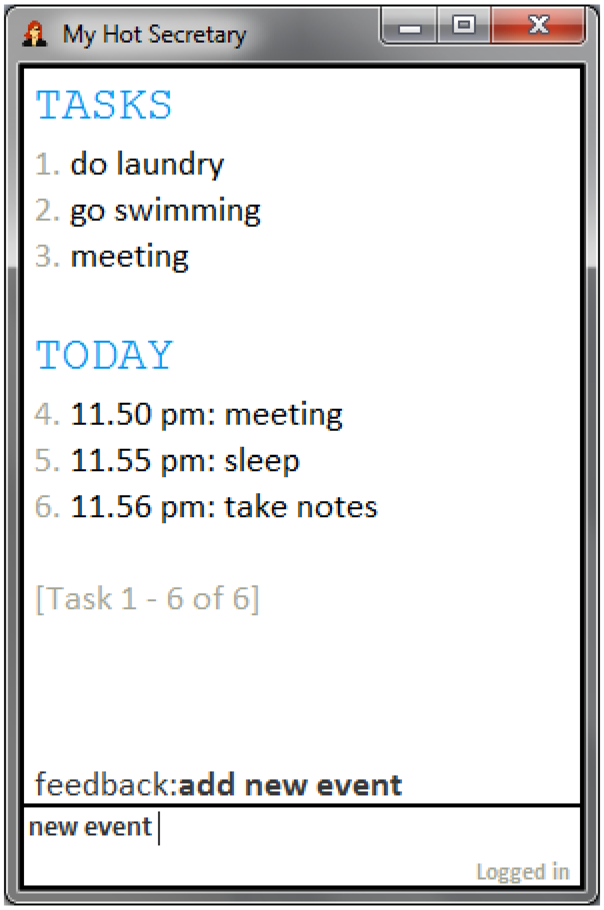
My Hot Secretary


My Hot Secretary

TASKS

1. do laundry
2. go swimming
3. meeting

TODAY

4. 11.50 pm: meeting
5. 11.55 pm: sleep
6. 11.56 pm: take notes

[Task 1 - 6 of 6]

feedback:**add new event**

new event

Logged in

Timothy Lim	**Shekhar B. R.**	**John Wong**	**Kahou Cheong**
Team leader	Test In-Charge	Integration In-Charge	Documentation In-Charge

Contents

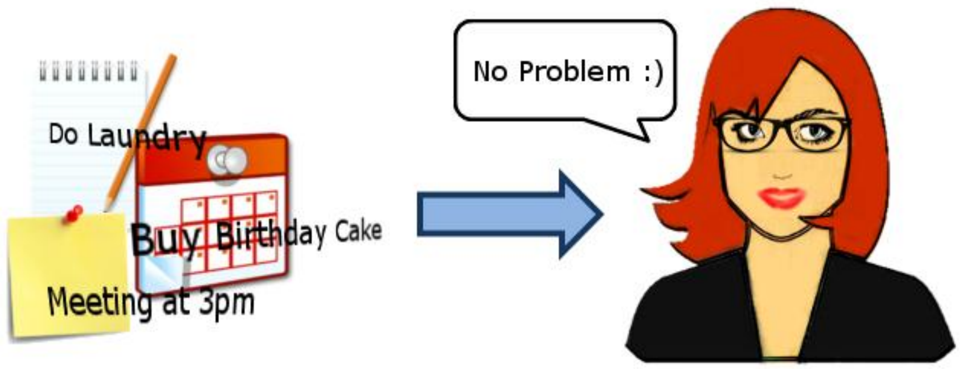
MyHotSecretary UserGuide

Installation Instructions

My Hot Secretary requires no installation, just download and run!

What is My Hot Secretary?

My Hot Secretary is an application that makes task management a breeze, just pass all your tasks to My Hot Secretary and let her manage them for you.



With My Hot Secretary you can easily create, retrieve, edit and delete all the tasks you need to keep track of. It's simple, here's how:

Adding a task

To add a task, just type the task name and the time.

For example, type "do laundry", hit enter, and My Hot Secretary will record it for you

Time information can also be specified, in fact, My Hot Secretary organizes your tasks into three categories, depending on the time information specified.

Floating Tasks: Tasks with no time information specified.

Deadline Tasks: Tasks with only one time specified.

Timed Tasks: Tasks with a start and end time.

