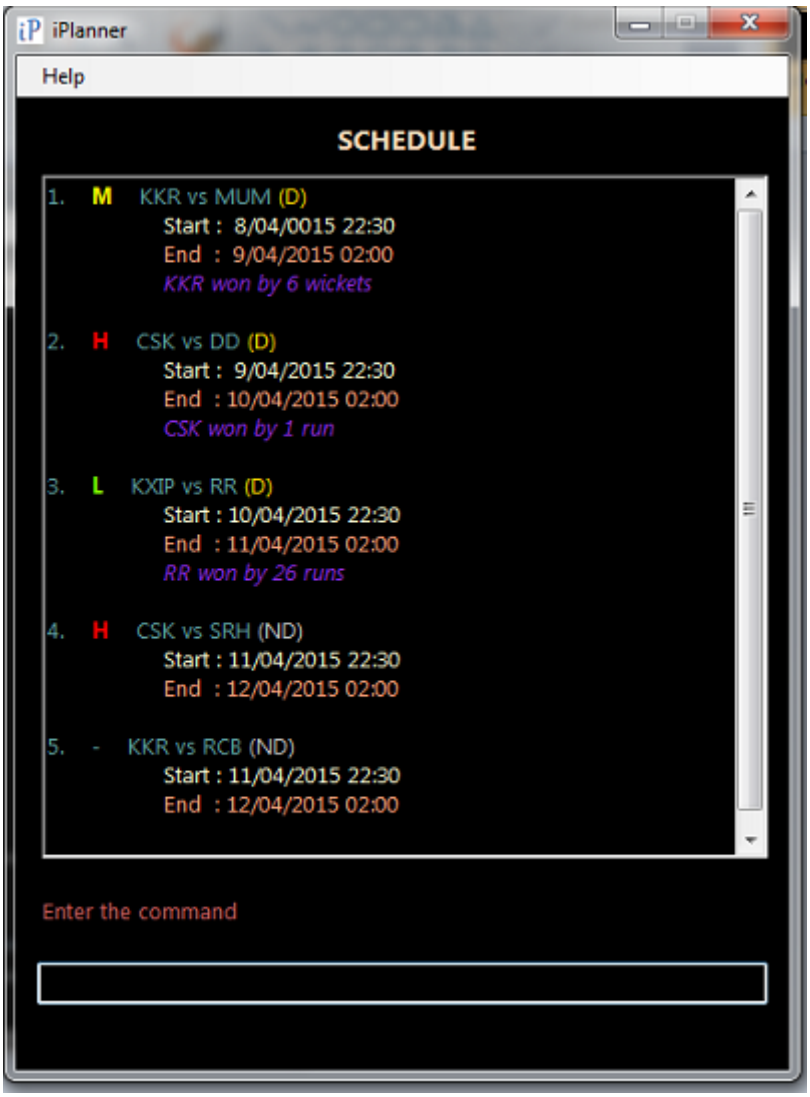
iPlanner



Supervisor: Huang Da **Extra Feature:** Automated Testing

Andy Soh Wei Zhi

Documentation
Code Quality

Lee Joon Fai

Integration
Documentation

Yu Youngbin

Team Lead
Scheduling*
Testing

Ng Chon Beng

Testing
Integration

Shri Kishen Rajendran

Code Quality
*Scheduling**

* Role includes scheduling, tracking, and ensuring adherence of deliverables to deadlines.

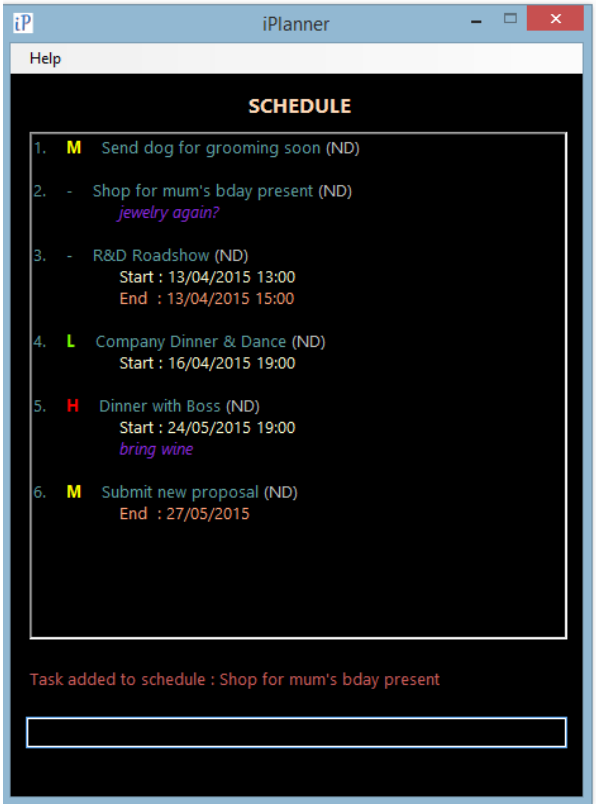
iPlanner User Guide

Welcome to iPlanner – planning and scheduling at your fingertips!

iPlanner is a light and user-friendly task managing software, targeted at the busy individual who requires an efficient way of managing tasks on a desktop or a laptop PC, even when offline. This quick-start guide will bring you through the features provided by iPlanner, everything at your fingertips!

The User Interface

iPlanner's user interface is geared towards providing users with fast, understandable, and easily-understood feedback. Control of iPlanner is achieved entirely through the keyboard, and the entering of commands to iPlanner is easy to pick up.



The image above shows how items are displayed in iPlanner. An item has up to six fields for information storage, namely Name, Description, Start Time, End Time, Priority, and Completion. The different fields have also been colour-coded to ensure easy recognizing of elements that are important (e.g. High priority items).

How Commands Work

iPlanner supports entering multiple relevant commands in the same line. For example, entering:

add Boss's birthday party -start 15/03/2015 1300 -end 15/03/2015 1430 -desc 24 Kent Ridge Park, buy present -priority H

adds a new event, titled "Boss's birthday party" from 15 March, 1pm to 15 March, 2:30pm, with additional information specified by the user – including location (Kent Ridge Park) and a task to do (buy a present). Additionally, the event is marked as high priority.

Main Commands

Main input commands:

- add - Add a new item into iPlanner
- del/delete - Remove an item from iPlanner
- edit - Edit the field(s) of an item in iPlanner
- done/undone - Marks an item as done/undone

Main display commands:

- search - Search for a keyword/phrase within the items in iPlanner
- sort - Sort the items on display according to date/name/priority
- view - View only items meeting the condition (high priority, range of dates, all, etc.)

Other main commands:

- clear - Removes all currently-displayed items from iPlanner
- undo - Undo the last undo-able action (add/delete/edit/clear)
- exit - Exits iPlanner

Sub-Commands

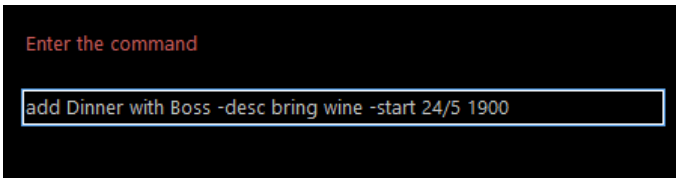
Input sub-commands: The following commands can be used after the '*add*' or '*edit*' commands to configure the fields of the item being added/edited.

