JOT IT DOWN: DEVELOPER GUIDE

June 14

2012

Jot It Down:

Introduction:

Jot it down is a To Do Manager, made for power users of windows who prefer a textual interface and want simple, fast and efficient work flow. Jot It Down makes sure that the new Multimedia user is not disappointed as well through its simple easy to use GUI. Moreover, the whole design tries to ensure sufficient abstraction and separation between the different components to avoid high coupling and maintain a clean and efficient code. The modular design tries to ensure that new functions can be easily introduced, modified and deleted. We hope that as developers, you will help us enhance our product in accordance with user requirements and uphold/enhance its quality.

Downloading the Code:

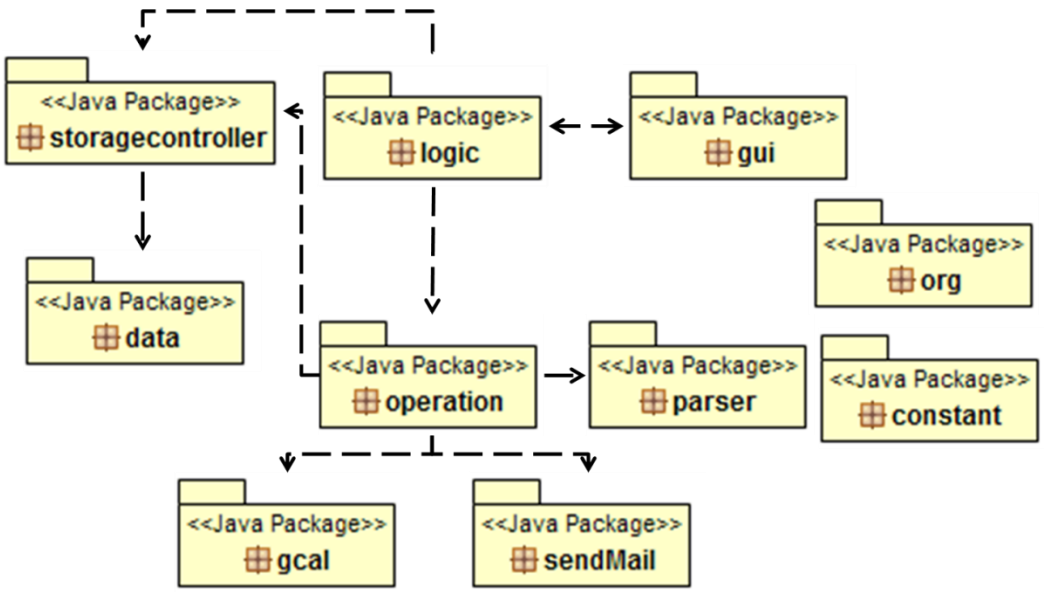
The source code of Jot It Down is available for free at :

https://github.com/shubhendra/IntegratedCS2103.git

Executing the Software:

To execute the program please run the main function in the UIController class. To ensure smooth running please use the latest version of the JAVA SDK (JAVA SDK 7 used in this product).

Getting Started:



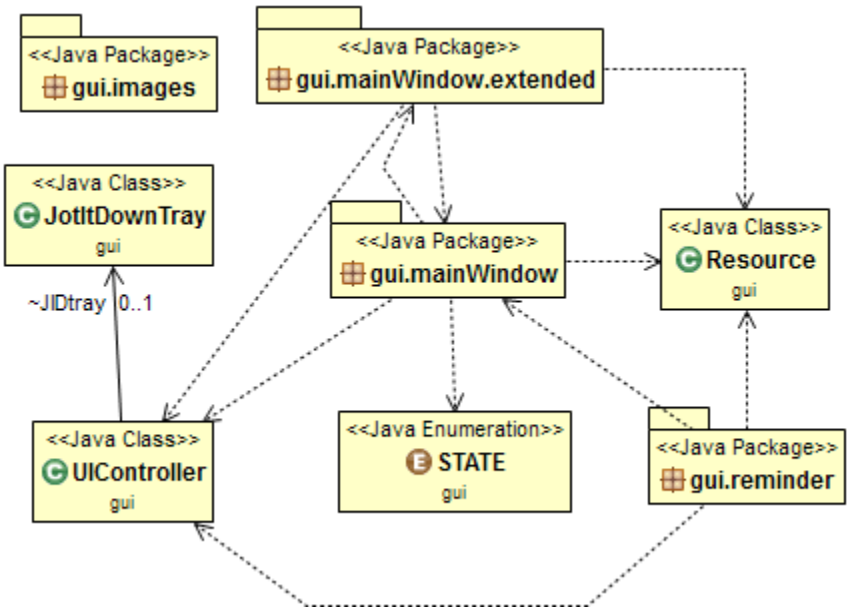
The overall architecture of Jot It Down is simple (refer to figure 1). The Graphical User Interface (GUI) is separated from the back end processing and exclusively deals with displaying the required information to the user. The UIController and the JIDLogic Classes form the interface between the backend and front end and exchange information to carry out the required functionality. These two classes together are the "brain" of this code.

Throughout this guide, we will introduce the various packages and classes that are there as well as highlight key APIs that you might find useful. The entire list of the APIs can be found on:

http://www.google.com

We have also enclosed the API List with this developer's guide.

PACKAGE – GUI



Overview

The graphical user interface (GUI) of Jot It Down! supports both keyboards and mouse. This GUI focuses on power users who prefer command line. Therefore, the design provides short command and hotkey to support the users to be able to use all of the program's features by using keyboard only.

The graphical user interface consists of the main window part, which composes of the command line, the help window, the table for displaying tasks, reminder window, email registration window, and the pop up to show the users a feedback, and tray icon. The interface supports numerous features; such as, incremental search, auto-hide feedback, and so on.

The section below describes the graphical user interface classes.

INDEPENDENT CLASSES

UIController

This class initializes the program. After users start the program, UIController will initialize JIDLogic, which is the main logic part of the program, the main window, the tray, and the reminder. UIController also controls the display of Jot It Down!

- Following are the Key APIs to take note of in this class:
- + refresh() : void
    - This function is called when there is an update on the graphical user interface.
- + sendOperationFeedback(OperationFeedback): void

    - This function is for the operation to send feedback back to the GUI for displaying a correct feedback.
- + showFeedbackDisplay(): void
    - This function prompts the GUI to show feedback
- + promptEmailInput(): void
    - This function is for asking for the email for the first time the program executes.

Action

This class is to support hot key implementation and some operations that do not have interaction with the command line i.e., those that prompts implemental search. The class consists of several inner classes such as ExitAction, UndoAction, DeleteAction (see more in table 1.1), and so on. All of them extend AbstractAction class.

JotItDownTray

This class is to support tray functionality; for example, exit, add new task, show message on the tray and call the Jot It Down! window.

- + showText(String caption, String str): void
  To show the message. The caption will be on the header with larger text size than str. Str will be shown as a description.

Resource

This class consists of constant values; for example, picture. The purpose of this class is to make the developer easy to change images.

STATE

This is an enumeration. The purpose of this enumeration is to help controlling the GUI since each command has different output. For example, search function and some functions needs a drop down box while others do not. Moreover, the feedback from each command is different.

- + getEndedString(Boolean isOneTask): String - This function returns the ending statement of the feedback which is different for each command and for the different number of tasks. For example, if the current state is “add” and “one” task is added, it will return “was added.”
- + getFeedbackText(): String - This function provides valid feedback that do not require task detail to be shown

SUB PACKAGE – images

This sub package stores all the images.

SUB PACKAGE – mainWindow

This sub package stores the main window for Jot It Down’s components. For example, MainJFrame, ExpandComponent, and so on.

MainJFrame

- creates the main window for the GUI
- controls the command line interaction with users.