Viewing Tasks

Once you've added your tasks, My Hot Secretary helps you organize it by category, time and date.

Viewing tasks in the Home Page

When you start up My Hot Secretary she will automatically show you the tasks you have for today as well as any floating tasks you may have.

Viewing tasks on a specific date or date range

To view tasks on a specific date, simply type "display (date)", for example, "display 21/11" will show you the tasks you have on for 21st November this year. You can also view tasks over a specific time range by typing "display (date) to (date)"

Viewing tasks by category

Lastly, you can view all tasks by typing “(category)”, for example, “deadline” will show your deadline tasks.

Editing Tasks

Have a meeting that changed timing, or need to rename a task? Not to worry, editing existing tasks in My Hot Secretary is easy.

Renaming a task

Type “rename (original task name) to (new name)”, for example “rename meeting to meeting with secretary” will edit the task with title “meeting” to “meeting with secretary”.

Editing a task’s timing

Type “edit (original task name) to (new time)”, for example, “edit meeting to 3pm” will change the timing of the task with title “meeting” to 3pm.

Changing a task

Type “edit (original task name) to (new task name and date)”, for example, “edit meeting to movie 4pm” will change the remove the task meeting and instead add a task movie at 4pm.

Deleting Tasks

Created a task that you no longer need to do? You can easily remove it from My Hot Secretary. Just type “delete (task name)”, for example, “delete laundry” will delete the task with title “laundry”.

Undo

Everyone makes mistakes; My Hot Secretary understands that, which is why it is easy to undo any mistakes you may have made, simply type “undo” to undo your last action.

Google Calendar

Need to access your tasks on the go? My Hot Secretary helps you sync all your tasks with your Google Calendar account, just type “sync”, type in your account details, and you’re good to go!

Marking Tasks as Done

There’s no greater satisfaction then completing tasks, increase the satisfaction by telling My Hot Secretary about it, just type “done (task name)”, for example “done laundry” will mark the task with title “laundry” as done, this will also move the task into a separate Google Calendar: “Completed Tasks (MHS)”.

Command Reference

Single Date

General Format	Example	Remarks
dd/mm	15/06	Recognized as 15th June this year, if that has already passed, it will be taken to mean 15th June next year
dd mmm	15 Jun	Recognized as 15th June this year, if that has already passed, it will be taken to mean 15th June next year
dd/mm/yyyy	15/06/2012	Recognized as 15th June 2012
dd mmm yyyy	15 Jun 2012	Recognized as 15th June 2012
Day	Weds	Recognized the closest Wednesday closest to the current date. Will find next Wednesday if today is Wednesday.

Date Range

General Format	Remarks
(1st date) to (2nd date)	Recognized as 12am of 1st date to 23.59pm of 2nd date
This Week/ Month/ Year/ Weekend	Recognized as the time range as stipulated.

Single Time

General Format	Example	Remarks
Hh:mm	15:30	Recognized as 3.30pm
hh.mm am/pm	3.30 pm	Recognized as 3.30pm

Time Range

General Format	Remarks

(1st time) to (2nd time)	Time range recognized as 1st time to 2nd time

Adding a Task

Task Type	General Format	Example	Remarks
floating	(task name)	do laundry	Floating task with title “do laundry” will be added
deadline	(task name) (time)	do laundry 7pm	Adds a deadline task “do laundry” with time information recognized as “7pm”
timed	(task name) (1st time) (2nd time)	do laundry 7pm to 8pm	Adds a timed task “do laundry” with time information recognized as “7pm to 8pm”

Viewing Tasks

Command	Remarks
Home	Displays floating tasks, deadline and timed tasks for today
Today	Displays deadline and timed tasks for today
Tomorrow	Displays deadline and times tasks for tomorrow
(category)	“Floating”: Displays all floating tasks “Deadline”: Displays all deadline tasks “Timed”: Displays all timed tasks
(date)	Displays all tasks from 12am to 11.59pm of the specified date.
(1st date) to (2nd date)	Displays all tasks from 12am of the 1st date to 11.59pm of the 2nd date.

