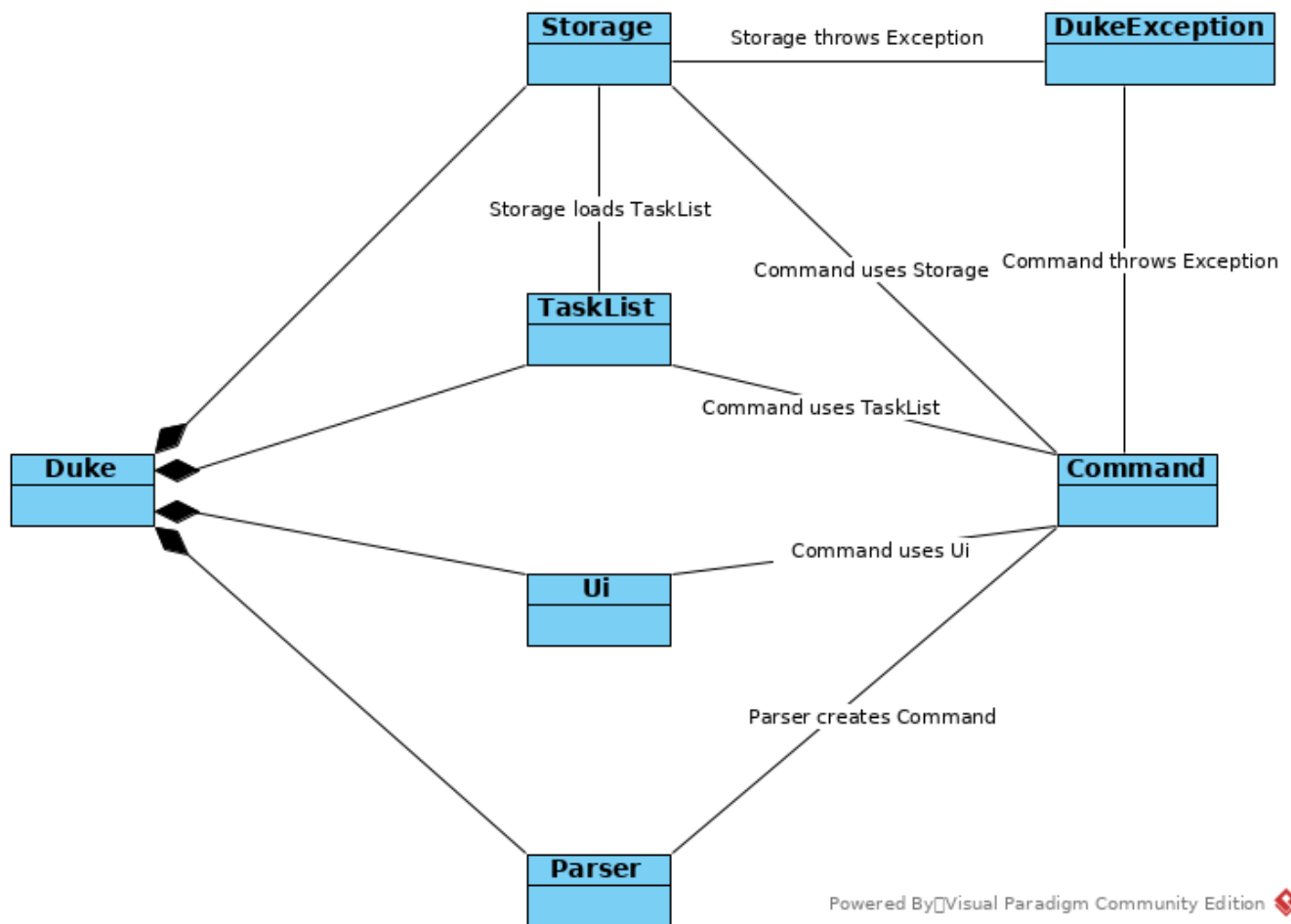
The following class diagram represents in details the abstract class `Command` with all its inherited concrete class.
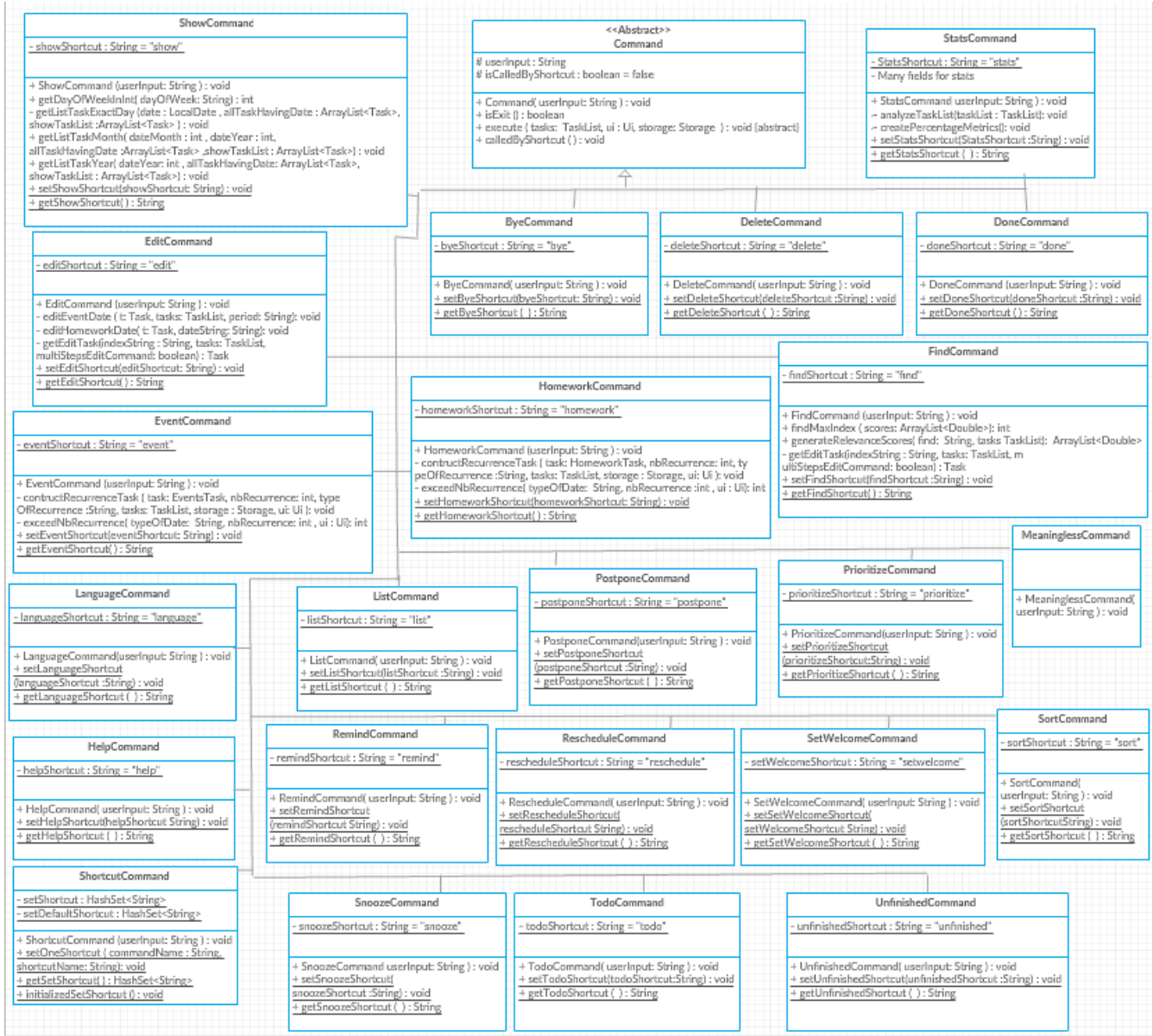
3. Implementation

3.1. Customization

The user can customize Le Duc in the following ways :

- shortcut: The user can implement and use shortcut for every command.
- language: The user can change the language for Le Duc.

3.1.1. Shortcut

The shortcut mechanism is done by the ShortcutCommand. As every other command, it extends Command with a HashSet containing all the command's shortcut name and another HashSet containing all the default command's shortcut name. Others commands include now a static attribute named shortcut that correspond to the command's shortcut. It implements these following methods:

- `ShortcutCommand#setOneShortcut` — set the shortcut of one command
- `ShortcutCommand#initializedSetShortcut` — initialized the HashSet contains all the default command's shortcut name

There are three cases:

- one shot one command: The user write the command for the shortcut in one line
- multi step one command: The user write which command he wants to add a shortcut to, then the console ask what is the shortcut, and the user write the name of the shortcut
- multi step every command: The user asks the console that he wants to modify all the command, and the console will show one by one every command, and the user will modify one by one each command.

When the user launches the application, the program will read the config file, then set all shortcuts to previous shortcuts that the user has decided. If the user has not decided to customized shortcuts, it will be the default shortcut.

These following diagram show how the 3 cases were implemented:

One shot one command

The user type the "entry" (not shown in the sequence diagram) as `shortcut CommandName ShortcutName`.