- -description/-desc - Configure item description
- -date - Configure item start (and end) date/time
- -start - Configure item start date/time
- -end/-due - Configure item end date/time
- -priority - Configure item priority
- -remove/-rmv - Remove a field from the item (only for '*edit*')

Furthermore, in the case where you accidentally type in more than one space between commands and texts, the programme will still be able to recognise the input.

Add New Item

Create a new item by entering *add <name of item> followed by the sub-command(s)* in the user interface.



Edit Item

Edit an existing item by entering *edit <display index of task to be edited> followed by sub-command(s) of item entity to be edited* in the user interface.

Delete Item

Delete an existing task by entering *delete <display index of task to be deleted>* or *del <display index of task to be deleted>* in the user interface.

Assign Timings – *start/date* and *end/due*

In iPlanner, items are categorized into Events (with specific start and end dates/times), Deadlines (with specific due dates/times) and Tasks (without specific dates/times attached). When adding a new item or editing an existing one,

1. Specify start time by entering
    a. *-start <dd/mm/yy, hhmm> or -start < hhmm, dd/mm/yy>*
    b. date *<dd/mm/yy, hhmm> or -date < hhmm, dd/mm/yy>*
2. Specify end time by entering
    a. *-end <dd/mm/yy, hhmm> or -end < hhmm, dd/mm/yy>*
    b. *-due <dd/mm/yy, hhmm> or -due < hhmm, dd/mm/yy>*
3. Specify start and end time by entering
    a. -date *<permitted date time> to <permitted date time>*
    b. -date *<permitted date time> - <permitted date time>*

Flexible date and time formats

iPlanner also supports the input of date and time in flexible formats. A list of supported formats and examples are listed below.

Date & Time variations

- 27/12/2015, 17:00 *<date>, <time>*
- 17:00, 27/12/2015 *<time>, <date>*
- 27/12/2015 *<date>* only
- 17:00 *<time>* only, date is set as current date

Date variations

- 27/12/2015 *<dd/mm/yyyy>*
- 27 December 2015 *<dd MonthName yyyy>*
- 27 Dec 2015 *<dd ShortMonthName yyyy>*
- 27/12 *<dd/mm>*, year is set as current year
- 27 December *<dd MonthName>*, year is set as current year
- 27 Dec *<dd ShortMonthName>*, year is set as current year

Time variations

- 17:00 or 1700hr/am/pm*<hh:mm>* or *<hhmm>hr/am/pm* (24 hour format)
- 5:00pm or 500pm *<hh:mm>am/pm* or *<hhmm>am/pm* (12 or 24 hour format)
- 5pm *<hh>am/pm* (12 or 24 hour format)

Adding Descriptions – *description/desc*

When adding a new item or editing an existing one, additional details or descriptions of the items can also be added by entering *-description <description>* or *-desc <description>* in the user interface after a main input command (add or edit). This may include the location of the event, *e.g. Singapore Flyer*, and other relevant details, *e.g. bring a rose, wear a bowtie*. Note that *<description>* will take on the user's input until the end of the command line or the beginning of a different sub-command.

Assign Priority – *priority/p*

When adding a new item or editing an existing one, assign priority to an item by entering *-priority <priority level>* or *-p <priority level>* in the user interface after a main input command (add or edit). Note that priority can only take on the values of low, medium, or high, and <*priority level*> in the user interface can be entered as *low*, *medium* or *high* or *L*, *M*, or *H*.

Remove Item Field – *remove/rmv*

While editing an item, you can choose to erase the information in the description, start time, end time, and priority fields.

- Remove Description – Remove only the item description by entering *-remove description/desc* or *-rmv description/desc* in the user interface.
- Remove Start Time – Remove only the item start time by entering *-remove start* or *-rmv start* in the user interface.
- Remove End Time – Remove only the item end time by entering *-remove end* or *-rmv end* in the user interface.
- Remove Start and End Times – Remove both the start and end times by entering *-remove date* or *-rmv date* in the user interface.

Mark Done/Undone – *done* and *undone*

You can mark an item as "Done" by entering *done <task index of task completed>*. User can mark an item as "Not Done" by entering *undone < task index of task completed >*.

Undo Previous Action – *undo*

Undo the previous undo-able action by entering *undo* into the command line interface. Only add, delete, edit and clear commands are undo-able.

Search – *search*

Search for an existing item by entering *search <keywords>* in the user interface. Note that *<keywords>* will take on the user's input until the end of the word, and only results that match the entire string, will be returned.

Multiple strings may be searched by including a '+' in between two words to be searched. For example, *search <keyword1>+<keyword2>+<keyword3>*

Sort Items – *sort*

The items may be sorted by name, date, priority, completion and update order.

- Sort by Date – The default arrangement of events/deadlines is chronologically, with the earliest ones displayed first, i.e. an event on 4 Jan 2015 will appear before an event on 14 Feb 2015. Tasks (without timestamps) will appear at the beginning. Revert to this arrangement by entering *sort date* in the user interface.
- Sort by Name – Sort items by their names by entering *sort name* in the user interface.
- Sort by Last Update – Sort items in the sequence that they have been updated by entering *sort update* in the user interface.
- Sort by Priority – Sort items in order of priority by entering *sort priority* (or *sort p*) in the user interface. High-priority items will appear first, followed by medium-priority ones, followed by low-priority ones. Items without priority indicators will appear last.
- Sort by Completion – Sort items in order of completion by entering *sort done* in the user interface. Items that are not completed are displayed first, followed by items that are completed.

View Items – *view*

The items may be filtered according to date range, priority, and completion.

- View All – All items in the Schedule can be viewed by entering *view all* in the user interface.
- View Range of Dates – Items between two dates can be viewed by entering *view <date1> <date2>* in the user interface. Events starting, ending, or spanning between that time period and deadlines on that day will be returned. If only one date is keyed in, then all events and deadlines that overlap with that specific time are returned.
- View by Priority – View only high-priority, medium-priority, or low-priority items by entering *view High* (or *view H*), *view Medium* (or *view M*), *view Low* (or *view L*) respectively into the command line interface.
- View by Completion – View only items that have been done or that have yet to be done by entering *view done* or *view undone* respectively into the command line interface.

Storage – *save*

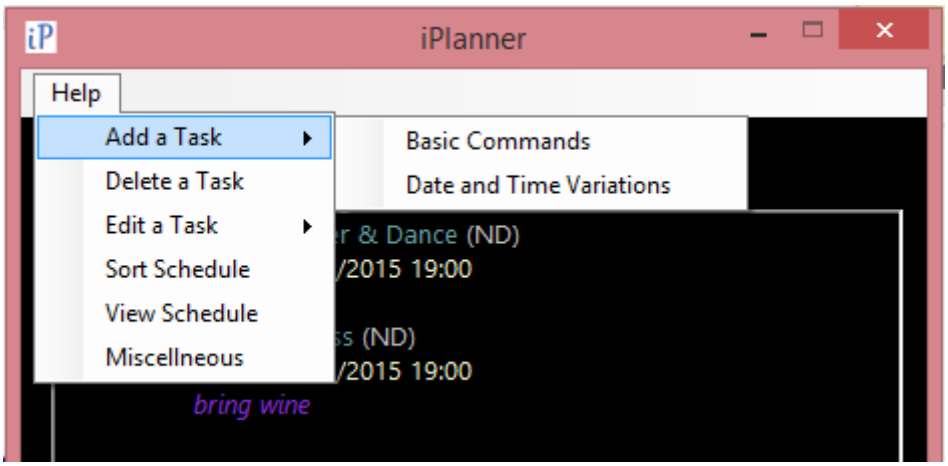
When the program is executed for the first time, the schedule is stored, by default, in the same folder as the executable file.