Editing Tasks

General Format	Example	Remarks

rename (task name) to (new task name)	rename laundry to wash clothes	Renames task with title “laundry” to “wash clothes”
schedule (task name) to (time)	schedule meeting to 3pm	Schedules task with title “meeting” to be at time “3pm”

Marking Tasks

General Format	Example	Remarks
mark (task name)	mark laundry	Marks the task with title “laundry” as completed
unmark (task name)	unmark laundry	Marks the task with title “laundry” as uncompleted

Deleting Tasks

General Format	Example	Remarks
delete (task name)	delete laundry	Deletes the task with title “laundry”

Undo/Redo

General Format	Example	Remarks
Undo	undo	Undo last task
Redo	redo	Redo last undo

Sync

Command	Remarks
Sync	Syncs all tasks from 1 month before current date to 1 year after current date. My Hot Secretary will prompt for login information if you are not currently logged in

Login	Authorizes MHS with your Google Services (Calendar and Tasks) via OAuth2.0 Mechanism.
Logout	Logs out of your Google account. Tasks will not be synced until next login.

Index

General Format	Example	Remarks
(command) (index)	delete 1	At any point in time, the index of a task is substitutable for the task name. So “delete 1” will delete the task at index 1. This feature is available for all commands.

Help

General Format	Example	Remarks
Help	help	Shows a list of available commands and their formats. Use indexes to get more detailed help.
Alt + x	Alt +x	Summons mhs to the screen from minimized view
Hide	Hide	Minimizes the UI

My Hot Secretary Developer Guide

What is My Hot Secretary?

My Hot Secretary is a desktop application that helps users to keep track of their tasks.

Functionality summary:

1. Support for natural language commands
2. Command hinting
3. Creating tasks
4. Retrieving and viewing tasks
5. Editing existing tasks
6. Deleting tasks
7. Synchronization Support for all tasks with Google Calendar & Tasks
8. Index based commands

Who is this guide for?

This guide is for anyone who wants to work on My Hot Secretary to understand the code behind the application.

Getting Started

My Hot Secretary is hosted on Google Code, the source code can be found at:

http://code.google.com/p/cs2103aug12-t14-2j/

Application Architecture

My Hot Secretary (MHS) utilizes the N-Tier architecture:

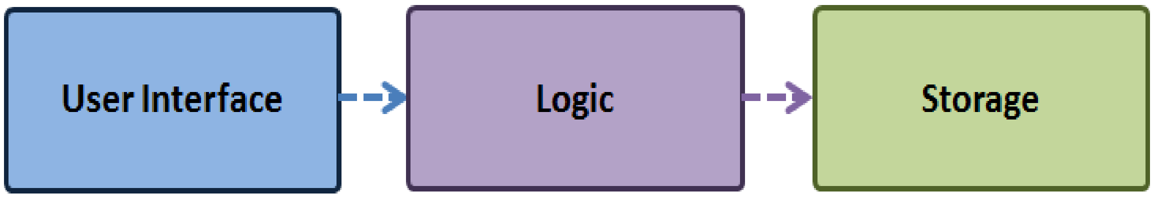


Figure 1 MHS Architecture overview

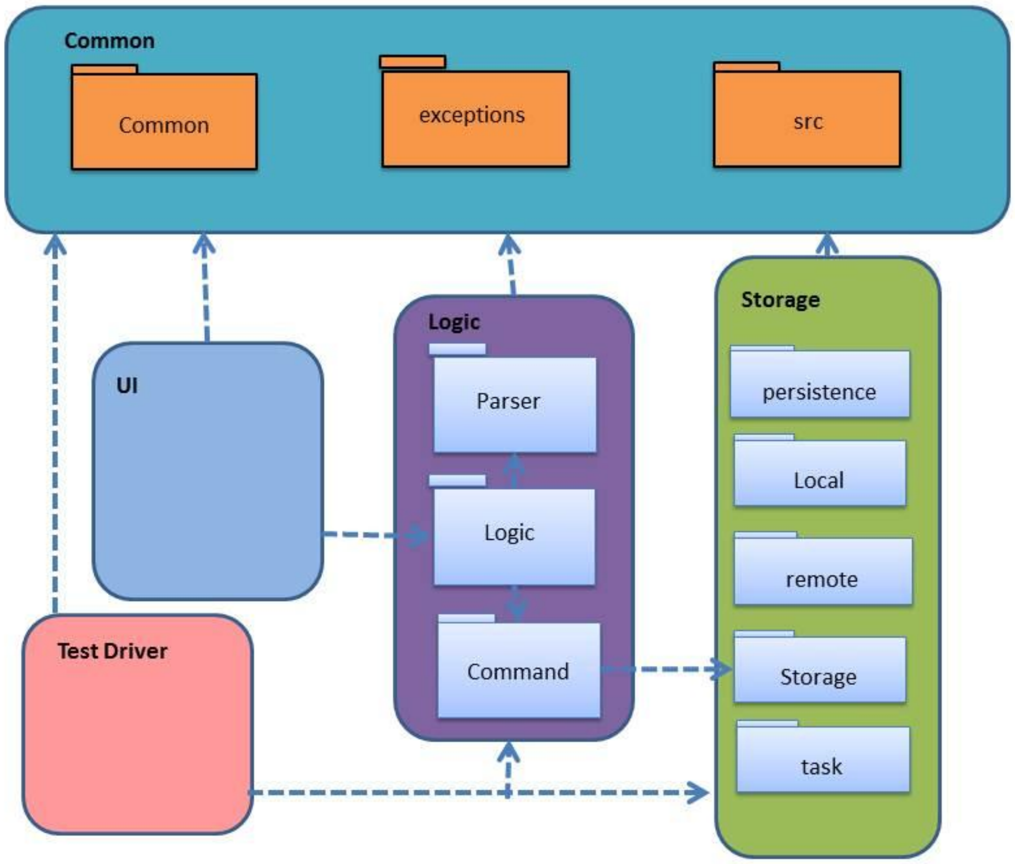


Figure 2 MHS Architecture detailed

The UI, logic and storage tiers all have access to a common tier in which there are methods that are used throughout the program.

The test driver tests the logic, database and common tiers.

User Interface

The User Interface-tier handles user-program interaction. It provides the interface for the user to input commands and for the program to display output to the user. It is also responsible for updating the logic-tier with user input and updating what is displayed to the user at the appropriate times.

Logic

The Logic-tier is responsible for processing user commands. It decides what to do with the user input sent from the User Interface-tier. It is responsible for parsing the command and executing them. It also decides what is to be displayed to the user.

Storage

The Storage-tier provides the methods for the Logic-tier to store and retrieve data. It is responsible for storing data persistently on local and remote storages and ensuring that all tasks are synchronized with remote storage services (Google Calendar and Tasks).

User Interface Package

UiController Class Diagram

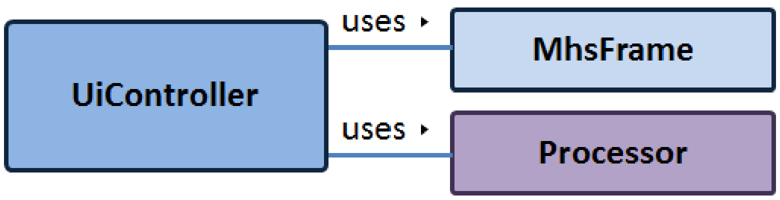


Figure 3 UiController Class Diagram

The responsibility of the UiController is to control interaction between the Processor and the MhsFrame.

MhsFrame Class Diagram

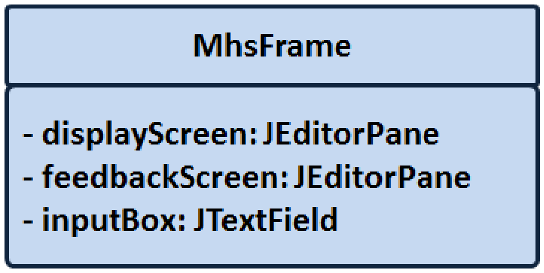


Figure 4 MhsFrame Class Diagram

MhsFrame provides the means for the user to interact with the application. It comprises an area to display output, an area to display one line feedback or confirmation messages, these two areas support the display of HTML formatted strings. Lastly, there is an input box for the user to input commands.

Observer Pattern

UiController makes use of the observer pattern in order to update the Processor when the user changes the state of MhsFrame, it does this through three nested classes which observe MhsFrame.