Modify a command's shortcut name: one shot
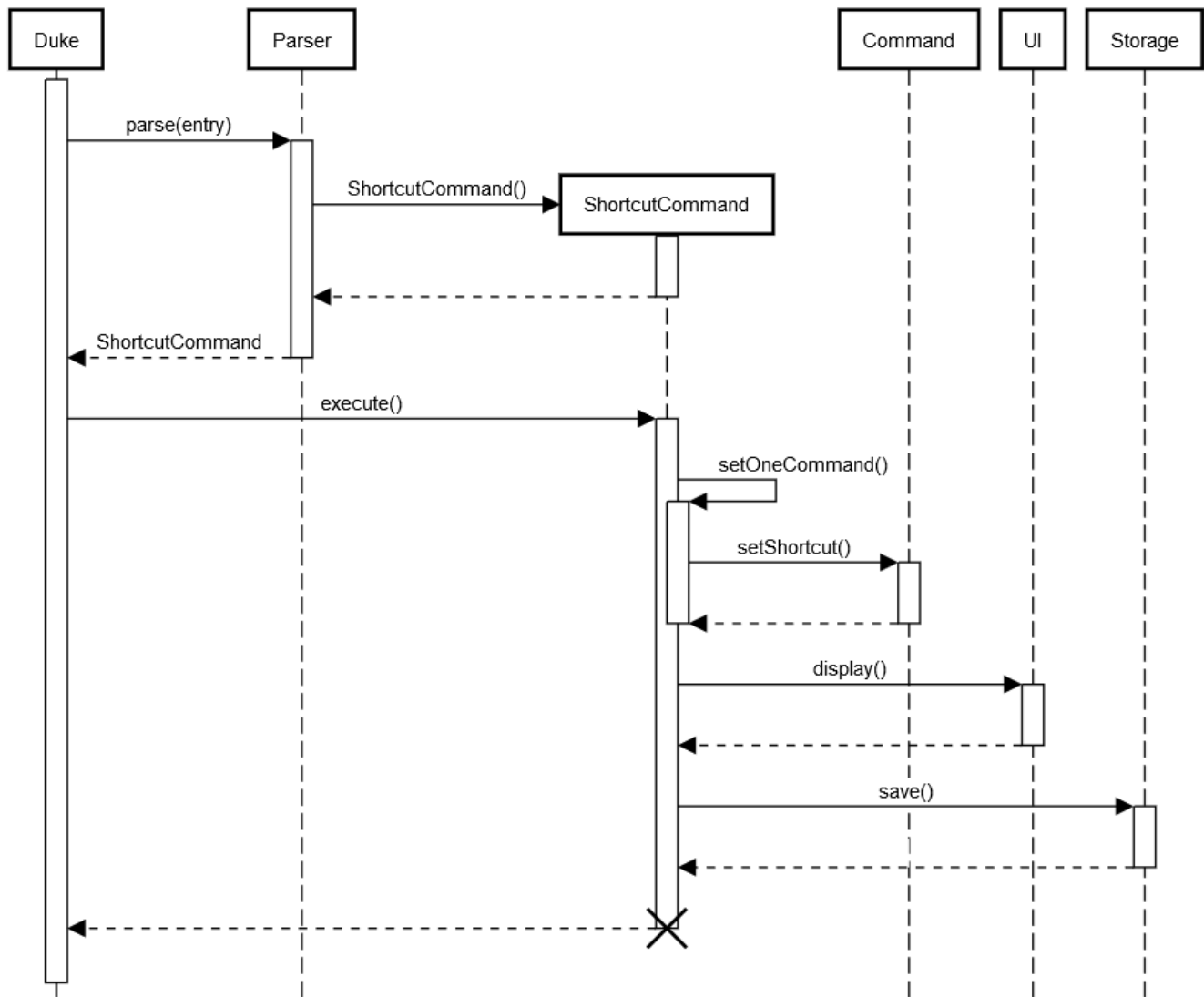
The method setShortcut is static, thus an object Command won't be created

Multi-step one command

The user type the "entry" (not shown in the sequence diagram) as `shortcut CommandName`. Then the console will ask what will be the new name for the shortcut.

Modify a command's shortcut name: one command
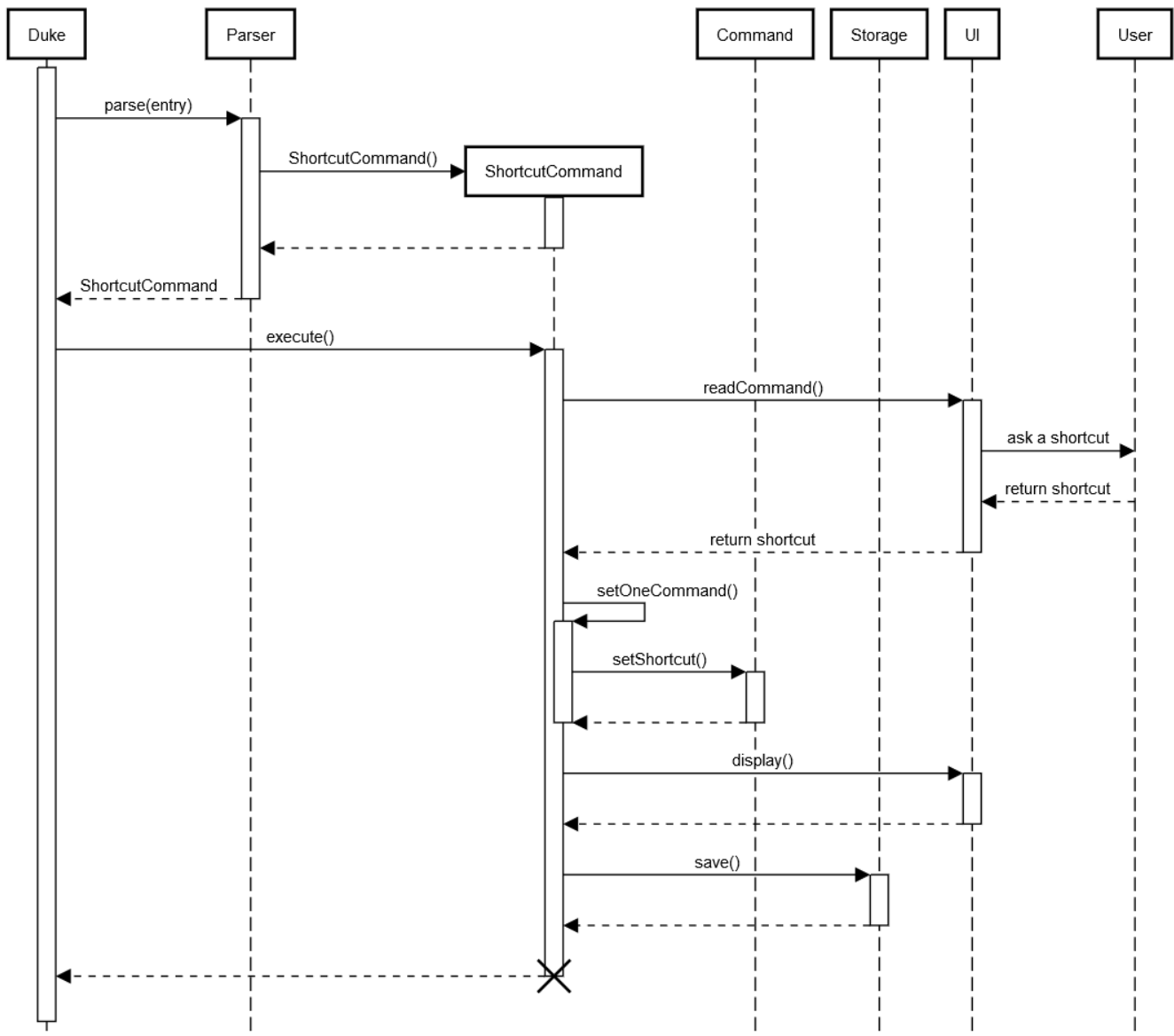
Multi-step every command

The user type the "entry" (not shown in the sequence diagram) as `shortcut`. The console will display one command's name, then the console will ask what will be the new name for the shortcut. The console will repeat until every command have a shortcut.

Modify a command's shortcut name: every command
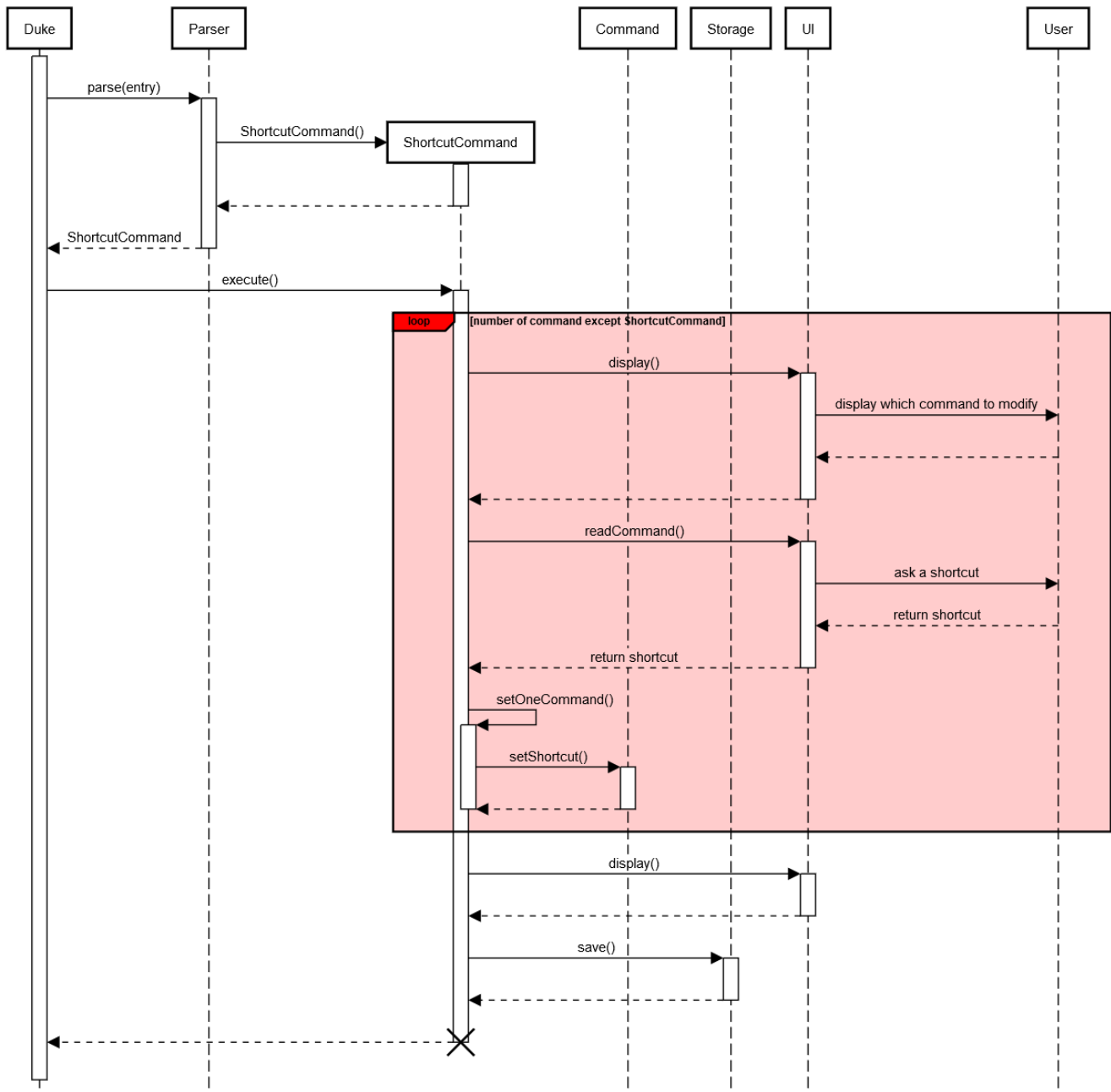
Consideration