- Change Save Directory – The save directory can be changed by entering *save <new directory to be saved>* in the user interface.

Help Menu

A comprehensive help menu detailing all the commands supported by iPlanner and variations of the input format can be called from within iPlanner.

- View Help Menu – View the list of commands supported, their specifications, and syntax by pressing *Alt,* followed by *Enter* in the user interface. The help menu can be scrolled through using the arrow keys and Enter button.



Exit Application

Exit iPlanner by entering *exit* in the user interface.

iPlanner Developer Guide

1 Introduction

Welcome to the developer guide written for new developers who may be interested in contributing to the development of iPlanner. This guide covers the conventions, recommended tools, as well as the architectural design of the product; it should provide you with the required knowledge to start contributing to iPlanner.

iPlanner aims to be a lightweight, yet functional and efficient to-do list manager software, and the original developers would appreciate it if you keep the above characteristics in mind when expanding this project.

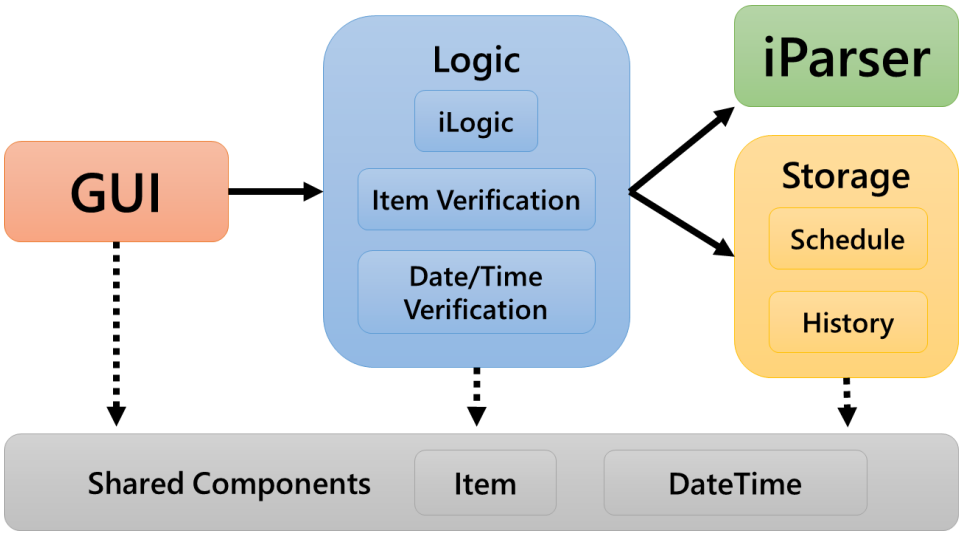
2 System Requirements

This project is developed with C++ 11, with the UI using the .NET framework, and the preferred Integrated Development Environment (IDE) is Visual Studio 2013. However, you may use any IDE of your choice, especially if you are already very much more familiar with another IDE. If you do choose your own IDE, please make sure that it supports working with Visual Studio projects seamlessly. In particular, please ensure that you leave no traces of using a different IDE and refrain from committing IDE-specific files into the repository.

The version control system used is Git, and you may pull from our GitHub repository, located at https://github.com/cs2103jan2015-f10-2c/main, with any Git client. Our preferred Git client is SourceTree developed by Atlassian, but other clients you are familiar with would work as well. Once done, open iPlanner.sln to begin coding.

3 Architecture

Architecture-wise, iPlanner is a relatively simple software, consisting of four main components, the Graphic User Interface (GUI), Logic, Parser, and Storage. GUI shares an association with Logic, while Logic shares associations with Parser and Storage. Additionally, the GUI, Logic and Storage components also share a dependency on the Item and DateTime classes. The relationship between the different components is illustrated in the architecture diagram below:



4 Component Introduction & Basic Workflow

GUI: The GUI is the “entry” and “exit” point of the entire program, also known as a “main”. When the user inputs a command, the GUI starts a propagation of actions deeper into Logic, and

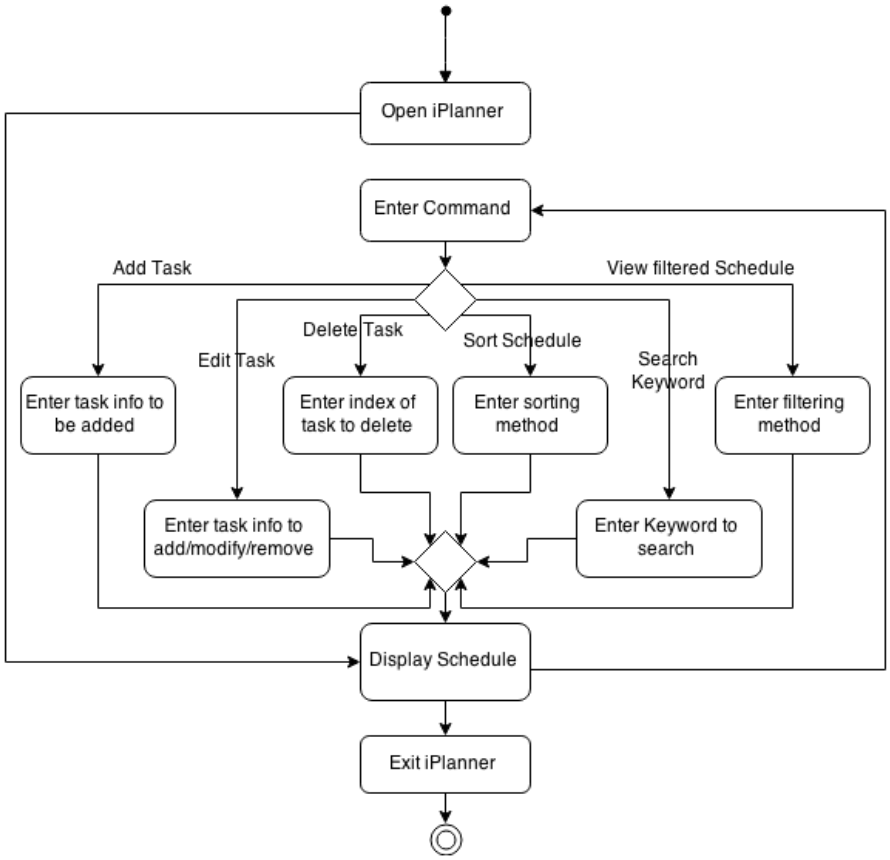
subsequently, Parser and Storage. GUI then refreshes the display for the user, and provides him/her with a feedback message and an updated schedule.

Logic: The Logic component takes the raw user input that it receives from GUI, and passes it to Parser for conversion into a form that is more easily interpreted by Logic. Running through the list of instructions received from Parser, Logic checks for errors in the user input and executes the required instructions, passing information to Storage where required. It finally returns GUI with a feedback message and an updated schedule for display to the user.

Parser: Upon receiving the raw user input from Logic, Parser interprets it and constructs a list of instructions for Logic to execute.

Storage: Storage is responsible for the record-keeping of iPlanner. This includes keeping track of all the items in the iPlanner schedule, organising the items in the schedule that has to be displayed to the user, and handling the undo functionality of iPlanner.