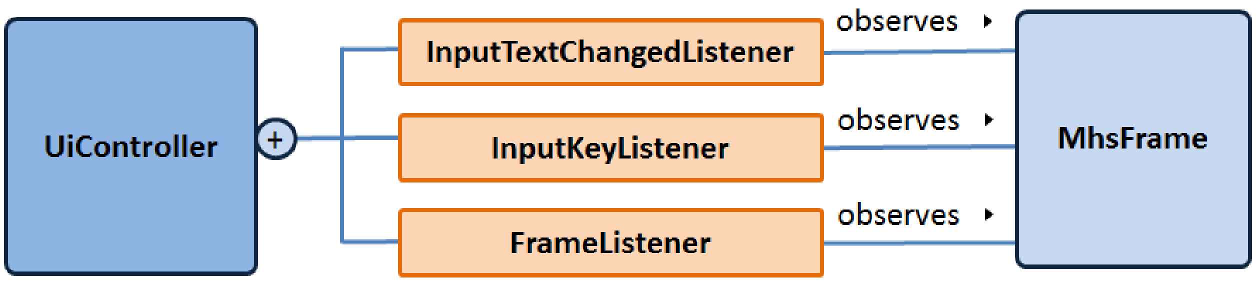


Figure 5 UiController observer pattern

InputTextChangedListener alerts UiController when the user changes the text in the input box of MhsFrame. UiController then updates Processor with the current user command.

InputKeyListener alerts the UiController when the user hits the enter key in MhsFrame, UiController then calls the Processor to execute the current command.

FrameListener alerts the UiController when the user resizes MhsFrame, UiController then updates the maximum number of lines being displayed to the user.

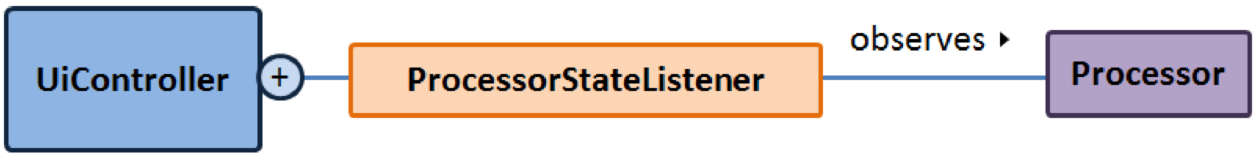


Figure 6 UiController processor observer pattern

Lastly, **ProcessorStateListener** alerts the UiController to update the display areas of MhsFrame when there is a change in the state of Processor (usually after Processor has processed a command).

Logic Package

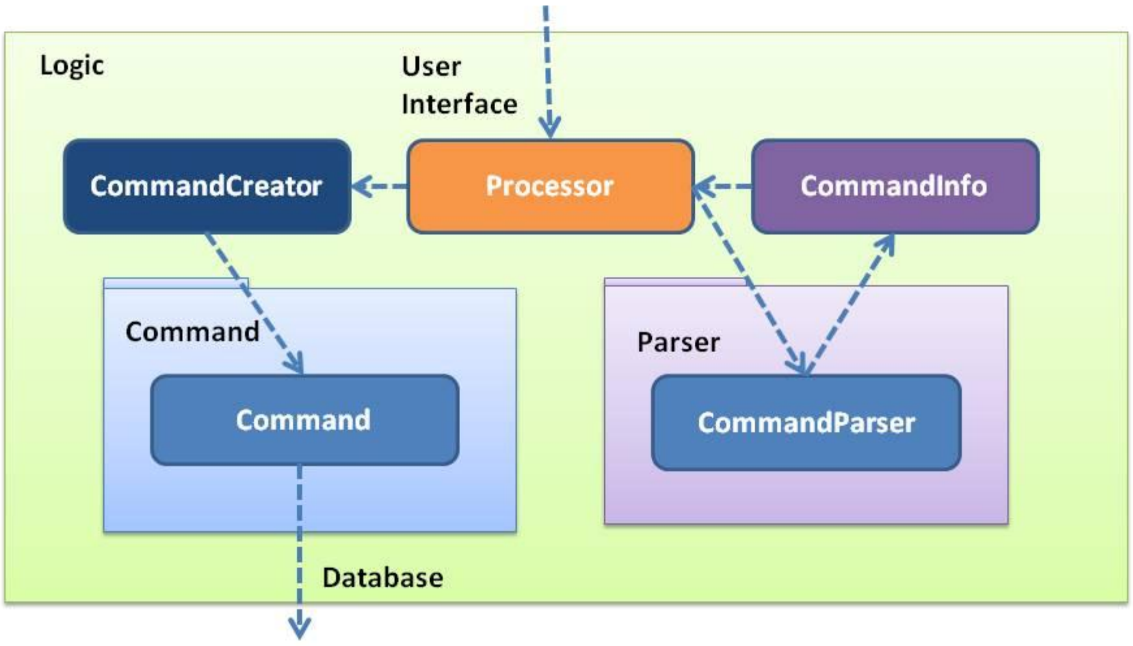


Figure 7 Logic tier Package diagram

The logic tier is where the command parsing and command execution is done. Processor is the façade of this tier and handles the interaction with the user interface while the command package will handle interactions with the database.