- The config file that contains all the name for the shortcut can be edit by hand, because it is faster to edit the config file than doing it via the application.

- When a command's shortcut is set, the default shortcut can still be used

3.1.2. Language

Changing the language mechanism is done by the `LanguageCommand`. For the moment two languages are available : french and english. Only the return message after a command and the error message are changed. After typing the command to change the language, the language is changed at the next execution of the program.

The following are the steps to change a language :

- The user open Le Duc (the program).

- Le Duc create the object `ui` as an instance of `UiEn`.

- The user type `language fr` (the program is previously in english)

- The program will change the config file.

- The user exit the program.

- The user reopen Le Duc.

- Le Duc load the config file with the new language.

- Le Duc create the object `ui` as an instance of `UiFr`.

- The language of Le Duc is french.

Modify a language for Le Duc
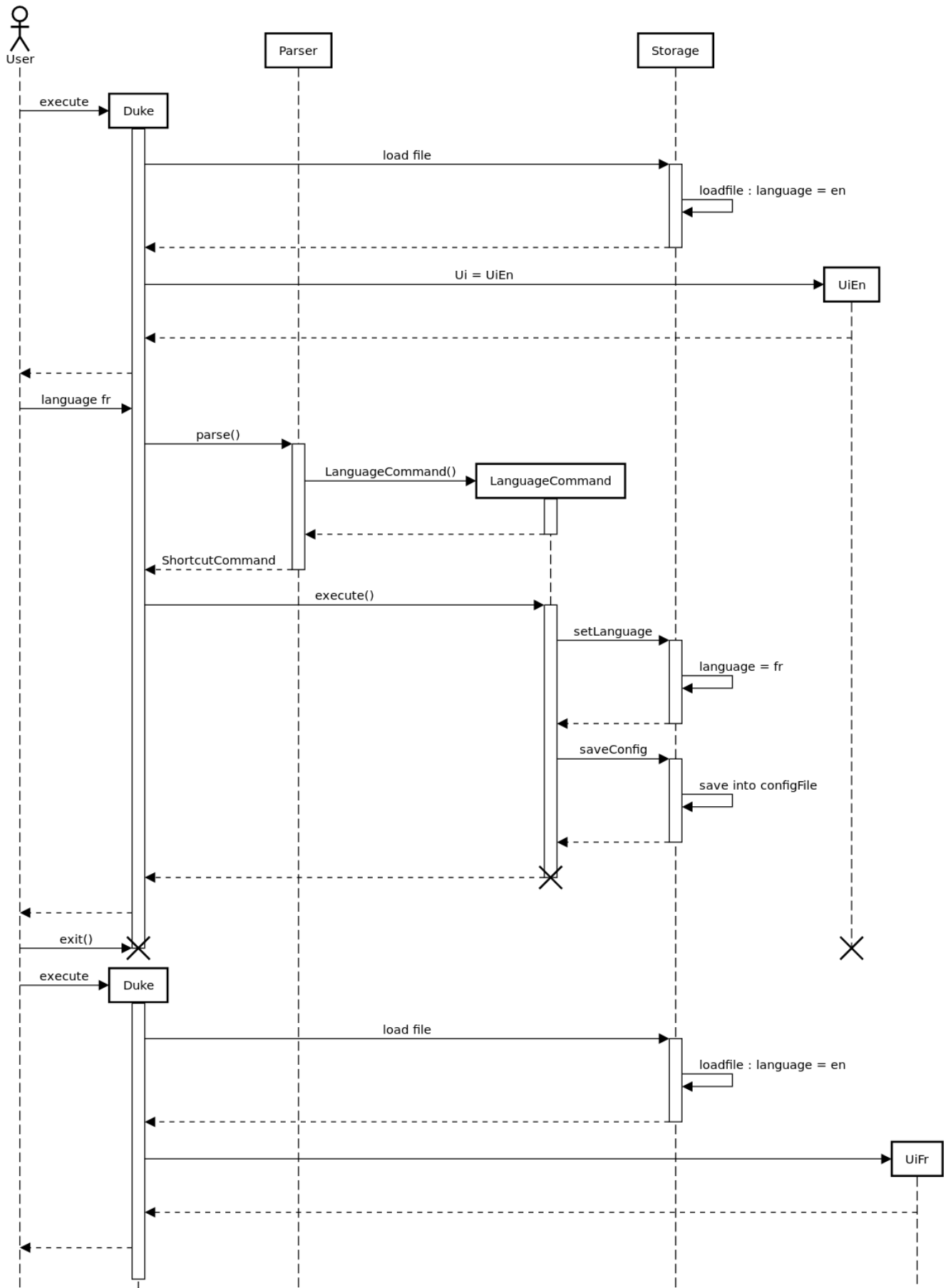
In the sequence diagram, `Parser` and `Storage` should be created and destroyed when Duke is created or destroyed, but for more clarity, it was not represented.

Consideration

- (Current implementation) Each message displayed to the user (error or a message returned by a command) correspond to an abstract method in `Ui` and an override method in `UiFr` and `UiEn`. It was done so because it is easier to add a new language because it is sufficient to create a new class and override the method.

- (Alternative) Make an if statement for each new language and an static attribut in `Ui`. There are less methods and less classes but if a new language is added in the future, every single command and every single exception have to be edited.

3.2. Modify a Task

Several commands allow the user to modify a task: `reschedule`, `postpone`, `snooze`, `edit` and `prioritize`. As every other command, these commands extend Command. As these commands relate to the modification of tasks, each command need to write into the data file after its execution.

3.2.1. Reschedule an event task

When rescheduling an event, two dates can't clash. This verification is done with the verifyConflictDate method which is in the TaskList class. Indeed, all task dates are needed to verify if there is a conflict. So, this allows to improve the cohesion.

Sequence Diagram : reschedule
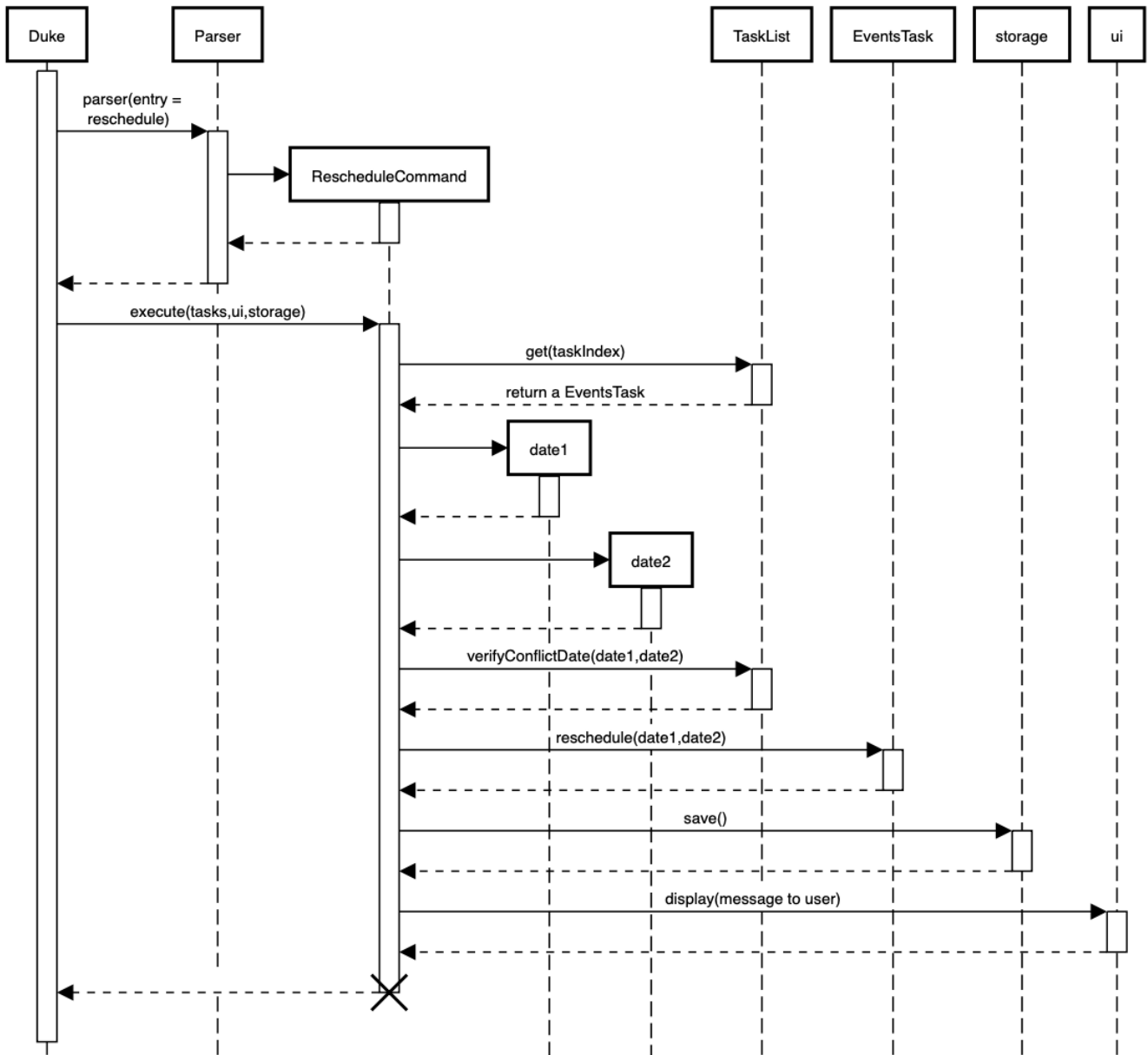
3.2.2. Snooze an homework task