4.1 Activity Diagram

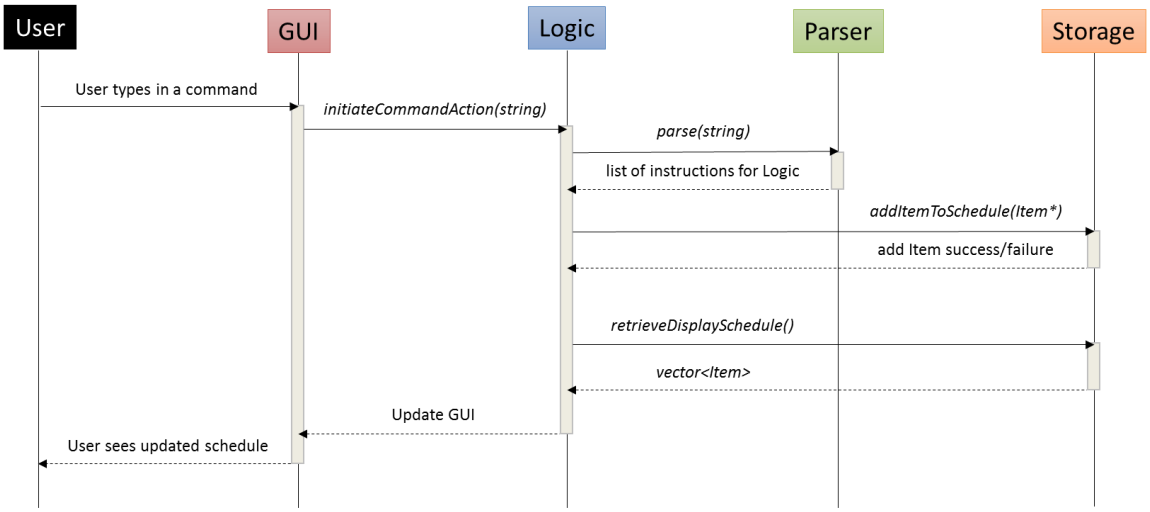


The activity diagram of the user above outlines a few of the possible use cases for iPlanner and is not exhaustive. The activity diagram begins with the user opening the application and the existing Schedule is displayed to the user. The program can be exited or a command can be entered to perform an action (add/delete/edit/sort/search/filter) can be performed. When adding a new task, the details of the task such as taskName, startTime, endTime, description, etc are entered and they are processed and stored in the schedule, and the modified schedule is displayed.

After performing an action, another command can be entered, the refreshed Schedule is displayed and the program can be exited or another command can be entered, etc. Similarly the process continues for each command entered, until the application is exited.

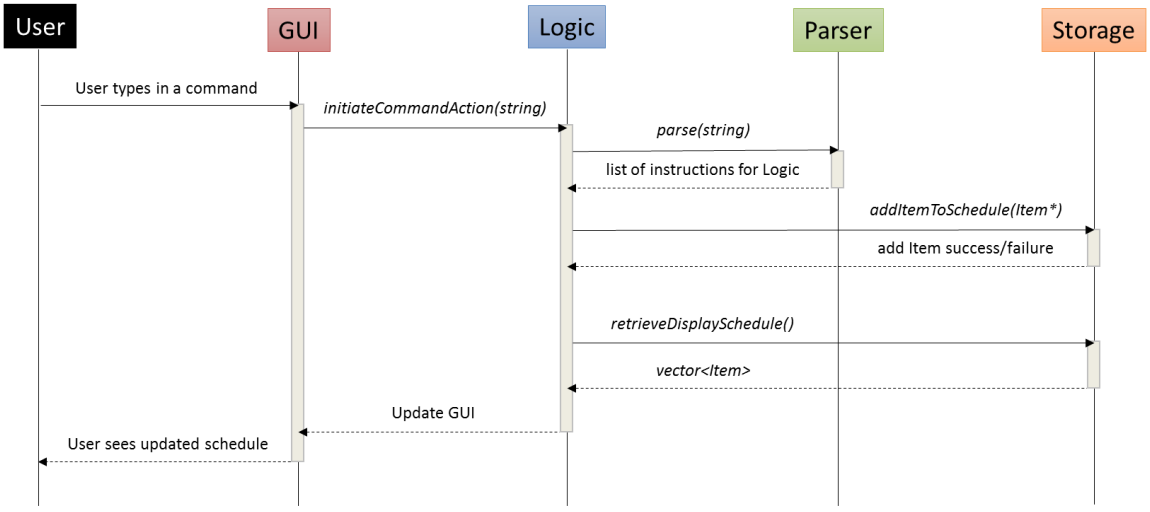
4.2 Sequence Diagrams

The activity diagram in the previous section should have given you a rough idea of how the internal components of iPlanner interact. The following sequence diagrams illustrating some general use cases aim to give you a clearer mental picture of how the various components invoke and propagate API calls to other components in specific use cases.



#1. Sequence Diagram for adding an item:

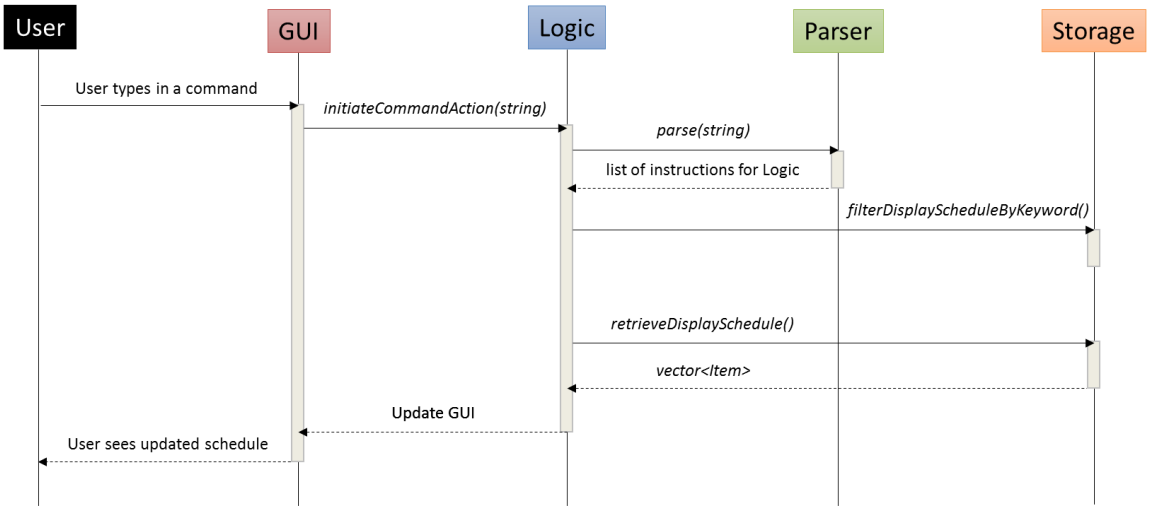
1. When the user types in a command to add a new item, GUI will immediately pass the raw user input as a string to Logic.
2. Logic passes the string on to Parser, who converts the user input into a list of instructions for Logic to execute.
3. Running through the list of instructions, Logic creates a new Item with all the fields filled in as specified by the user. Upon reaching the end of the list, Logic does a final check to verify that the fields within the Item are “valid”, before passing the newly-created Item to Storage for addition into the schedule.
4. After addition of the Item, Logic calls Storage again, this time to retrieve a copy of the updated schedule to display to the user.
5. Finally, Logic passes the updated schedule on to GUI, which displays the schedule in an accessible and easily-understood format to the user.