Binding

- makes hotkeys for a particular JFrame by editing InputMap and OutputMap**.**

AutoCompletion

- manages the command line and the drop down menu.

SUB PACKAGE - mainWindow.extended

This contains the extension of the main window which do not always show.

ExpandComponent

This initialize the components that are appear when the main Window is expanded. This consists of JScrollpane for providing scroll bar when there are many tasks and a JTable for displaying tasks.

TaskTable

This edits table component in the ExpandComponent class and makes it compatible for displaying task details and images. This also controls how the user interact with this table **.**

TopPopUp

This is a JFrame sub class. This JFrame will appear on the top of the main window to show messages.

HelpFrame

This is a JFrame sub class for showing some critical commands This frame will appear on the right of the main Window.

MailDialog

This is a JDialog sub class. This mail dialog is called when Jot It Down! needs email from the user to support its email integration functionality.

LogInDialog

This is a JDialog sub class. This log in dialog is called when Jot It Down! needs username and password to log in to Google in order to use Google Calendar.

SUB PACKAGE – reminder

This sub package manages the reminder and its display as well as its alarming sound.

Reminder

This controls the logic part for the reminder. It will find the time that the alarm window will appear and create that alarm window if the task is important; otherwise, the reminder will appear as a message on the tray

AlarmFrame

This is a subclass of JFrame. This initializes the appearance of the alarm window and starts ringing the sound. The sound will not stop until the user presses “stop” button.

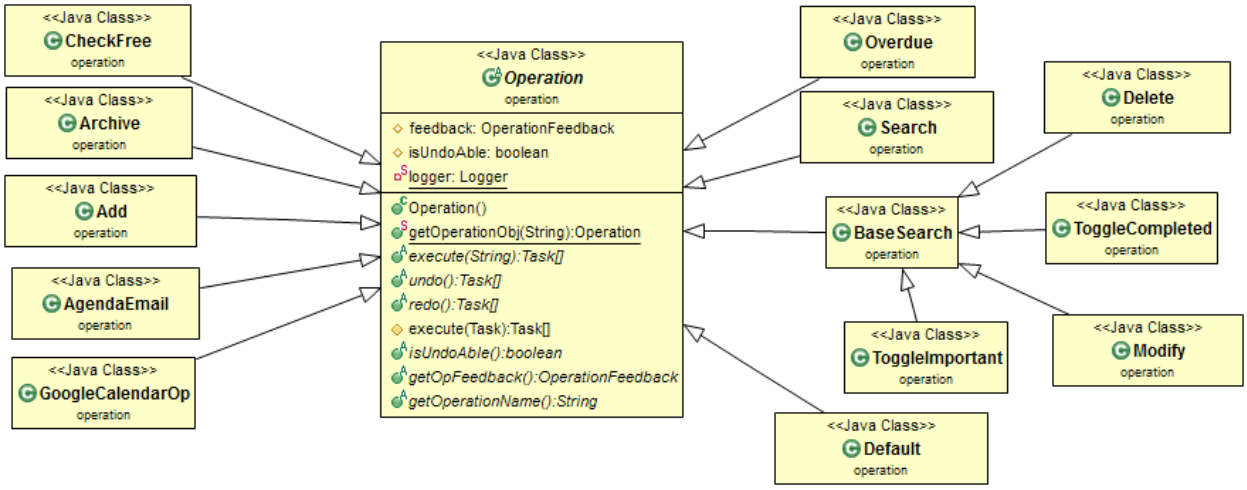
AlarmSound

It finds the music file and manages the alarm sound by providing the function to start and stop the sound.