Snooze is applicable to a homework task. The snooze time is fixed at 30 minutes(it could be easily changed in the snoozeLocalDateTime() method of Date.

Sequence Diagram : Snooze
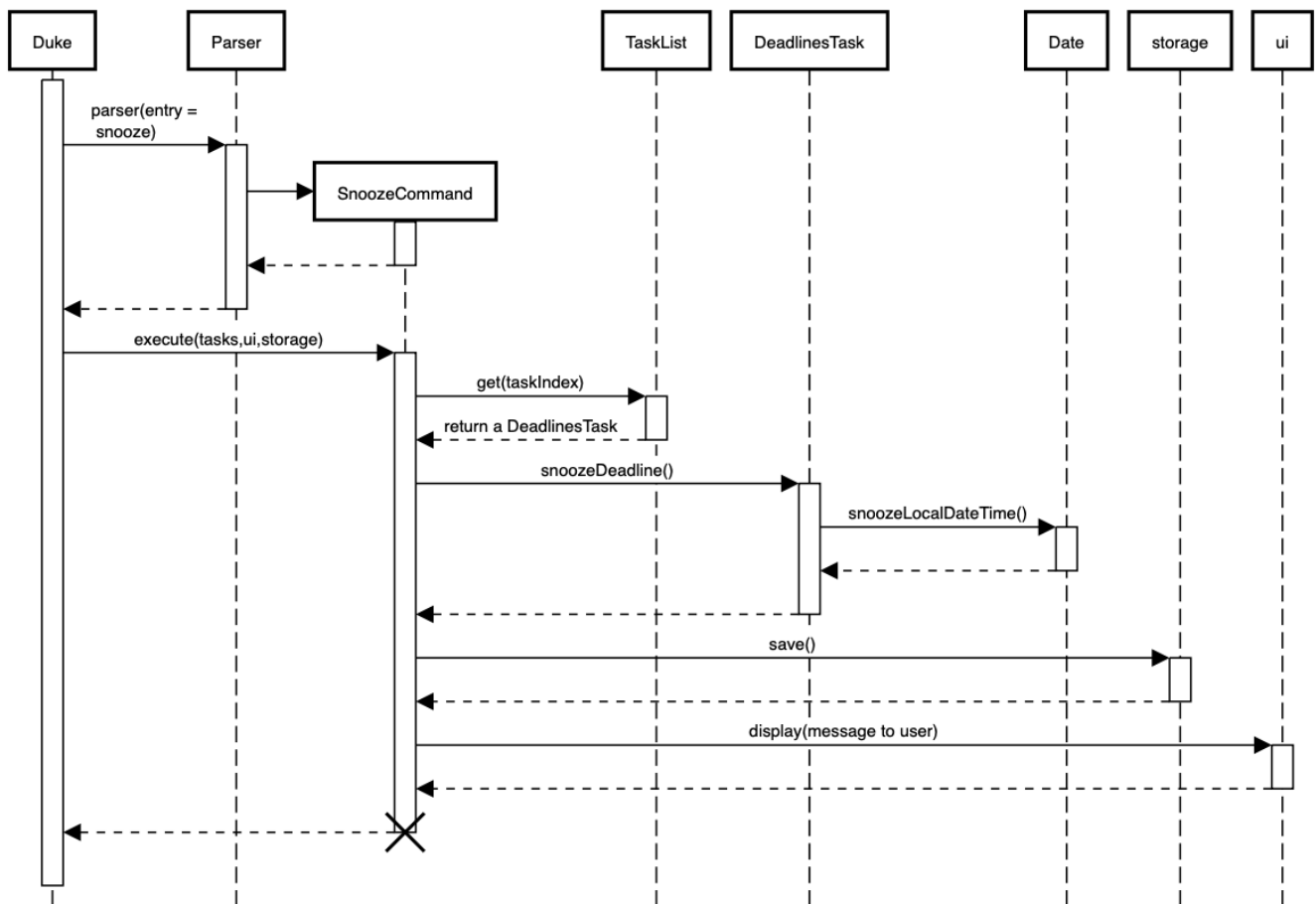
3.2.3. Postpone an homework task

Postpone is also only applicable to a homework task. The new date should be after the old one. This is verified inside the execution of the postponeCommand.

Sequence Diagram : Postpone
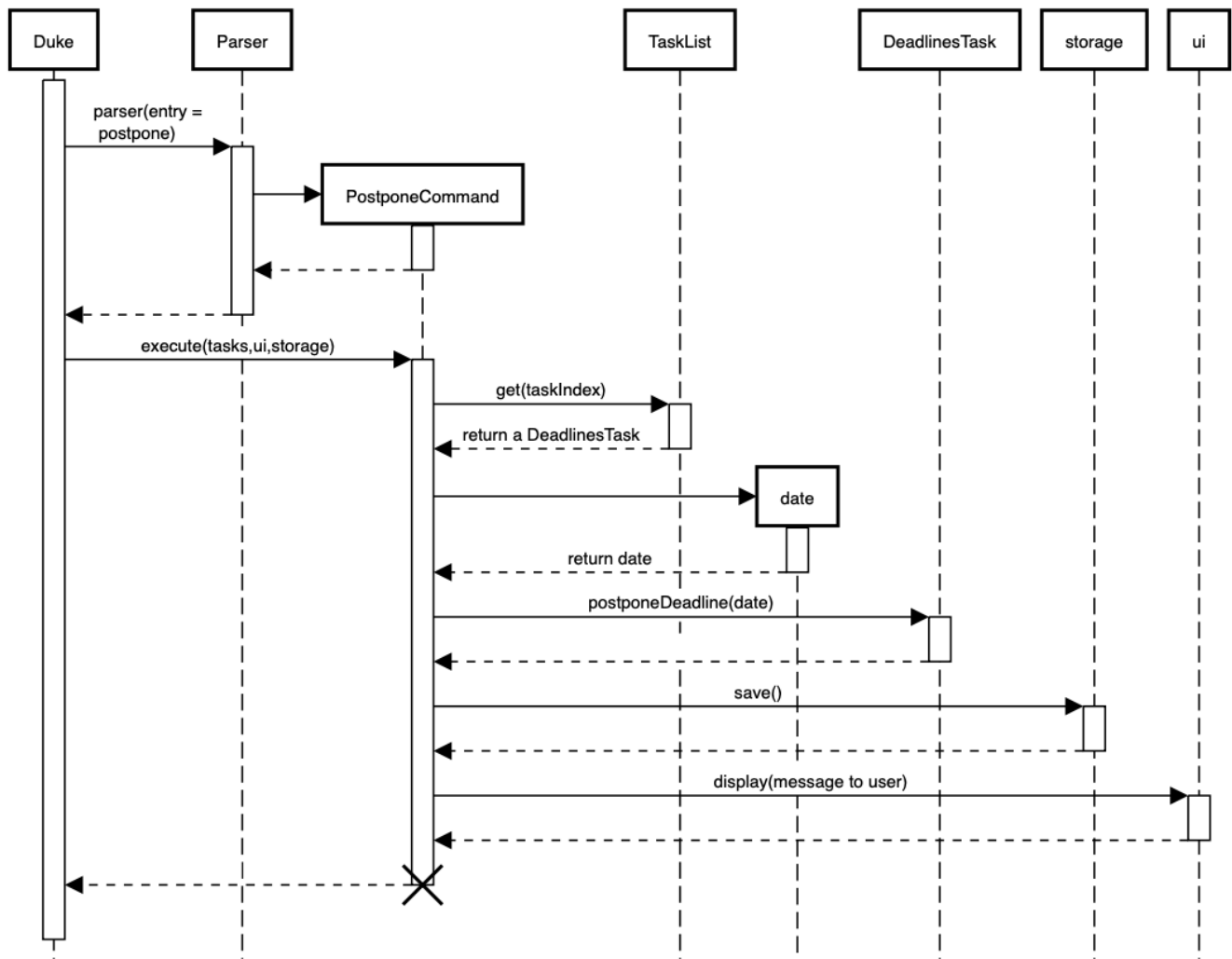
3.2.4. Edit a task

- Multi-steps command: to edit a task, the user has to follow these instructions:

  1. `edit`

  2. All of the tasks will be displayed, you have to choose a task INDEX

  3. Depending on the type of task:

     - If it is a todo task, you have to enter the new DESCRIPTION

     - If it is not a todo task, you have to choose 1) if you want to edit the description or 2) if you want to edit the date

       - Then, enter the new DESCRIPTION or the new DATE of the task

The sequence diagram shows the interactions between different classes when the user want to edit the description an homework or event task with a multi-steps edit command.