#2. Sequence Diagram for editing an item:

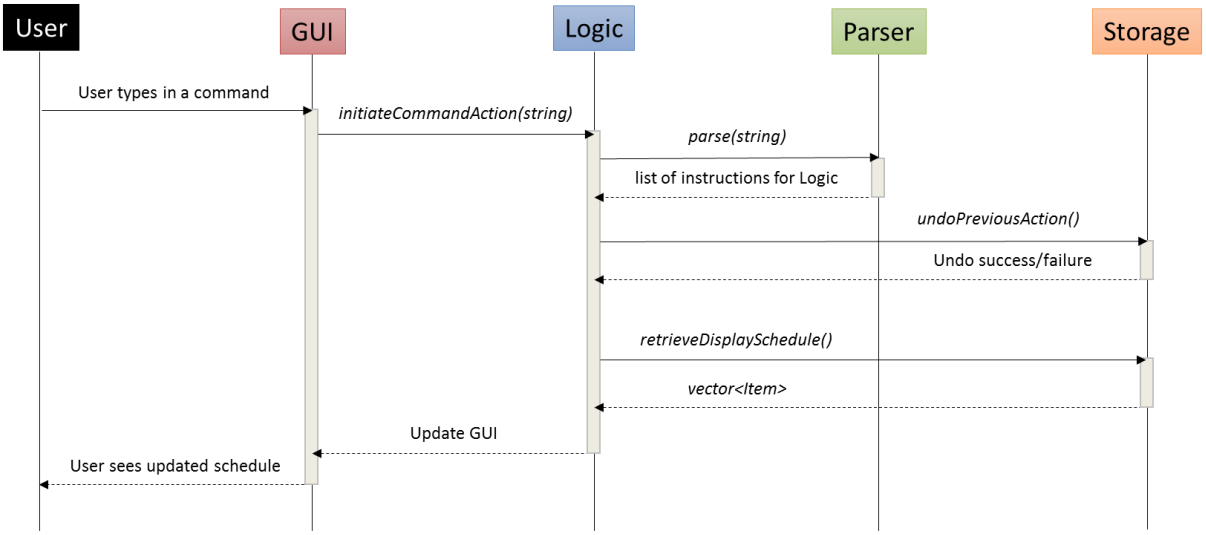
1. Same as in #1.

2. Same as in #1.
3. Logic calls Storage to retrieve the Item that the user wants to edit, and performs the necessary changes to the Item according to the list of instructions returned by Parser. Upon reaching the end of the list, Logic does a final check to verify that the fields within the Item are "valid", before passing the edited Item to Storage to replace the "old" Item in the schedule.
4. After editing the Item, Logic calls Storage again, this time to retrieve a copy of the updated schedule to display to the user.
5. Same as in #1.



#3. Sequence Diagram for searching via a keyword:

1. Same as in #1.
2. Same as in #1.
3. Logic instructs Storage to filter the schedule by keyword, preparing to show the user only the Items containing the search term. If multiple search terms are provided by the user, Logic will repeatedly instruct Storage to filter the schedule.
4. After the filter(s), Logic calls Storage again to retrieve a copy of the updated schedule to display to the user.
5. Same as in #1.



#4. Sequence Diagram for undoing an action:

1. Same as in #1.
2. Same as in #1.
3. Logic instructs Storage to undo the last undo-able action.
4. After the undo, Logic calls Storage again to retrieve a copy of the updated schedule to display to the user.
5. Same as in #1.

5 iPlanner Components

5.1 GUI

The GUI component is used by the User to view the Schedule when a command is executed. It also contains a drop-down menu containing the supported commands, for the user's reference.

5.1.1 Package Overview

iPlannerUI contains methods to primarily display the schedule, on starting the application and on entering a command. A command entered by the user, is passed to Logic to process. After processing, Logic returns a Schedule, which is finally displayed by the GUI. Methods are also present to display the Help Menu, containing a list of supported commands, when they are clicked.

5.1.2 Policies

Façade Pattern: This pattern is used to restrict GUI interaction with only Logic and the shared component Item.

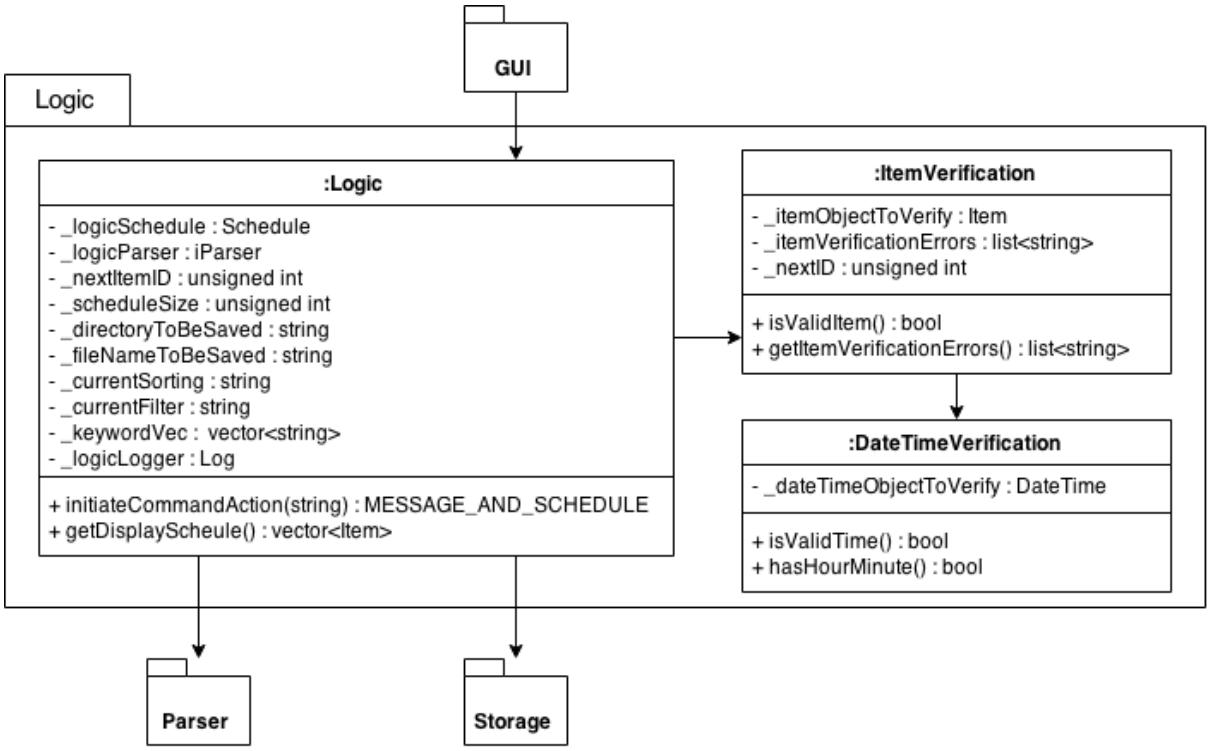
MVC Pattern: This pattern is used by GUI to process a command entered by the user and display the Schedule. Schedule, returned by the user is temporarily stored in a vector of Items before it is displayed. Events handled by GUI include KeyDown (when the Enter key is pressed in the input box) and Click (in the Help Menu).

5.1.3 Extensibility

Display by Grid view:

To display by a Grid view, you would need to create a DataGridView in the place of the RichTextBox. Columns for Display index number, task Priority and Task may be used. Appropriate commands would be required to display the task entities of a task in their respective columns.