Action Inner Class

ExitAction	exit the program
UndoAction	Undo the previous operation
RedoAction	Redo what have been undone
DeleteAction	Delete tasks that are selected on the task table
CompletedAction	Toggle completed on the tasks that are selected on the task table
ImportantAction	Toggle important on the tasks that are selected on the task table
OverdueAction	List all overdue tasks in task table
ListAction	List all tasks in task table
ExpandAction	Expand/contract the main window
HelpAction	Toggle help window
GCalendarAction	Log in to Google Calendar
GCalendarOutAction	Log out from Google Calendar
GCalendarSyncAction	Sync data with Google Calendar
GCalendarImportAction	Import tasks from Google Calendar
GCalendarExportAction	Export tasks to Google Calendar
ArchiveAction	Archive completed tasks
ImportArchiveAction	Import tasks from Archive
ClearArchiveAction	Clear tasks in the Archive

Package - Operation



Overview

This is the package that has implemented all the functionalities related to our Software. It consists of the abstract Operation class which consists of various abstract functions that are implemented in its sub classes. For each kind of functionality to be carried out an object of that class is created first. Then the execute() method of that object is called with necessary user command as parameter. This particular method is overwritten in each subclass. Thus through dynamic binding we ensure that correct execute function is called in each subclass. The kind of object or functionality executed depends upon the keyword specified by the user for example "add" for adding a Task. Another thing to note is that all the execute() functions in the subclasses return a task array of affected tasks to the calling function.

In the absence of a keyword no functionality is executed. However, future developers could look into it to develop default functionality under the default class. In addition, the operation package classes interact with the parser and storage manager to parse the user input into Task objects as well as amend the storage depending upon the operations.

BaseSearch Class

This is a very important class. We realized that for executing some of the functionalities the task must be searched/found first before we can perform an operation on it. Thus BaseSearch was developed to implement this similar search functionality available with delete, modify, toggleCompleted, toggleImportant functionalities respectively. This class extends from operation and other classes like Delete extend from it. So first this class searches for a task based on the user input and then returns an array of tasks to user to choose from. Then once the user chooses a

task to operate on, the GUI sends the command with the task Id of that task. Thus only when the task ID is found in the user Command, a particular functionality is performed on it.

In addition, for recurring events we have a delete all, completed all, star all functionality that can modify all the recurring tasks in one go without asking the user to individually perform these operations on each one of them. This is implemented through the executeall() function.

Multiple undo/redo for Multiple tasks

This is another unique feature of our operation package. By deriving itself from the same Operation class the same undo() and redo() functions can be overridden to implement this functionality. There are two aspects to its implementation. First, for undoing a operation one must store the operation that was affected e.g. if one wants to undo a delete operation, one must store the deleted event somewhere. Secondly, one must know the kind of operation to undo/redo. Thus to solve these two problems we first store the affected task in our operation subclass in the form of a task ArrayList and then store the operation object in a stack. So for undoing the last item from the stack is popped and its undo function called. Then this operation object is pushed in the redo stack. On initiating redo, a process similar to undo is repeated.

By storing our tasks in an ArrayList for most operations (except Modify) we can add multiple tasks to the ArrayList to undo and redo at once.

A word of caution is that not all functions can be undone. For example, GoogleCalendarOp etc. Thus to make sure we don't store items in our undo and redo stack that can't be undone we use the isUndoAble() function which returns a Boolean value to inform us about its undoable nature depending upon the nature of the operation and the stage of completion of a particular operation.

Other Classes:

- **Add** – used to add tasks to our to do list.
- **AgendaEmail** – used to specify the user Email for emailing agenda to the user. Also starts the thread for sending automatic email.
- **Archive** – used to shift the already completed tasks from the liveStorage to the archive storage to prevent cluttering of list. Provides functionality for clearing as well as importing of archives back to liveStorage.
- **CheckFree** – Used to check whether a certain time range is free or not.
- **Delete** – used to delete tasks from the to do list. Can be used to delete multiple tasks as well.
- **Default** – can be used to implement a default functionality in the future.
- **Modify** – allows the user to modify a task details.
- **GoogleCalendarOp** – entails several function pertaining to google calendar synchronization. Currently provides functionality for export, import and syncing our To Do List with the google calendar.
- **Overdue** – lists that tasks that have not been completed before the current date.