Sequence Diagram : Edit multi-steps
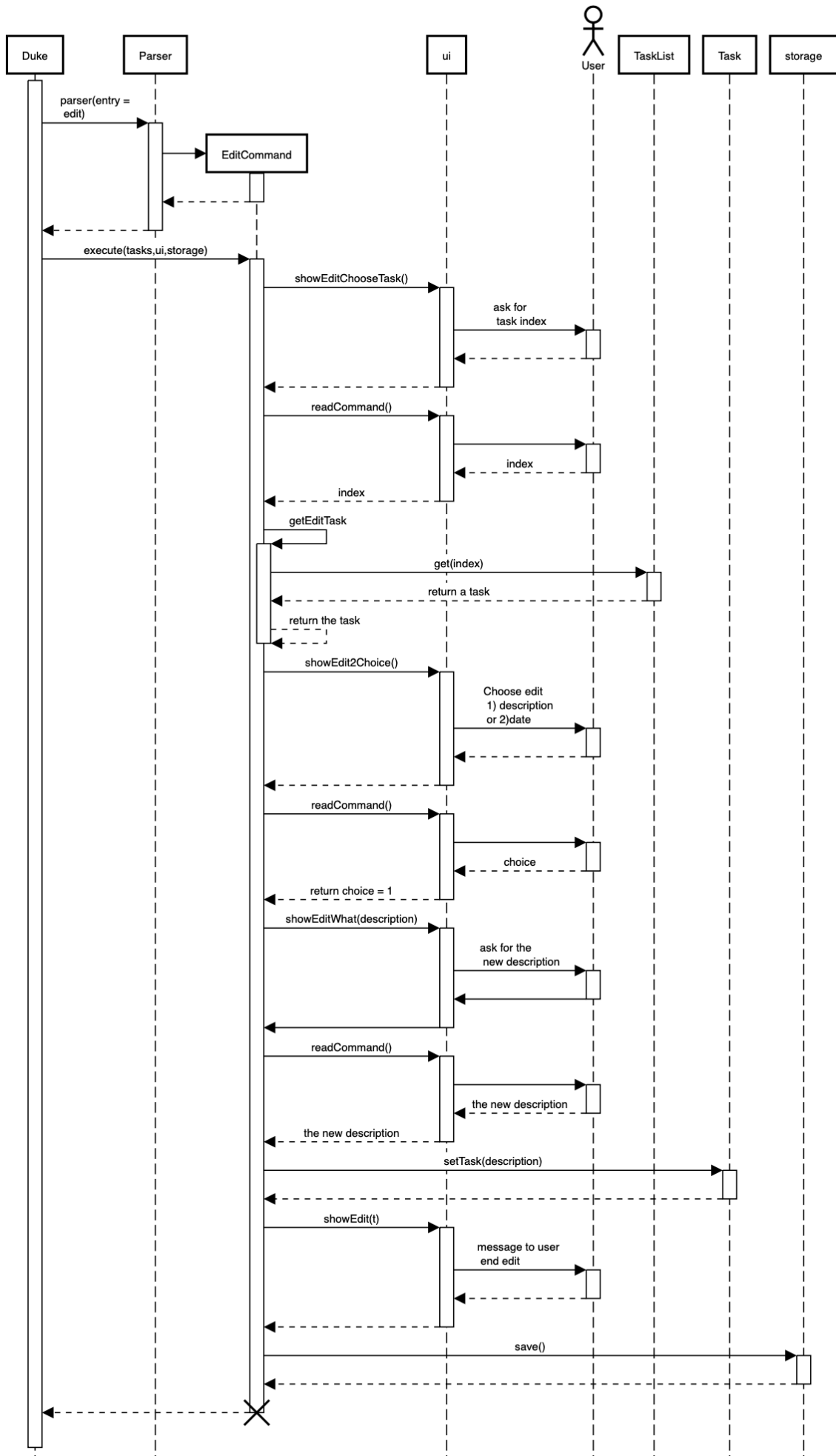
- For one shot command:
  - edit the description: `edit INDEX description DESCRIPTION`
  - edit the date of an homework task: `edit INDEX /by DATE`
  - edit the period of an event task: `edit INDEX /at DATE - DATE`

The sequence diagram shows the interactions between different classes when the user input `edit 2 description DESCRIPTION`.

Sequence Diagram : Edit one shot
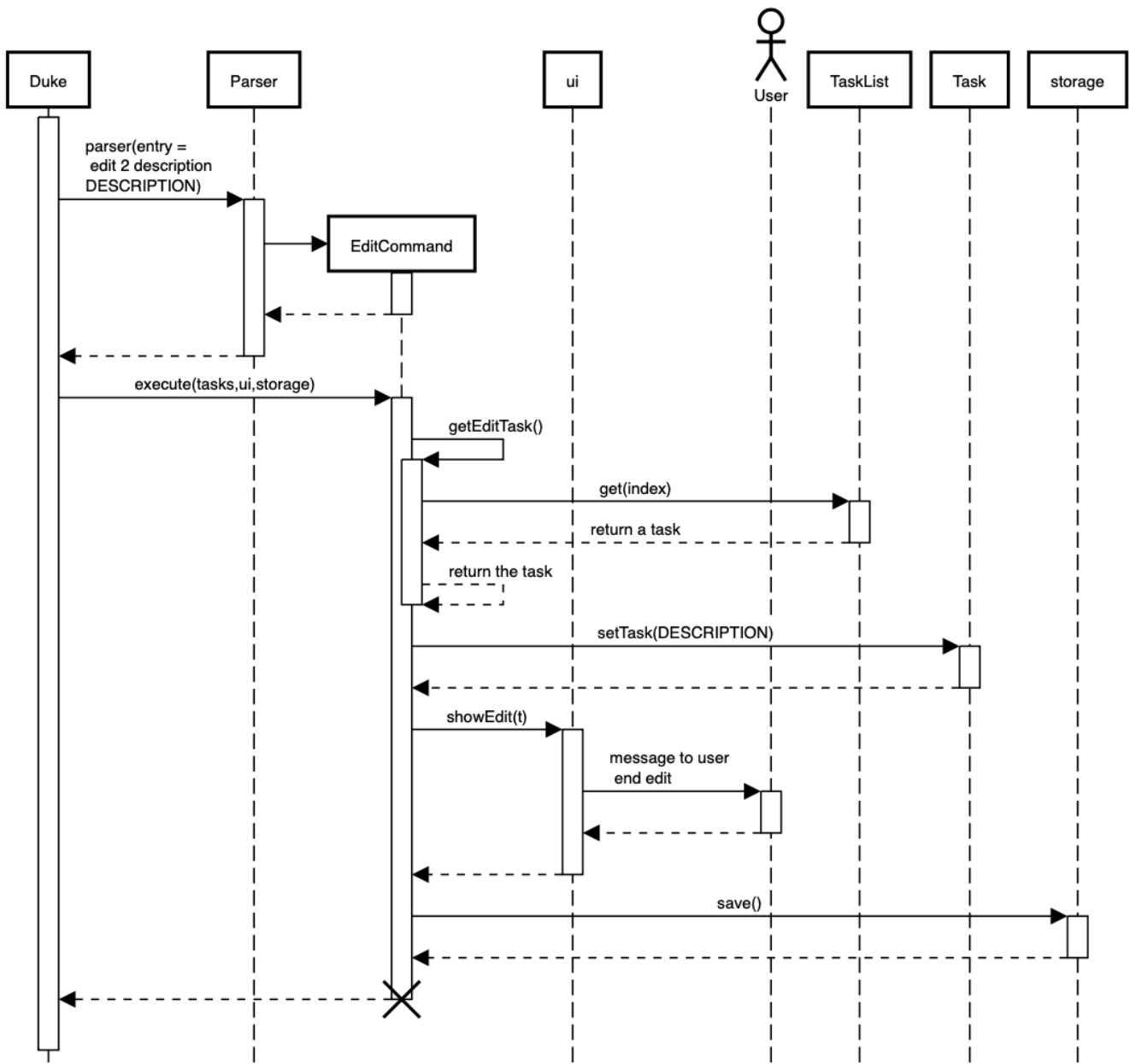
3.2.5. Prioritize a task

A task has initially a priority of 5. The priority of a task goes from 1 to 9. This command allows the user to change the priority of a task.

The sequence diagram show the interactions between different classes when the user wants to change to priority of the first task to 2.