5.2 Logic



The Logic component determines and initiates appropriate functions to be called. In particular, it handles the following operations:

- Calling for a parser and interpreting parsed information
- Initiating and executing storage functions such as add, edit, delete etc. depending on the command given by the user
- Retrieving display vector by various sorting and filtering methods, as the user specified.
- Retrieving item from schedule and modifying item parts
- Verifying the validity of each item components
- Reading data from save file and writing data on to the file
- Passing display message and display vector to GUI

5.2.1 Package Overview

Logic.Logic:	Interacts with parser, storage and GUI and controls the change in the storage or display method specified by the user.
Logic.ItemVerification:	Provides a validity check of an item that is created or edited.
Logic.DateTimeVerification:	Provides a validity check of the DateTime component of an item that is created or edited.

5.2.2 Policies

Façade Pattern:	This pattern is used to restrict access of other components to the internal details in the Storage component.
Command Pattern:	This is used in Logic Class. Logic component only invokes appropriate functions for the command given and receives a message of success or failure of the command.

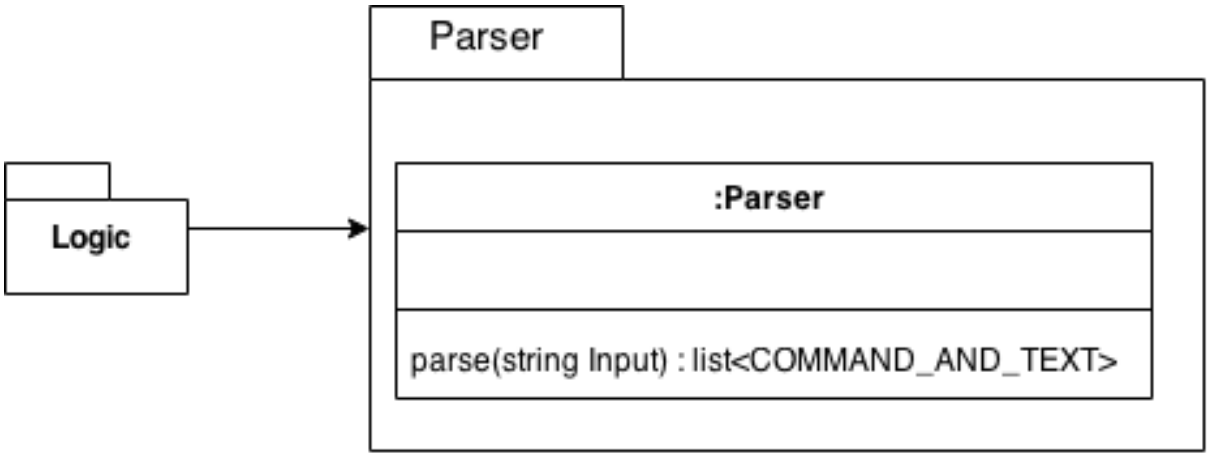
5.2.3 Extensibility

Allowing more flexible commands given by the user:

To allow variable commands or modifiers such as “tomorrow” or “monday”, you will need to include a function that interprets those flexi commands and convert it into our appropriate DateTime object.

Implementing detection method over the save text file corruption:
To detect the corruption of the save file, you need to introduce isValidNumberofLines of the saved data. Moreover, you will need to go on and check line by line whether the data in each line represents valid item parts.

5.3 Parser



The Parser component takes in a user input string from Logic and it returns a list of commands and text back to Logic. Primarily, it tries to interpret the user input string as accurately as possible and return the commands and text for Logic to execute out the user’s desired actions.

5.3.1 Package Overview

Parser: Provides only one public function for Logic component to call in order to interpret user’s input.

5.3.2 Policies

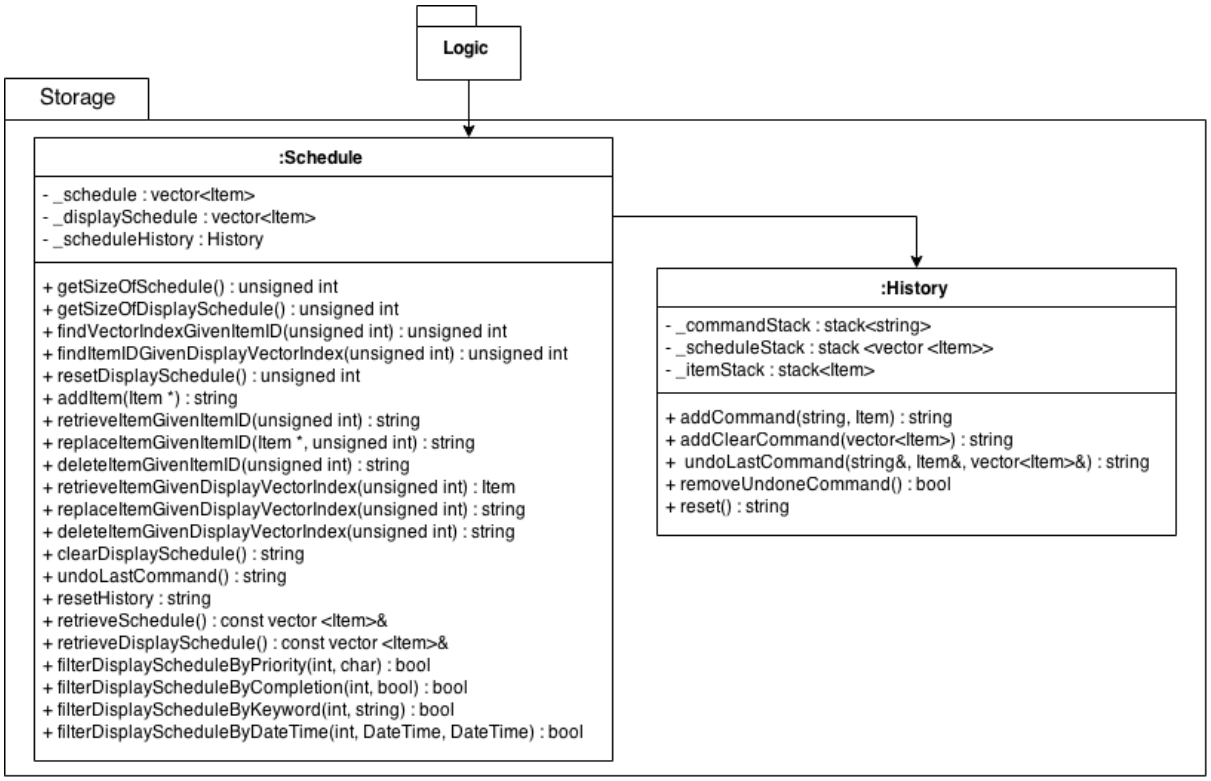
Façade Pattern: This pattern is used to restrict access of other components to the internal details in the Parser component.

Singleton Pattern: This is used to ensure that there is only exactly one instance of parser class.

5.3.3 Extensibility

Implementing better Natural Language Processing:
The parser requires the user to input a rather rigid input as keywords coupled along with a hyphen is required for the commands and modifiers. Better algorithms will be required to allow the user to enter less rigid inputs.

5.4 Storage



The Storage component supports Create, Update, and Delete operations, as well as a search feature, a sort feature and an undo feature. In particular, it handles the following operations:

- Storing a temporary copy of each Item present in the storage files
- Creating, updating, and deleting tasks
- Searching for items with a user-specified keyword
- Filtering items by a given attribute – priority, completion status, date
- Sorting items by a given attribute – priority, completion status, date, name, last update
- Undoing commands executed during the run

5.4.1 Package Overview

Storage.Schedule: Provides all the storage methods that will be called by the Logic components, as well as the relevant tests and supporting functions.

Storage.History: Provides all the methods and data members for the implementation of the undo feature.