- **Search** – searches for the task according to the given search string
- **Toggle Completed** – Used to toggle the completed status of the task
- **Toggle Important** – used to toggle the important/ starred status of the task

The Key APIs to take note of are:

- + execute(String userCommand): Task[] - executes the core functionality of each class.
- + undo(): Task[] - implements the code related to undo for each feature provided it is relevant.
- + redo(): Task[] – implements the code related to redo for each feature provided it is relevant.

JIDLogic

This particular class acts as a façade between the UI and the backend. It serves as the back end brain of our software.

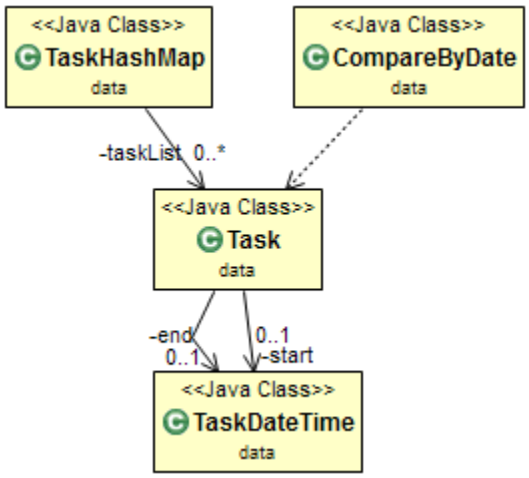
Key APIs-

- executeCommand(String UserCommand):Task[] - This functions handles all the features requested by the user. It also handles the undo and redo functionality as explained previously in the Operation package in a stack.
- JIDLogic_init() – initializes the Storages and runs the automatic email sender and auto save feature in a separate thread.
- JIDLogic_close() – this saves the file back to the hard disk while exiting the program.

Note to Developers:

This class interacts with Storage Manager and the UI Controller during execution.
Please do not forget to set the operation feedback as it tells the GUI more extensively about the kind of error that is occurring in the back end. It helps giving meaningful messages to the user.

PACKAGE – Data



Overview:

This package is the fundamental building block of the program. The classes of the package are used all over the software. Therefore, understanding this package is important for its further development.

Task

- This class is the fundamental element of the software. All the tasks in JotItDown are of this type.
- For a list of event attributes please view the code.

TaskDateTime

- The start/end date and time of each task can be stored in this class.
- Makes use of the Gregorian Calendar class in Java.

TaskHashMap

- This is the data structure we used for implementing the live Storage is a Hash Map.
- We choose these over other because addition, deletion and retrieval are O(1) processes for HashMaps.
- The key for each Task is a string composed of the date and time of the task and a randomly produced letter. The key is of the following format: $$__dd-MM-yyyyHHmmssC__$$, where C is an example of a randomly generated letter.

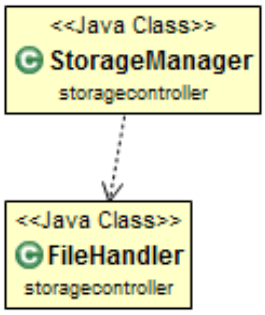
Key APIs:

- generateUniqueId()- generates a unique id which acts as a unique key of a specific task.

CompareByDate

- Compares start/end date and time of 2 tasks. It is implemented using the java.util.Comparator interface.

PACKAGE - Storage Controller



Overview

The data in the live Storage is stored in the form of a HashMap whereas it is permanently stored in the form of an XML file. The two forms of storage are continuously synced in order to prevent data loss.

StorageManager

- Interacts with the logic and accordingly performs the save and load operations.
- Is also responsible for the addition and deletion of tasks from live Storage and live Archives.

FileHandler

- Lowest Level function responsible for the permanent storage of JotItDown tasks and for all the read and write operations from and into the files respectively.
- Stores the live Storage and live Archives in the form of xml files. Also, stores the date when the agenda was sent and the email id to whom the agenda is to be sent in the form a text file.