Processor Class Diagram

Processor class along with its associated Logic classes acts as the controller in MHS. It manages the flow of the user's command and leads it to execution.

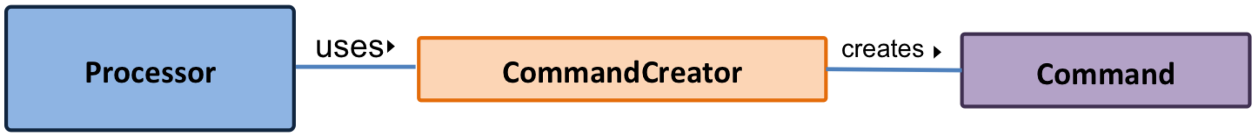


Figure 8 Processor class diagram

Command Pattern

To ensure scalability and ease of development Command Pattern has been implemented in the processor and its associated Logic classes.

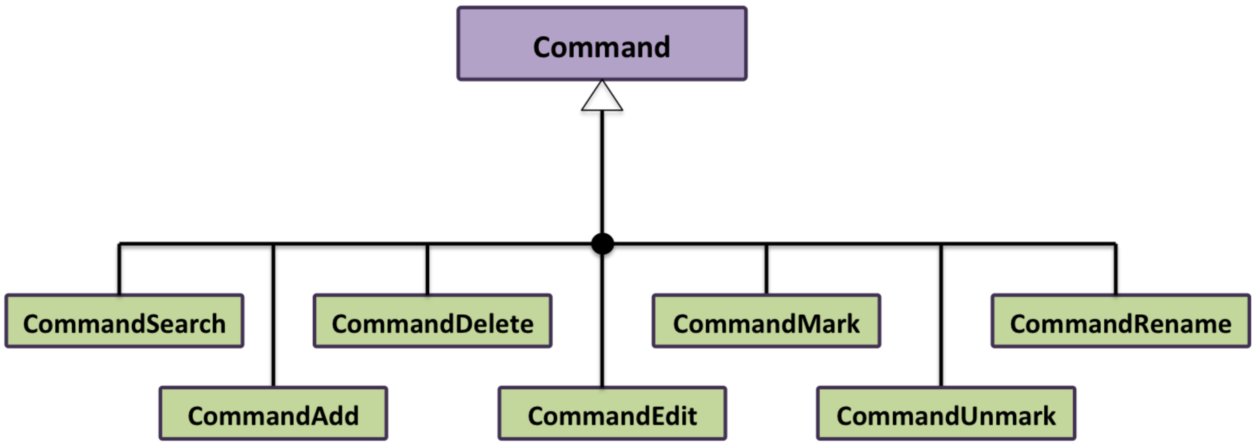


Figure 9 Command pattern

Flow of Logic in Command Pattern

When a command is received by the processor which is not a login/sync command, the Command Pattern logic comes into play.

- Input string is passed to Command Parser for parsing which returns an object of CommandInfo.
- This object is passed to a method in the singleton class **CommandCreator**. CommandCreator creates a new Command based on the input command type.
- Each subclass of **Command** has two methods execute() and executeByIndex() that handle the execution operations.

Command Class Diagram

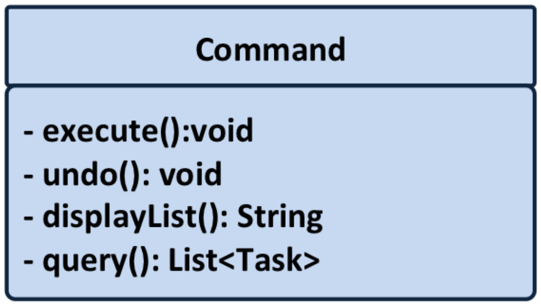


Figure 10 Command class diagram

- The Command queue has been excluded from the command pattern because Commands are immediately executed after they are created.