5.4.2 Policies

Façade Pattern: This pattern is used to restrict access of other components to the internal details in the Storage component.

Singleton Pattern: This is used in both Schedule and History classes, and it ensures that there is only exactly one instance of each class.

5.4.3 Extensibility

Implementing new methods to search for/filter Items:

To change the manner in which Items are searched, you need to modify existing or introduce additional filterByAttribute and isMatchingAttribute methods. Moreover, you will need to modify existing or introduce additional public retrieveDisplayScheduleFilteredByAttribute methods.

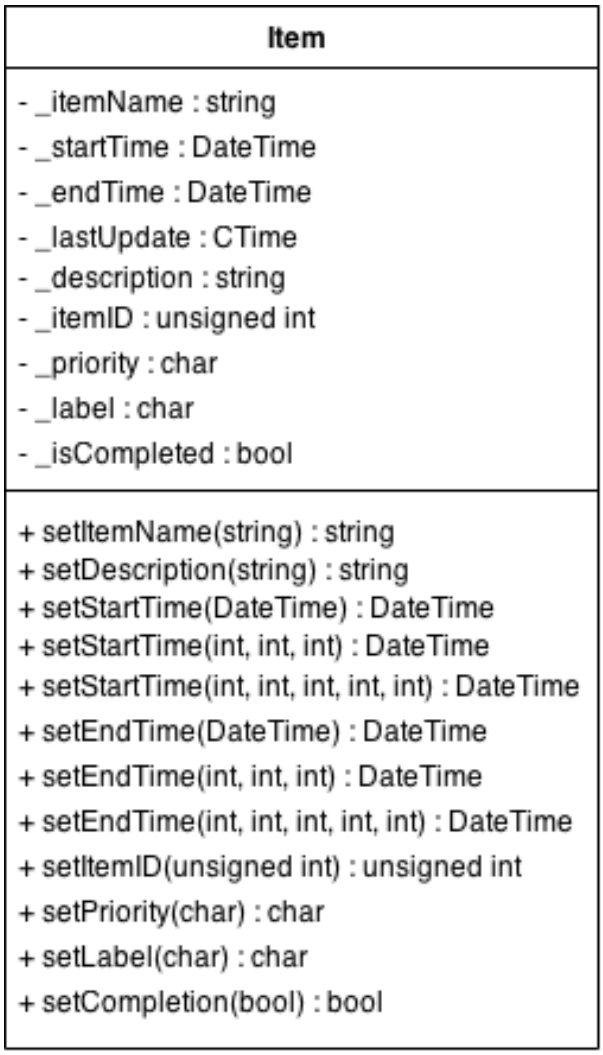
Implementing new methods to sort Item:
To implement new sorting methods, you need to modify existing or introduce is…Than methods. Moreover, you would need to modify existing or introduce additional public retrieveDisplayScheduleByAttribute methods.

Increasing coverage of undo functionality:
To allow the undo feature to cover more commands, you need to introduce new command members into the addCommand and undoLastCommand methods in the History class. Alternatively, new stacks and their corresponding methods can be implemented where appropriate.

5.5 Shared Component

The Shared component stores all the common classes that are shared among the whole project, namely Item, DateTime, and Log.

5.5.1 Item Class



The Item class is the common Item object that is used by Logic and Storage to store all the attribute-value pairs of all items. In addition, the GUI uses the Item class to interpret the schedule passed from Logic to GUI. The Item object allows for this by providing setter and getter functions for each attribute, with exception to lastUpdateTime which only has a getter. In this instance, the setter is private and is done automatically by obtaining the system time from the user's terminal. Additionally, Item class provides information pertaining to the Item object and its internal attributes in the form of a string.

There are three kinds of Items: firstly, events have both valid starting and ending times; secondly, deadlines have valid end times but no valid start times; and lastly, tasks have neither starting nor ending times. An Item can be converted between the abovementioned types simply by setting or removing the relevant times.

In order to implement new attributes for the item, e.g. user-defined labels, private members have to be introduced together with the relevant setters and getters. Additionally, if the attribute can be translated into a string for display to the user or for error-checking in other components, you are encouraged to do so. The relevant Boolean variables to indicate if the Item has a valid attribute should also be implemented.

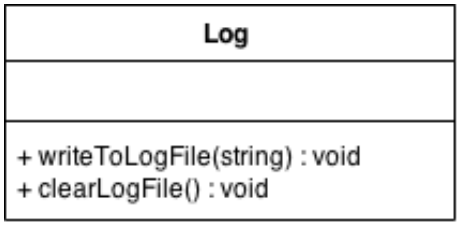
In order to implement the recurring functionality, additional private members, such as a Boolean variable to indicate if an Item is recurring, a variable to indicate the frequency of recurrence, and a DateTime to indicate when the recurrences end, have to be introduced. Corresponding setters and getters have to be introduced accordingly.

5.5.2 DateTime Class

The DateTime class is the common DateTime object that is used by Logic and Storage to store all the attribute-value pairs of all dates and times. The DateTime object allows for this by providing setter and getter functions for each attribute. Additionally, the DateTime class provides information pertaining to the DateTime object and its internal attributes in the form of a string.

As DateTime objects are able to be sorted chronologically, comparators to allow the comparison of two or more DateTime objects have been implemented. Also, in order to introduce added layers of defence, DateTime checks for the validity of the value before allowing the attribute to be set; for example, before setting the minute, it checks if the value is between 0 and 60.

5.5.3 Log Class



The Log class is the common Log object that is used by all the other components to log both successful completion of certain operations and errors when they occur. The Log object allows for this by writing logs directly to a designated text file for logging as logs are generated by the other components.

In order to balance trade-offs between the file size of the logging text file and the coverage of the logs, the log text file is overwritten every time the program is rebooted. Should you want to review possible errors that might have occurred during a given run, the developer has to take care not to

reboot the program, or to save the log file under in a different destination between each time the program is run.

The log class can be further extended to save in a different text file every time the program is booted up should you feel that it is necessary to do so (possibly useful if you are implementing additional complex features or functionality). However, it is important to carefully manage the extent of logging and the corresponding file sizes.

6 Testing

As you code, it would be ideal if you write unit tests for each class to test the correctness of your codes. This continuous testing process is made streamline and efficient using a testing framework; in this project, the unit tests have been implemented with the Native Unit Testing Project framework available in Microsoft Visual Studio Ultimate 2013. More information on the functionality and support provided by Microsoft Visual Studio Ultimate 2013 can be found in the online Microsoft guides as well as through the Help section of the IDE.

Although the development of iPlanner is not strictly test-driven, you should try to write tests for code that you intend to merge into the master branch to the furthest extent possible. This will minimize time spent on debugging faulty code, both for yourself and the other developers working on the project. It is important to note that since you will be the one most familiar with your code, it is ideal that you are not dependent on other developers to write the unit tests for you.

For iPlanner, the tests for each component have been stored separately, and your unit tests should be created within the component you are modifying. Please ensure that all the tests pertaining to the codes you have written have passed prior to merging into the master branch, so as to minimize time wasted in the debugging of faulty code (both for you and other developers). It is important to catch regressions or bugs as soon as possible, before the adverse impacts of such bugs and regressions are amplified over time.

As of production release, iPlanner v0.5, approximately 83% of the code blocks have been covered. In general, it is desirable to have as much test coverage as possible, as it is an indicator of the robustness of the software. Also, there is much value in testing boundary cases to ensure that the code works in the largest range of values that it is intended to function in. Additionally, it is to be noted that unit tests of private functions have been muted after they have been passed. As far as possible, unit tests for private functions should be implemented, especially if they are highly complex, for instance, those in Parser component. Where possible and implementable, it is desirable if friend classes/functions are utilized to test private functions.

