Liu Peize - Project Portfolio

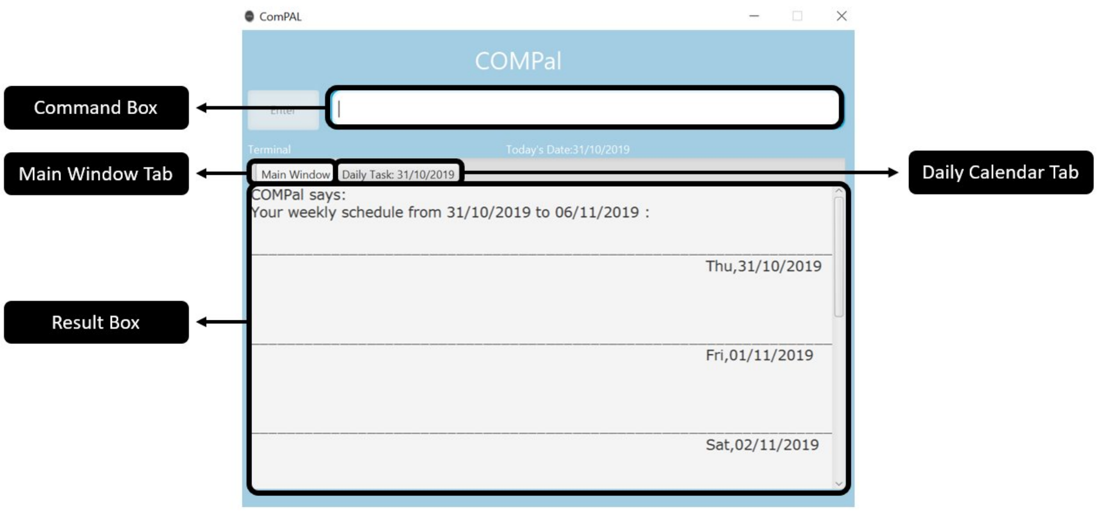
PROJECT: COMPal

Overview

My team of 4 software engineering students and I were asked to build a command line desktop application for our Software Engineering project. We chose to morph from our individual projects Duke to design an application for the hectic schedule for the modern student in mind, which is COMPal. By simply inputting their busy and compact schedule, the application is able to automatically generate a prioritized daily schedule for the user! This ensures that the student can focus on the more important upcoming task! Additionally with features such as reminders of task and also finding of free time slot, COMPal allows the ease of planning for future task.

It is catered to student-users who prefer to use and are adept at using a Command-Line Interface (CLI), while still having a clean Graphical User Interface (GUI) to properly visualize schedules and organize tasks better.

This si what our project looks like:



```
COMPal says:
Your weekly schedule from 31/10/2019 to 06/11/2019 :
________________________________________________
                                   Thu,31/10/2019
________________________________________________
                                   Fri,01/11/2019
________________________________________________
                                   Sat,02/11/2019
```

My role is to design the `help` feature and the non-recursive `deadline` feature. The following sections illustrate these enhancements in more detail, as well as what I have contributed to the user and developer guides regarding these features.

Summary of contributions

- **Major enhancement**: added **the ability to search for help about specific commands**
  - What it does: allows the user to get an overview of all commands and search for the usage of specific commands.
  - Justification: This feature improves the product significantly because a new user could learn how to use this program through this feature.
  - Highlights: This enhancement affects existing commands and commands to be added in future. It required information about

existing and upcoming commands and need to be regularly updated during the developing process.

- **Minor enhancement**: set up the base command for adding a non-recurring deadline type task.
  - **Code contributed**:

    Please click this link to view my code contributions on RepoSense :
    [Contributed code]
    Please click this link to view my contributed pull requests :
    [Contributed pull requests]

- **Other contributions**:
  - Enhancements to existing features:
    - Wrote additional tests for existing features to increase coverage. My contributed code part has coverage of over 95%.
  - Documentation:
  - Wrote all documentation related to the features that I am in charge which can be found in the User Guide and Developer Guide.
  - Community:
    - Reviewed Pull Requests. These Pull Requests can be viewed here.