Sequence Diagram : Prioritize
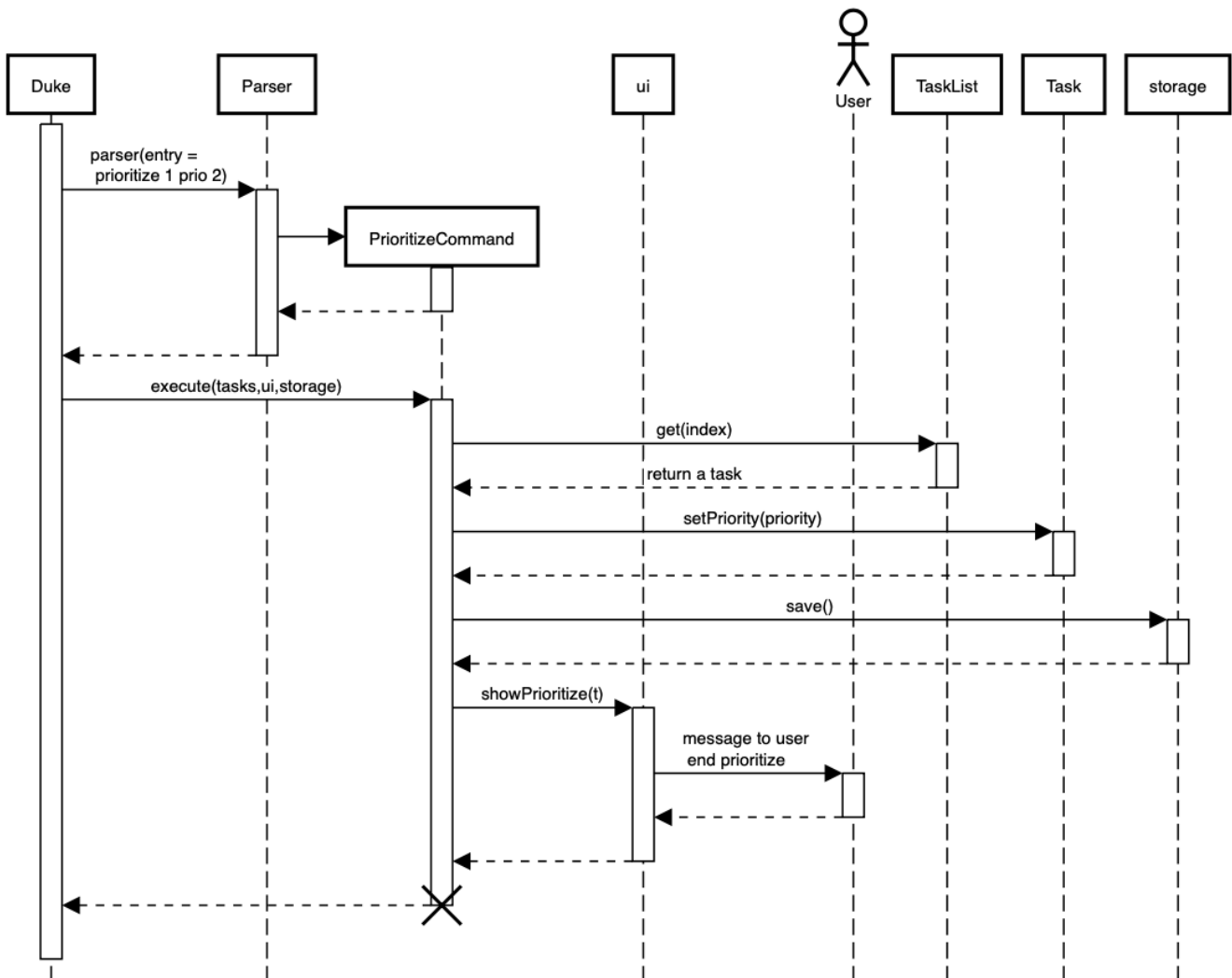
3.2.6. Consideration

There are two different commands for modifying the priority (`prioritize`) and the description/date (`edit`) o f a task. Indeed, the edit command is considered to be used when a user have initially created a incorrect task, whereas the prioritize command is supposed to be used regularly as the priority of a task generally increase with the time. However, these two commands are obviously easy to combine into one command.

3.3. Sort the task list

Sort all task by date/description/priority/type of task/ done or not: `sort SORTTYPE` SORTTYPE is either date, description, priority, type, done

- Sorting by date will sort tasks in chronological order
- Sorting by description will sort the descriptions in alphabetical order
- Sorting by priority will sort tasks in ascending urgency
- Sorting by type will sort tasks depending on its task type (event, homework, todo)
- Sorting by done will sort tasks depending on it the task is done or not.

To implement the sort command, the comparing static method of Comparator interface introduced in Java 8 is used. So, here the sort key are the desciption or the priority of the task.

3.4. Remind

The Remind feature is done by the RemindCommand. Along with all of the other implemented commands, it extends Command. The feature will process each tasks date/timestamps to order them, and then remind the user of the 3 earliest, unfinished tasks. The following methods were implemented in this feature:

- `filterTasks` - Extracts the Homework and Event tasks into a seperate ArrayList

- `sort` - Orders the filtered TaskList in chronological order.

- **Sequence Diagram of the Remind Feature:**
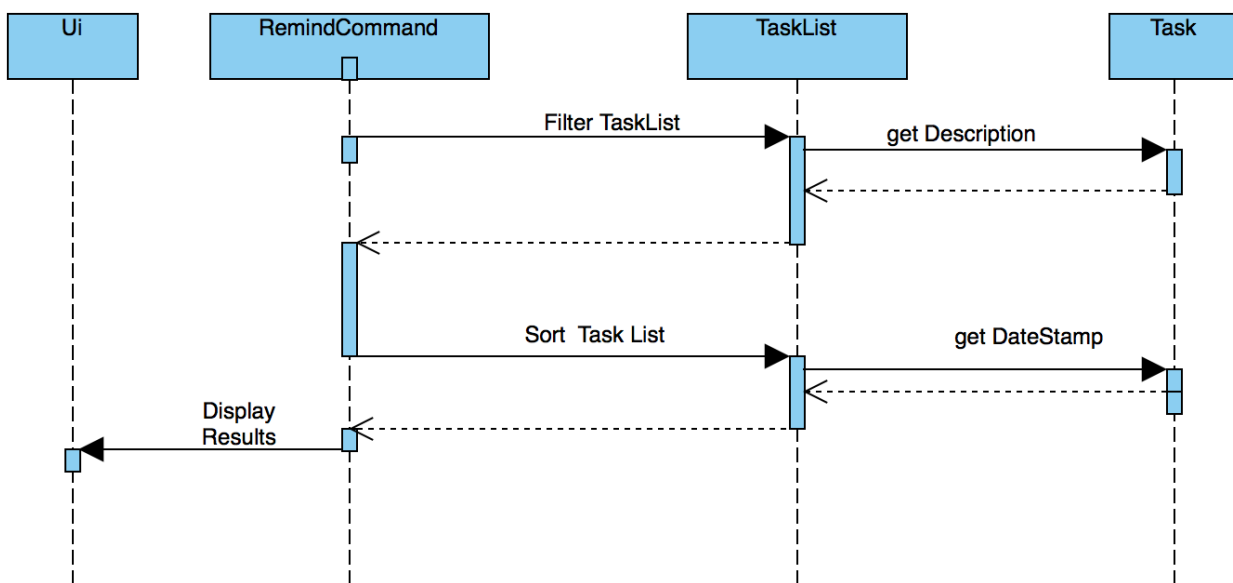
There are 4 cases:

- TaskList contains only Homework and Event objects
- TaskList contains only Todo objects
- TaskList contains a mix of all objects
- TaskList contains no objects

The task of sorting the tasklist in chronological order becomes challenging as not all tasks have associated timestamps. This problem is divided into 4 subproblems, each of which are handled seperately.

3.4.1. TaskList contains only Homework/Event tasks

- The original TaskList is passed through a filter to isolate the Homework and Event tasks.
- The TaskList.extractTodo() method will attempt to isolate all Todo tasks. It will return an empty array in this case

- After filtering by TaskList.filterTasks(), the filtered TaskList will be equal to the original TaskList, as there are no Todo objects to filter out in this particular case.

- The filtered TaskList will then be sorted by TaskList.sort().

- The sort() method will call each tasks task.getDate() method in preparation for sorting. For "Event" tasks, the first timestamp will be used for the purposes of sorting.

- After performing insertion sort, the array of Todo's will be appended to the end of the sorted list. In this case, there are no "todo" tasks, so nothing will be appended.

- The first 3 most upcoming tasks will be displayed to the user.

- **Output:**

```
remind
1. [H][X] d1 by: 14/09/2019 22:33 [Priority: 5]
2. [E][X] e1 at: 21/09/2019 00:00 - 28/10/2019 22:22 [Priority: 5]
3. [H][X] d2 by: 22/09/2019 22:33 [Priority: 5]
```

3.4.2. TaskList only contains Todo tasks

- The original TaskList is passed through a filter to isolate the Homework and Event tasks. Because there are only Todo tasks, it will return an empty array.

- The TaskList.extractTodo() method will attempt to isolate all todo tasks. It will return the original tasklist in this case, becase every task is a "todo"

- The filtered TaskList will then be sorted by TaskList.sort(). The method will return an empty array, because the input array containing Homework and Event tasks is empty.

- The sort method will return an empty array, as the input array of Homework/Event tasks is empty in this case.

- The array of Todo's is appended to the empty, returned array from TaskList.sort(), resulting in the original tasklist of only Todo tasks.

- **The original tasklist is considered to be the sorted tasklist. The order in which the Todo's were created is treated as an "implicit order".**

- The first 3 most upcoming tasks will be displayed to the user.

- **Output:**

```
remind
1. [T][X] todo1 [Priority: 5]
2. [T][X] todo2 [Priority: 5]
3. [T][X] todo3 [Priority: 5]
```

3.4.3. TaskList contains a mix of all task types

- The original TaskList is passed through a filter to isolate the Homework and Event tasks.