Key APIs:

Storage Manager:

- loadFile()- loads the live Storage from the xml file.

- saveFile()- saves the live Storage into the xml file.
- addTask()- adds a task to live Storage.

PACKAGE- GCal

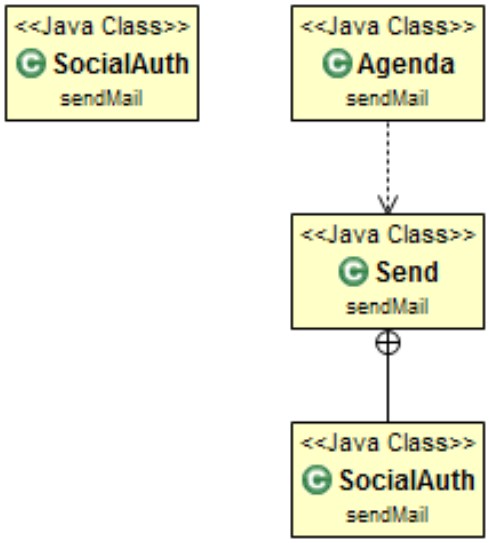
GoogleCalendar:

- Responsible for importing from Google Calendar to JotItDown and exporting to the Google Calendar from JotItDown.
- Converts a CalendarEventEntry object(a google Calendar Task object) into a Task object and vice-versa.
- Prevents duplication of tasks in both JotItDown and the Google Calendar by comparing the ids provided by Google calendar.

  Key APIs:

- calendarEntryToTask()- converts a google calendar entry into an object of Task class
- addTask()- converts an object of Task class into a google calendar entry.
- getAllEntries()- gets all the entries present in google calendar.
- sync()- exports from JotItDown to google Calendar. Compares the google ids to prevent duplication of events on either side.

PACKAGE- SendMail



Overview

This uses the JavaMail Library 1.4.5. This is the class that interacts with Agenda Email to send an automatic email to the user with the agenda of the coming day at a fixed time/

SocialAuth

- Authorizes the JotItDown email address with JavaMail.

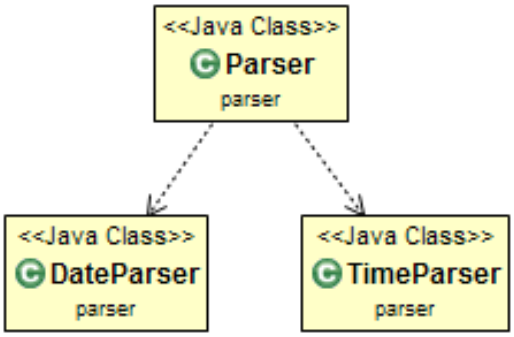
Send

- The function which sends the mail once its function is called.

Agenda

- Responsible for calling the function of Send after every 24 hours.
- Makes use of the daily Iterator and Scheduler classes.

PACKAGE: Parser



Overview

This package features classes that provide Natural Language Processing and Parsing capabilities for other classes to make use of. The main class in this package, Parser class, is dependent on the helper classes TimeParser and DateParser. It is only the Parser class which interacts with other classes. As such, it provides an API in the form of the following public functions which can be accessed by other classes i.e. the JID Logic class, Operation classes, etc.

TimeParser

This particular class deals with extracting the time from the user input and converting it into a format acceptable by the Gregorian Calendar. By using extensive regular expression matching it

allows the user to enter time in a variety of formats (listed in the code). Developers can look into this to make the time parsing more advanced in the future.

DateParser

Similar to the TimeParser, this class helps in parsing the date entered by the user. This also scope for a lot of future developments.