Contributions to the User Guide

Here are the links to my contributed sections in the User Guide:

- General Commands: Searching for help: `Help`

Below is my addition to the User Guide for the help feature.

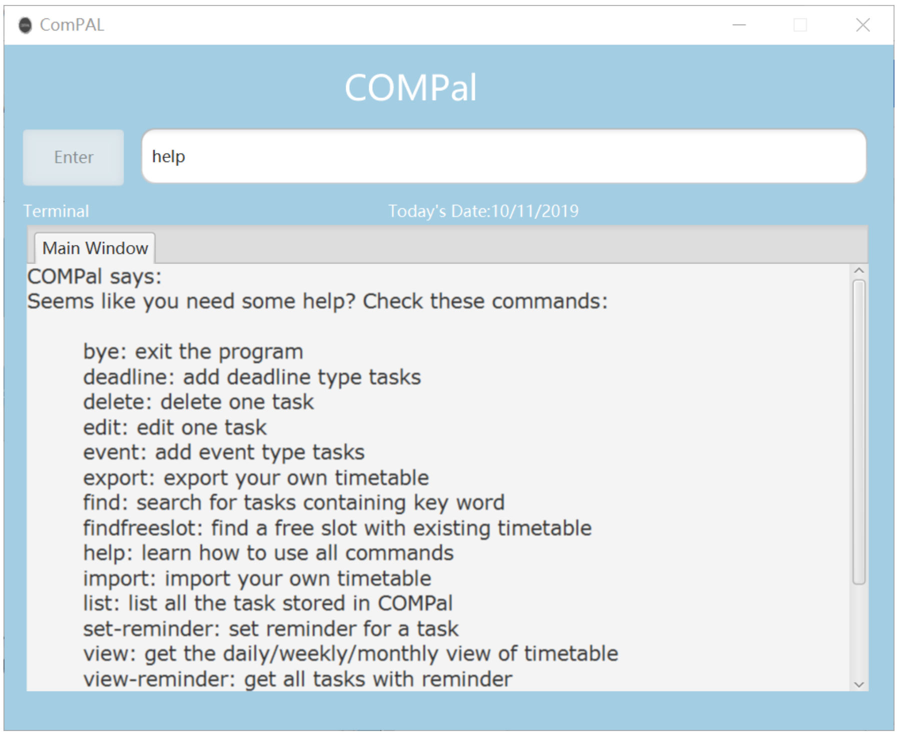
1. Viewing Help: `help`

Can't remember so many tedious commands?

Format for ***general help***: `help` or `TRASH_COMMAND`

You can see a list of all commands available. There is also a guide to tell you how to use this `help` command.

i	`TRASH_COMMAND` is really "trash command" 😉, it could be any command that is invalid.