- The TaskList.extractTodo() method will attempt to isolate all todo tasks. It will return an array of

all todo's in this case

- After filtering by TaskList.filterTasks(), the filtered TaskList will contain only Homework and Event tasks.

- The filtered TaskList will then be sorted by TaskList.sort().

- The sort() method will call each tasks task.getDate() method in preparation for sorting. For "Event" tasks, the first timestamp will be used for the purposes of sorting.

- After performing insertion sort, the array of Todo's will be appended to the end of the sortedlist. If a task is a "todo" it is assumed that it has a "soft deadline" unlike a Homework task. Therefore, it is pushed to the bottom of the sorted tasklist.

- The first 3 most upcoming tasks will be displayed to the user.

- **Output:**

```
remind
1. [H][X] d1 by: 14/09/2019 22:33 [Priority: 5]
2. [E][X] e1 at: 21/09/2019 00:00 - 28/10/2019 22:22 [Priority: 5]
3. [T][X] t1 [Priority: 5]
```

3.4.4. TaskList Contains No Objects

```
------------------------------------------------------------------------------
    There are no upcoming tasks in your list
------------------------------------------------------------------------------
```

3.4.5. Consideration

- Sorting the TaskList in place was considered, but it reduced cohesion of the design, as the same function is reused by other features.

- It was considered to only remind the user of tasks that are coming up in the next week, but that would limit its potential utility ---

4. Appendix A: Project Scope

Target user profile:

- High School student

- Has a busy academic and extracurricular life

- Comfortable with the command line

- Requires a schedule management software to otpimize day to day decisions

- Value proposition: A student will be able to better manage their time, and be more productive. ---

5. Appendix B: User Stories

Target	User Stories	Priority
As a student,	I want to sort my task by type of task/by date/by description/by tag	* * *
As a student,	I want to prioritize different task	* * *
As a student,	I want to have recurrent task	* * *
As a student,	I want to display the task by day/week/month/year	* * *
As a student,	I want to edit a task	* * *
As a student,	I want to a better find feature (fuzzy matching)	* * *
As a student,	I want to postpone a task	* * *
As a student,	I want to have shortcut	* * *
As a student,	I want to display two or more different task list (for example professional and personal)	* * *
As a student,	I want the app to send me an email about my task	* *
As a student,	I want to display only the task that hasn't been done yet	* *
As a student,	I want the app to alarm me when a deadline is near	* *
As a student,	I want to import from other task list	* *
As a student,	I want to have subtask	* *
As a student,	I want to see which task was done late, which task hadn't been done and which task was done on time	* *
As a student,	I want to set a reminder	* *
As a student,	I want to see my progression	* *
As a student,	I want to add an excepted time	* *
As a student,	I want to see the time left until the allocated time for that task is over	* *
As a student,	I want to have different languages	* *
As a student,	I want to know the location, the address	* *
As a student,	I want to combine two tasks	* *
As a student,	I want to visualize the task (GUI)	*
As a student,	I want to have a login and a password or have a profile	*

Target	User Stories	Priority
As a student,	I want to customize the welcome message	*
As a student,	I want to have secret task	*
As a student,	I want to have shared task	*
As a student,	I want to delete a profile (like an admin)	*
As a student,	I want to have some statistics	*
As a student,	I want to see others students task	*
As a student,	I want to have a message feature	*

6. Appendix C: Use Cases

- **Edit Command**: (only in multi-step commands yet)

  - **System**: Le Duc

  - **Actor**: High school student

  - **Use Case**: Modify task

    - User will type "edit"

    - Le Duc will list the entire task list

    - User type the index of the task

    - Le Duc asks which part will be modified if it is not a Todo task

    - User answer and modify

- **Shortcut**:

  - **System**: Le Duc

  - **Actor**: High school student

  - **Use Case**: Create shortcut

    - User type "shortcut"

    - Le Duc will show the first command to be modified

    - User type the shortcut for that command

    - Le Duc will show the second command to be modified

    - User type the shortcut for that command

    - ...

    - Le Duc will show the k command to be modified

    - User type the shortcut for that command

    - Le Duc prompt an error, because there is a conflict between two shortcuts, and will ask to enter a new shortcut

- User type another shortcut for that command

- ...

- Le Duc shows all the shortcuts ---

7. Appendix D: Non Functional Requirements

- **Task list size requirement**: The user is a super busy students, so he will have a lot of task. Size of task list possibly infinite (use of Arraylist, depends on the computer and the storage doesn't use much as it is a written file).

- **Quality requirement**: The system is easy to understand and to be handled by a new user.

- **Mastery requirement**: The system is easy to be mastered, the typing of new task should be easy and fast.

- **Disaster recovery requirement**: If the system crash, the user shall find all his tasks in the storage file.

8. Appendix E: Glossary

- **Fuzzy matching**: When searching for task descriptions via keyword, the "Sorensen-Dice" Fuzzy Matching algorithm is used to return top matches. This ensures that typos in the user query does not affect search performance

- **Recurrent task**: A task that repeat every day/week/month...

9. Appendix F: Product Survey

10. Appendix G: Instructions for Manual Testing

10.1. G.1. Launch and Shutdown:

- Download the Jar file and copy it into an empty folder

- Run the Jar file from the command line, with command "[CS2113-T16-1][Le Duc].jar"

10.2. G.2. View all Commands:

- **View all the commands the user can make**

- Command: `help`
- Test Case: `help`
  - Expected:

```
help
--------------------------------------------------------------------------------
All command will be display as :
commandName [PARAMETERS] : description of the command
All parameters will be written in UPPER_CASE
Parameters are :
DESCRIPTION : the description of a task
SORTTYPE : date, description, priority, type or done
DATE : the date of a task
INDEX : the index of the task (goes from 1 to ...)
KEYWORD : the keyword to find a task
WELCOME: the welcome message
DATEOPTION
Date format is DD/MM/YYYY HH:mm except for show
All blank space should be respected
Here are the list of all command:
todo DESCRIPTION prio INDEX: create a todo task ( prio index is optional) with
priority index
homework DESCRIPTION /by DATE prio INDEX recu DATEOPTION INDEX: create a homework task
( prio index is optional) with priority index (recu is optional) with a recurrence of
DATEOPTION(day, week, month) and the number of recurrence of INDEX
event DESCRIPTION /at DATE - DATE prio INDEX  recu DATEOPTION INDEX: create an event
task ( prio index is optional) with priority index (recu is optional) with a
recurrence of DATEOPTION(day, week, month) and the number of recurrence of INDEX
list : show all the tasks
bye : exit the application
done INDEX : mark as done the task of index INDEX
delete INDEX : delete the task of index INDEX
find KEYWORD : find the task with a keyword
snooze INDEX : snooze a task of index INDEX
postpone INDEX /by DATE : postpone a deadline task
sort SORTTYPE : Sort all task by date/description/priority/type/done or not
reschedule INDEX /at DATE - DATE : reschedule an event task
remind : remind the first three task
setwelcome WELCOME : customize the welcome message
edit :
For multi-step command : 'edit' and then follow the instructions
For one shot command:
edit the description: 'edit INDEX description DESCRIPTION'
edit the date of an homework task: 'edit INDEX /by DATE'
edit the period of an event task: 'edit INDEX /at DATE - DATE'
show DATEOPTION DATE: show task by day/dayofweek/month/year ( day format is
DD/MM/YYYY; dayofweek format is monday,tuesday...; month format is MM/YYYY; year
format is YYYY)
prioritize INDEX prio INDEX : give priority to task
unfinished: Find and display all unfinished tasks
language LANG: change the language of the program at the next execution. LANG is equal
to en or fr
help : show the list of all command
--------------------------------------------------------------------------------
```

10.3. G.3. Adding a Todo Task:

- **Add a task of type Todo**

- Command `todo DESCRIPTION` or `todo DESCRIPTION prio INDEX`

- Test Case `todo read book`

  ◦ Expected:

    ▪ The task will be given a default priority of 5 (Priorities range from 1-9)

```
--------------------------------------------------------------------------------
Got it. I've added this task:
        [T][X] read book [Priority: 5]
Now you have 14 tasks in the list.
--------------------------------------------------------------------------------
```

- Test Case `todo sell book prio 3`

  ◦ Expected:

```
  --------------------------------------------------------------------------------
        Got it. I've added this task:
          [T][X]  sell book  [Priority: 3]
        Now you have 15 tasks in the list.
--------------------------------------------------------------------------------
```

10.4. G.4. Adding a Homework Task:

- **Add a task of type Homework**

- Command `homework DESCRIPTION /by DATE` or `homework DESCRIPTION /by DATE prio INDEX` or `homework DESCRIPTION /by DATE recu DATETYPE NBRECU`

  ◦ The second INDEX can't be less than 1 or greater than 9.