In addition, as part of good defensive coding practice, it is ideal if appropriate assertions and exception handling is implemented, again to ensure the high robustness of the software.

Appendix A: User Stories. As a user, …

[Likely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
addTask	add a task by name of task only	can record tasks that I want to do a certain day or some day
editTask	edit a task by changing details of an already created task	make changes to the details of the task should there be any
removeTask	remove a task that had been added earlier	delete a task that is no longer required to be done
rescheduleTask	reschedule a task by changing the deadline or time of an already created task	make changes should there be any
undoLastAction	undo a single action which was previously done (e.g. add/ remove/ reschedule etc)	do not have to delete and remake the whole schedule one more time
specifyDeadline specifyTimedTask	specify a deadline or time for a task should there be any	can keep track of the important deadline or time
clearTasks	clear some or all tasks	do not have to clear them one by one
addToArchive	archive tasks which are done or marked as complete	can view them in the future should there be a need to
markAsComplete	mark or unmark a task as complete	know what is done / not done and can prioritise my tasks
chooseSaveFolder	specify where he wants the data .txt file to be saved	can access it on file-sharing software like Dropbox
viewTasks	view upcoming tasks	can have a clearer understanding of my tasks at hand
searchTask	search for task(s) using keywords	can filter out the task(s) that I am searching for
labelItem	can label tasks into categories	can have an understand which category the tasks belong to
addSubTask	can add a daughter task to a parent task	know what tasks are to be done which belongs to the same task
addDescription	can add description to a task	do not forget what the task is about should the task name be ambiguous
assignPriority	assign priority to a task	know what are the more urgent tasks at hand
recurringTasks	set recurring tasks	do not have to add them over and over again

[Unlikely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
executeHotKey	can make use of hot keys (e.g. ctrl + z)	execute any actions faster
notifyClash	should be informed of clashes in timings of scheduled tasks	can plan what to do with the clashes
changeView	can filter my tasks in views which i want to see (e.g. category, date, priority etc)	can have a clearer understanding of my tasks at hand

setReminder	set reminder to my task	will be informed of the task's at the time of reminder which was set
viewFreeTime	can view my available timeslots on certain day(s)	can make plans on free timings should there be a need
viewHelp	can view help document	have an idea how to use the application

Appendix B: Non-Functional Requirements

Desktop

iPlanner will work on a desktop or laptop without network or Internet connection. It is not a mobile application or cloud-based application.

Windows

iPlanner will work on Windows 7 or Windows Vista OS, and on both 32 bit and 64 bit personal computers.

Standalone

iPlanner works as a standalone software; it is not a plug-in to another software. However, extensions might be available in later versions to maintain compatibility and exporting of data with other programs or other existing software.

Non-Installation

iPlanner works without requiring an installer.

Command Line Interface

Command Line Interface is the primary mode of input, and a basic graphic user interface is used to display the output. There is no need to use the mouse for any point-and-click functions.

Data Storage

Data storage is done via text files created when the program is run. Text files will be human-editable, and users are able to manipulate the task list by editing the data file.

Object-Oriented Programming

Most of the iPlanner will follow the object-oriented programming paradigm.

Appendix C: Product Survey

Product: **Remember The Milk**
Documented by: **Lee Joon Fai**

Strengths:

- Easy syncing between smartphone app and desktop interface
- Adding of tasks and assigning details done very conveniently within one command line
- Sort and filter tasks according to labels and priority
- Maintains record of completed tasks
- Option for app to send automated email reminders
- Very flexible and customisable search system
- Geo-tagging of tasks is available
- User can add notes to tasks along the way
- Allows flexible postponement of tasks
- Easy undo-ing of accidental actions
- Can share to-do list with other users of the app

Weaknesses:

- No calendar view, hence difficult for user to visualize longer-term tasks
- No option for adding floating tasks i.e. all tasks must have a deadline
- Smartphone app interface is very different from desktop interface. The switch between devices is not intuitive enough
- The software has remained pretty stagnant over the years, updates are not frequent

Product: **wunderlist**
Documented by: **Yu Young Bin**

Strengths:

- It is easy to post on facebook immediately since it is connected to facebook account.
- It has simple and easy user interface.
- User can customize the background picture.
- User gets notification on important events.
- User can categorise the events and to-do lists.
- User can search for a certain event.
- User can even post pictures related to the event.
- User can sort the events in accordance with the alphabetical order or deadlines etc.
- It is easy to create, delete, and move any events.
- User can print out the to-do list directly from the application
- User can e-mail the to-do list directly from the application

Weaknesses:

- It does not seem quite necessary to create an account and login to use this application.
- There is no user guide for the new users to get familiarized with the product.
- There is no calendar view. It is difficult to visualize the sequence of the upcoming events.
- When a user accidentally presses the tick-box on the to-do list, the event just disappears.

Product: **Microsoft Outlook Calendar**
Documented by: **Ng Chon Beng**

Strengths:

- Can add item into calendar view
- Able to reschedule and delete item
- Able to view calendar offline

- Easy calendar view to see schedule
- Has a reminder function
- Has recurring function
- Able to categorise items with different colours
- Able to change view (e.g. daily, weekly, monthly)
- Has a search function
- Has weather forecast (provided there is internet connection)
- Able to share calendar with others

Weaknesses:

- Unable to see timing at default view
- Unable to mark as done nor archive action which are done
- Unable to quick launch software using shortcut key
- Unable to add item with no specific due date
- Takes multiple actions to add an item

Product: **Google Calendar**
Documented by: **Shri Kishen Rajendran**

Strengths:

- It has a structured and organised look.
- It can be viewed in the format of a day, week or month.
- It allows differentiating tasks by the usage of different colours for different events.
- It has a search function which can also search for files on the user's Google Drive.
- User is a given a reminder of a task half an hour before a scheduled event.
- User is allowed to sync other calendar(.ics) files.
- User can have a task list where additional sub tasks can also be added under tasks.
- User can specify a deadline for a task.

Weaknesses:

- It does not work without internet connection and a Google account.
- It can be difficult to read if there are too many appointments.
- The interface is more useful for planning events than tasks.
- No feature to indicate if an event is complete or not.

Product: **ColorNote**
Documented by: **Andy Soh Wei Zhi**

Strengths:

- Two types of notes are available – free-style text notes and lists
- Notes are simple and intuitive; they only have three properties – title, content, colour
- Sorting of the notes can be by time modified and colour
- Notes can be viewed as a list (with titles only), list (with details), grid, or a large grid, giving users flexibility to decide what works best for them.
- Notes can be backed-up on cloud storage.
- List-styled notes can be dragged easily to re-order items within the notes
- Sticky notes that can appear on home screen for user convenience
- Notes can be edited directly by double-tapping them, again simple and intuitive.
- Accidental deletion of notes is prevented (delete and permanently delete options).
- Notes can be assigned to days via a calendar.
- Time reminders can be set for each note.
- Notes can be locked with password to ensure privacy.

Weaknesses:

- Only platform supported is Android (no iOS or PC).
- Notes are not file compatible with other programmes, i.e. cannot export notes to a Microsoft Excel compatible file or Google Calendar.
- Pictures, photographs, and drawings are not supported.
- High customizability comes with a trade-off – lack of automaton, e.g. manually marking certain items/lines as completed.