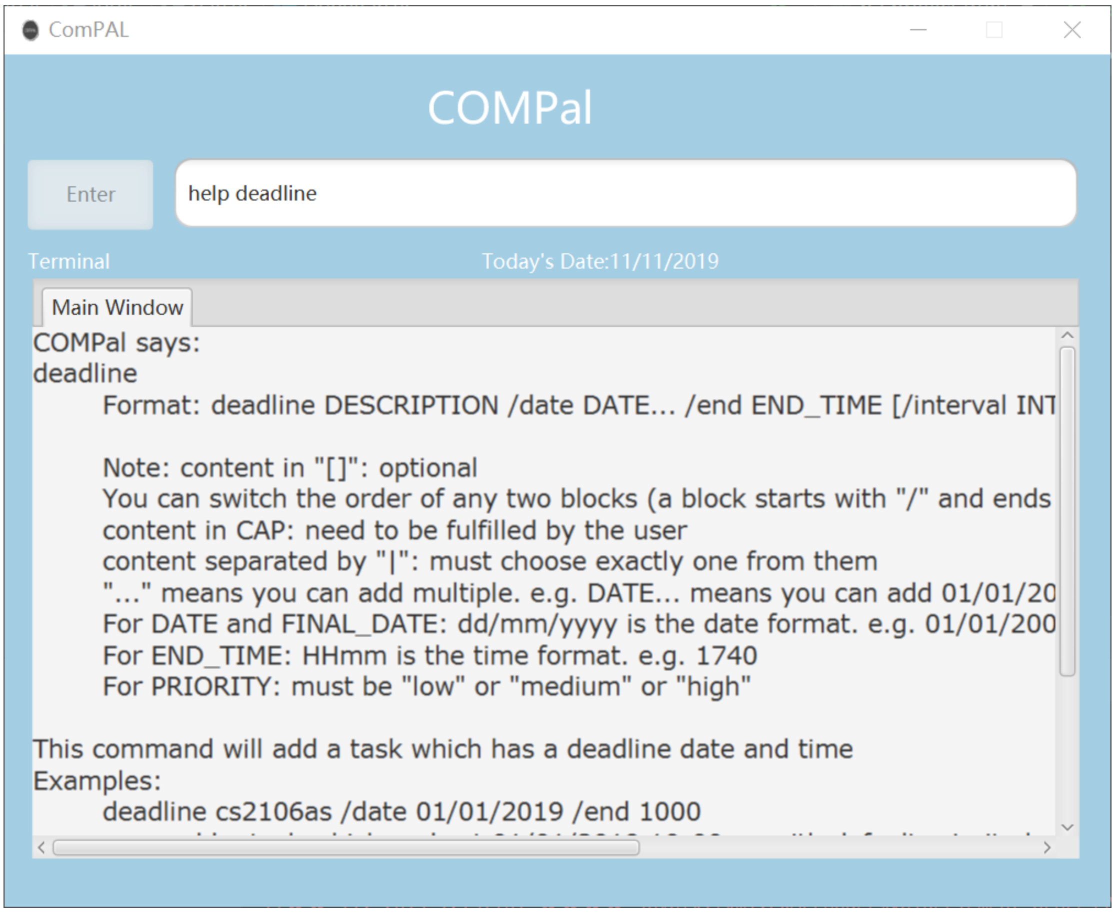
```
COMPal says:
Seems like you need some help? Check these commands:

        bye: exit the program
        deadline: add deadline type tasks
        delete: delete one task
        edit: edit one task
        event: add event type tasks
        export: export your own timetable
        find: search for tasks containing key word
        findfreeslot: find a free slot with existing timetable
        help: learn how to use all commands
        import: import your own timetable
        list: list all the task stored in COMPal
        set-reminder: set reminder for a task
        view: get the daily/weekly/monthly view of timetable
        view-reminder: get all tasks with reminder
```

Figure 1.Output after entering the `help` command.

Format for ***specific help***: `help COMMAND_NAME`

You can use this command to search for specific information about `COMMAND_NAME`.

i	`COMMAND_NAME` is any command name you can see when you do `help` command.



```
help deadline
```

```
COMPal says:
deadline
        Format: deadline DESCRIPTION /date DATE... /end END_TIME [/interval INT

        Note: content in "[]": optional
        You can switch the order of any two blocks (a block starts with "/" and ends
        content in CAP: need to be fulfilled by the user
        content separated by "|": must choose exactly one from them
        "..." means you can add multiple. e.g. DATE... means you can add 01/01/20
        For DATE and FINAL_DATE: dd/mm/yyyy is the date format. e.g. 01/01/200
        For END_TIME: HHmm is the time format. e.g. 1740
        For PRIORITY: must be "low" or "medium" or "high"

This command will add a task which has a deadline date and time
Examples:
        deadline cs2106as /date 01/01/2019 /end 1000
```

Figure 2. Output after entering `help deadline` command.

Contributions to the Developer Guide

Here are the links to my contributed sections in the Developer Guide:

- **Design**: Logic Component
- **Implementation**: Priority Feature, Help Feature
- **Appendix F: Instruction for Manual Testing**: Adding a task, Viewing help

Below is an example of my addition to the Developer Guide for the **Help feature**.

Help Feature

This feature allows the user to search for usage of a command. Whenever the user enters an invalid command, it will be regarded as a help command.