Parser

As mentioned previously this class parses most of the user input into Task objects to be used by the other

- +getErrorCode(): OperationFeedback
  Used by the relevant Operation class to obtain the error code, if any, that was recorded while parsing. For example, if the start time for a task entered by the user was greater than the end time specified.
- +validateEmailAdd(): Boolean
  Used by the relevant Operation class to validate an e-mail address i.e. to check if it is in the acceptable format. For example the address string “abcdef@domain.com” or “abcdef2@domain.co.in” when passed into this function returns TRUE while “abcdef@gmail” returns FALSE.
- +fetchTaskId(): String
  Used by the relevant Operation class to search for a valid Task ID in the parameter string. This function Returns a string representing a Task ID if a match is found.
- +fetchTaskIds(): String[]
  Used by the relevant Operation class to search for valid Task IDs in the parameter string. This function Returns a string array containing Task IDs.
- +fetchGCalDes(): String[]
  Used by the GoogleCalendar class to fetch information relevant for an object of Task class. A string representing a task listed in Google Calendar is passed as parameter and a string array containing the relevant information is returned.
- +parseForSearch(): Task
  Used by the Search operation class to parse a user input command for searching a task. A string representing the command is passed as parameter and a Task object, created from the details fetched by the user command, is returned.
- +parseForAdd(): Task[]
  Used by the Add operation class to parse a user input command for adding a task. A string representing the command is passed as parameter and a Task object array containing various tasks (based on the number of times the task is supposed to recur) is returned.

Testing

Currently we have implemented unit testing and some form of integration testing through JUnit Testing Plugin in Eclipse. Our tests can all be viewed under the test package of our source code.

To analyse the quality of our test cases we used the method of Code Coverage. For most of our completed test classes we have achieved a code coverage of more than 90 percent with some being as high as 98 percent. Code coverage was analysed using EclEmma plugin freely available on the Eclipse Market Place.

Our code is designed to enable developers like you to enhance the testability of our various modules. Thus we would recommend developing an automatic test driver that could be run with the JIDLogic class to test all the back end functionality. Else, developers could continue using our current approach of using JUnit for testing the vsrious modules of our software or any future changes made.

Major Changes from V0.1

1. Made the User Interface neater.
2. Included more hot keys.
3. Improved the quality of feedback messages given to the user.
4. Implemented additional functionalities like archive, overdue, checkfree.
5. Integrated Google Calendar synchronization.
6. Integrated daily email reminder for our user.
7. Improved the quality of our search function.
8. Made the parser more extensive by including more date and time formats.
9. Implemented test cases for various parts of the code.
10. Implemented undoing/redoing an operation conducted on multiple tasks in one go.
11. Implemented performing actions like delete/ toggle Completed/ toggle Important on multiple tasks in one go.
12. Integrated a small shortcut help feature.
13. Integrated desktop reminders for normal tasks and alarm reminders for important tasks.
14. Revamped the organization GUI classes.
15. Added basic support like adding/ deleting/ toggle completing / toggle starring the recurring events.

Future Goals

1. Integrating Facebook events and birthdays with our Jot It Down.
2. More extensive parsing with support for more date/time formats.
3. Implementing autocomplete and spell checker.
4. Developing more test cases.
5. Supporting the batch operations like editing on recurring events.
6. Improving the quality of tasks synchronization with Google Calendar.
7. Implementing a functionality to suggest free time slots to the user.
8. Integrating global hotkeys in our program.

Acknowledgements

We will like to thank our lecturer Dr. Zhou Lifeng, Mr. Orry and Ms. Xialing for their excellent support, constructive criticism through the course of this project. We would also like to acknowledge our senior Mr. Vaarnan Drolia whose timely input helped us overcome various difficulties while trying to develop this product.

Besides we will like to acknowledge ObjectAid Plugin for creating the diagrams we have used in this guide. Credit also goes to stackoverflow.com, Wikipedia and of course Google for helping us in state of dire need.

Conclusion

We hope that this developer's guide aids you in understanding our source code and also encourages you to make valuable changes to our product keeping the end user in mind. We reiterate our request to you to uphold the quality of our code. We really want you to have fun in developing our code and wish you all the very best!