  ◦ 1 is the least urgent, 9 is the most urgent

  ◦ DATETYPE can be day, week, month

  ◦ NBRECU indicate the number of recurrence +1 you want

  ◦ If you want to add recurrence and priority, prio should be before recu

  ◦ The NBRECU can't be negative

  ◦ Depending on the DATETYPE, the NBRECU have a maximum threshold: for day, NBRECU can't exceed 30, for week NBRECU can't exceed 26, for month NBRECU can't exceed 12. Every time it exceeds, it will be brought back to the threshold

- Test Case `homework CS4239: lab3 /by 21/09/2019 23:59`

- Expected:

  - Homework is given a default priority of 5

```
-------------------------------------------------------------------------------
        Got it. I've added this task:
          [H][X] CS4239: lab3 by: 21/09/2019 23:59 [Priority: 5]
        Now you have 16 tasks in the list.
-------------------------------------------------------------------------------
```

- Test Case `homework CS4239: lab4 /by 25/09/2019 23:59 prio 7`

  - Expected:

    - The task is given a priority of 7

```
 -------------------------------------------------------------------------------
        Got it. I've added this task:
          [H][X] CS4239: lab4 by: 25/09/2019 23:59 [Priority: 7]
        Now you have 17 tasks in the list.
-------------------------------------------------------------------------------
```

- Test Case `homework test code /by 25/09/2019 23:59 recu day 3`

  - Expected:

    - Three recurrent tasks are also created, one for each of the next 3 days

```
-------------------------------------------------------------------------------
        Here are the tasks in your list:
        18. [H][X] test code by: 25/09/2019 23:59 [Priority: 5]
        19. [H][X] test code by: 26/09/2019 23:59 [Priority: 5]
        20. [H][X] test code by: 27/09/2019 23:59 [Priority: 5]
        21. [H][X] test code by: 28/09/2019 23:59 [Priority: 5]
-------------------------------------------------------------------------------
```

10.5. G.5. Adding an Event task:

- **Creates a task of type Event**

- Command: `event DESCRIPTION /at DATE - DATE` or `event DESCRIPTION /at DATE - DATE prio INDEX` or `event DESCRIPTION /at DATE - DATE recu DATETYPE NBRECU`

  - When creating an event, two dates can't clash

  - The second INDEX can't be less than 1 or greater than 9.

  - 1 is the less urgent, 9 is the most urgent

  - DATETYPE can be day, week, month

  - NBRECU indicate the number of recurrence +1 you want

- If you want to add recurrence and priority, prio should be before recu

- If, by creating a recurrence, the date of the next occurence of the event is in conflict with the 4 previous one, it will

- You can't create a recurrence of an event when the second date minus first date > DATETYPE: For example 07/12/2019 09:00 - 08/12/2019 11:00 have second date minus first date superior as the day DATETYPE

- The NBRECU can't be negative

- Depending on the DATETYPE, the NBRECU have a maximum threshold: for day, NBRECU can't exceed 30, for week NBRECU can't exceed 26, for month NBRECU can't exceed 12. Every time it exceeds, it will be brought back to the threshold

- If you want to add a priority and a recurrency, make sure the priority flag is entered before the recurrency flag.

- Test Case `event CS4211: mid-term /at 07/10/2000 09:00 - 07/10/2000 11:00`
  - Expected:
    - Event is created with default priority of 5

```
----------------------------------------------------------------------
        Got it. I've added this task:
          [E][X] CS4211: mid-term at: 07/10/2000 09:00 - 07/10/2000 11:00 [Priority:
 5]
        Now you have 22 tasks in the list.
----------------------------------------------------------------------
```

- Test Case `event CS4211: final exam /at 07/12/2019 09:00 - 07/12/2019 11:00 prio 8`
  - Expected:
    - Event is created with priority 8

```
----------------------------------------------------------------------
        Got it. I've added this task:
          [E][X] CS4211: final exam at: 07/12/2019 09:00 - 07/12/2019 11:00
[Priority: 8]
        Now you have 23 tasks in the list.
----------------------------------------------------------------------
```

- **Test Case** `event CS4211: sport /at 07/10/2001 09:00 - 07/10/2001 11:00 recu day 3`
  - Expected:
    - 3 recurrent tasks are created, each one day apart

```
        -------------------------------------------------------------------
        Here are the tasks in your list:
        24. [E][X] CS4211: sport at: 07/10/2001 09:00 - 07/10/2001 11:00 [Priority:
5]
        25. [E][X] CS4211: sport at: 08/10/2001 09:00 - 08/10/2001 11:00 [Priority:
5]
        26. [E][X] CS4211: sport at: 09/10/2001 09:00 - 09/10/2001 11:00 [Priority:
5]
        27. [E][X] CS4211: sport at: 10/10/2001 09:00 - 10/10/2001 11:00 [Priority:
5]
        -------------------------------------------------------------------
```

- **Test Case** `event CS4211: cooking session /at 17/12/2019 09:00 - 17/12/2019 11:00 prio 8 recu week 2`

  - Expected:

    - Will create 3 events, each one being 1 week apart

```
        -------------------------------------------------------------------
        Here are the tasks in your list:
        28. [E][X] CS4211: cooking session at: 17/12/2019 09:00 - 17/12/2019 11:00
[Priority: 8]
        29. [E][X] CS4211: cooking session at: 24/12/2019 09:00 - 24/12/2019 11:00
[Priority: 8]
        30. [E][X] CS4211: cooking session at: 31/12/2019 09:00 - 31/12/2019 11:00
[Priority: 8]
        -------------------------------------------------------------------
```

10.6. G.6. Find a Task by Relevancy:

- **Searches For a task by Character Relevancy (Not Keyword Relevancy!)**

- Command `Find DESCRIPTION`

  - The "Sorensen Dice" character matching algorithm is used to generate relevance scores for each task

  - Tasks are scored based on % character match across the entire description, and a 50% minimum match threshold value.

- Test Case `find homewasrdtyst`

  - Expected:

    - Despite typos, the correct result will be returned

```
    -------------------------------------------------------------------------------
        Here are the most relevant tasks in your list:
        6. [H][V] homeworktest by: 04/04/2004 04:04 [Priority: 5]
        13. [T][X] homework [Priority: 5]


    -------------------------------------------------------------------------------
```

- Test Case `find CS4221:`
  - Expected:
    - Despite Partial queries, relevant tasks are still returned.

```
find CS4221:

    -------------------------------------------------------------------------------
        Here are the most relevant tasks in your list:
        24. [E][X] CS4211: sport at: 07/10/2001 09:00 - 07/10/2001 11:00 [Priority:
5]
        25. [E][X] CS4211: sport at: 08/10/2001 09:00 - 08/10/2001 11:00 [Priority:
5]
        26. [E][X] CS4211: sport at: 09/10/2001 09:00 - 09/10/2001 11:00 [Priority:
5]
        27. [E][X] CS4211: sport at: 10/10/2001 09:00 - 10/10/2001 11:00 [Priority:
5]


    -------------------------------------------------------------------------------
```

10.7. G.7. Delete a Task:

- **Delete a task from the task list.**
- Command: `delete INDEX`
- Test Case `delete 1`
  - Expected:

```
    -------------------------------------------------------------------------------
        Noted. I've removed this task:
            [T][X] td1 [Priority: 5]
        Now you have 29 tasks in the list
    -------------------------------------------------------------------------------
```

- Test Case `delete 500000`
  - Expected:

> ▪ Invalid indices throw an error

```
NonExistentTaskException:
            ☐ OOPS!!! The task doesn't exist
```

10.8. G.8. List all tasks

- **List all tasks in the list**

- Command `list`

- Test Case `list`

  - Expected:

```
--------------------------------------------------------------------------------
        Here are the tasks in your list:
        1. [E][X] e at: 21/09/2019 00:00 - 28/10/2019 22:22 [Priority: 5]
        2. [T][V] td3 [Priority: 5]
        3. [T][V] td4 [Priority: 5]
        4. [H][X] math by: 11/11/2011 01:01 [Priority: 5]
        5. [H][V] homeworktest by: 04/04/2004 04:04 [Priority: 5]
        6. [H][V] science by: 05/05/2005 05:05 [Priority: 2]
        7. [H][X] test by: 01/01/2001 01:01 [Priority: 5]
--------------------------------------------------------------------------------
```

10.9. G.9. Complete a Task

- **Marks a task as done**

- Command `done INDEX`

- Test Case `done 1`

  - Expected

    - ▪ [X] becomes [V] when completing a task

```
--------------------------------------------------------------------------------
        Nice! I've marked this task as done:
        [E][V] e at: 21/09/2019 00:00 - 28/10/2019 22:22 [Priority: 5]
--------------------------------------------------------------------------------
```

- Test Case `done 60000`

  - Expected

    - ▪ Invalid indices will throw NonExistentTaskException

```
NonExistentTaskException:
            ☐ OOPS!!! The task doesn't exist
```

10.10. G.10. Snooze a Homework Task

- **Snooze a Homework task by 30 minutes**

- Command `snooze INDEX`

- Test Case: `snooze 4`

  - Expected:

    - The task is now snoozed by 30 minutes

    - Before:

      ```
      `4. [H][X] math by: 11/11/2011 01:01 [Priority: 5]`
      ```

  - After:

    ```
    ----------------------------------------------------------------------------
    -----
          Noted. I've snoozed this task:
                 [H][X] math by:11/11/2011 01:31
    ----------------------------------------------------------------------------
    -----
    ```

- Test Case `snooze 3`

  - Expected

    - Snoozing a non-Homework task will throw HomeworkTypeException

```
HomeworkTypeException:
            ☐ OOPS!!! The task should be a homework task
```

10.11. G.11. Postpone a Homework Task

- **Postpone a Homework task by a user-specified amount of time**

- Command `postpone INDEX /by DATE`

  - DATE is the new date of the homework task.

  - The new date should be after the old one.

- Test Case `postpone 4 /by 12/12/2011 22:10`

  - Expected:

- The Homework task deadline has been postponed

- Before:

```
`4. [H][X] math by: 11/11/2011 01:01 [Priority: 5]`
```

- After:

```
-------------------------------------------------------------------------------
-----
        Noted. I've postponed this task:
                [H][X] math by:12/12/2011 22:10
-------------------------------------------------------------------------------
-----
```