1. Current Implementation

The current implementation allows the user to get the basic information about all commands with any invalid input or specific instructions of one command with **`help`**:

Command: Any possible invalid input or `help`
Upon invoking an invalid command (refer to User Guide for `help` usage), a sequence of events is then executed.
A graphical representation is included in the Sequence Diagram below for your reference when following through the sequence of events. The sequence of events is as follows:

1. The `help` command is passed into the `logicExecute` function of `LogicManager` to be parsed.
2. `LogicManager` then invokes the `processCmd` function of `ParserManager`.
3. `ParserManager`, in turn, invokes the `parseCommand` function of the appropriate parser for the `help` command which in this case, is `HelpCommandParser`.
4. Once the parsing is done, `HelpCommandParser` would instantiate the `HelpCommand` object which would be returned to the `LogicManager`.
5. `LogicManager` is then able to invoke the `commandExecute` function of the returned `HelpCommand` object.
6. In the `commandExecute` function of the `HelpCommand` object, the `HelpCommand` object has the description of the command.
7. With the description of the command, `HelpCommand` will match it with existing commands and the `CommandResult` object will be instantiated with the matched command. If the description is empty `CommandResult` will be instantiated with the default message of basic information of all commands.
8. The `CommandResult` object would then be returned to the `LogicManager` which then returns the same `CommandResult` object back to the **`UI`** component.
9. Finally, the **`UI`** component would display the contents of the `CommandResult` object to the user.

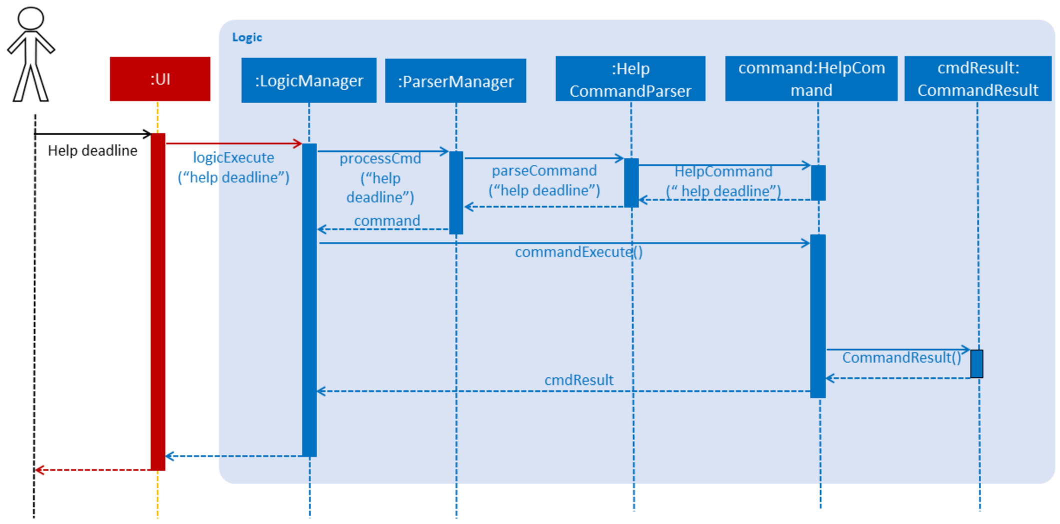


Figure 3. Sequence Diagram executing the `Help` command.

2. Design Considerations

Aspect: Functionality of `help` command

- **Alternative 1 (current choice):** Store a brief help in a string for all commands for invalid command and store detailed instructions for a specific command in strings in respective command classes for `help <command name>`.
  - **Pros:** It is easier to add help for new commands and can modify the help for specific command easily.
  - **Cons:** Need to store things in multiple classes. When adding a new command, need to change strings in both help and the new command class.

- **Alternative 2:** Store the help for all commands in a string in help class when `help` is called.
  - **Pros:** Only need to store everything in help class. When any command changes, only need to modify help command string.
  - **Cons:** The string in help command could be very long and need to be careful to deal with updating.

Alternative 1 was chosen as it is more user-friendly and easier to update for developers. The user can get an overview of all commands first and then search for usage of specific commands. The developer can add new command's help easily. If more commands are added, alternative 2 requires the developer to look through a huge page of texts to find the format of one instruction.

3 Future Implementation

1. More details and examples of each command. With more examples, the user could have a better idea of the full functions of each command.

2. Another method to show the list of full instructions regarding all commands. It is more intuitive for users who use COMPal for the first time to know how to use it in a shorter time.