CommandParser Class Diagram

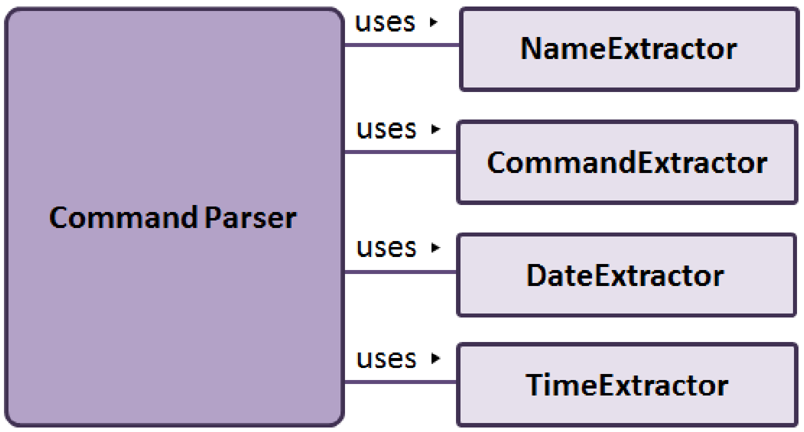


Figure 11 Command parser class diagram

Facade

CommandParser acts as the facade for the parser package.

The responsibility of the CommandParser is to extract the task name, command keyword, date and time elements of a user's command. It accomplishes this through four independent extractors.

The enumerations used in each of the extractors are used to list the various formats accepted by the program and the actual values that each of them point to. Addition of new key words can be added in the enumerations and would be reflected throughout the program.

NameExtractor

Enum SpecialKeyWords : The keywords that are used for sentence structure and omitted during name parsing.

Extracts the name of a task from the user's command by picking out all strings that are not dates, times, commands or special keywords (such as "to" or "on"). Anything within quotation marks will be extracted out as a name.

Once the name is extracted from the parse string, the name within quotation marks would be removed and returned to avoid interference with other extractors.

CommandExtractor

Enum CommandKeyWord : The commands that are recognized by the program and the actual command that they relate to.

Extracts command keywords, by comparing each string of a user's command with a command keyword list and extracting out the first word of the string.

DateExtractor

Enum DayKeyWord : The day keyword and the day of the week they point to.

Enum MonthKeyWord : The month keywords and the month of the year they point to.

Enum UniqueDateTypeKeyWord : The special key words that are related to dates.

Extracts strings that match various date formats.

TimeExtractor

Extracts strings that match various time formats.

Besides the task name, command keyword, date and time elements, a fifth element index is also extracted. This is done within CommandParser.

After all these elements are extracted, CommandValidator is used to combine all the elements together to form a logical command. For example, the date and time parameters extracted need to be combined for them to make sense.

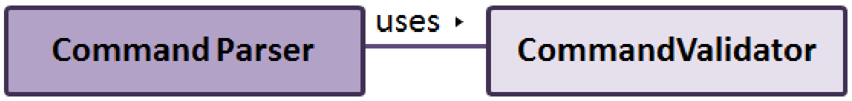


Figure 12 CommandParser class diagram 2

CommandValidator also validates and defaults all the parameters according to the command and creates a CommandInfo object which encapsulates the extracted and arranged data from the user’s command.

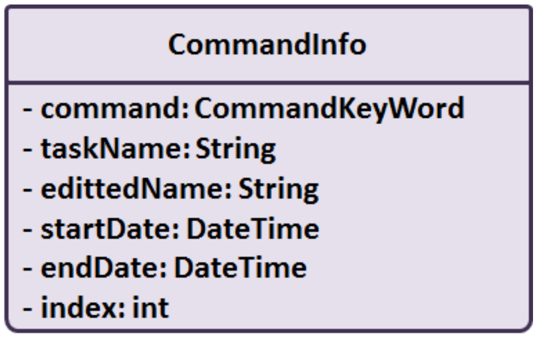


Figure 13 CommandInfo class digram

Command Validator will write all the parameters into CommandInfo which is returned by CommandParser. Processor reads from CommandInfo to get the required parameters needed for execution.

Storage Package

The Storage package contains classes for the storage tier which are used perform CRUD operations on tasks and other data that is persistent.

Database

Database is a singleton class, modelling the **Storage Tier** following the n-tier architecture.

Database Creation

DatabaseFactory encapsulates **Database** Creation and its life-cycle handling to ensure a one-hot Singleton Database with user specific pre-set configurations.

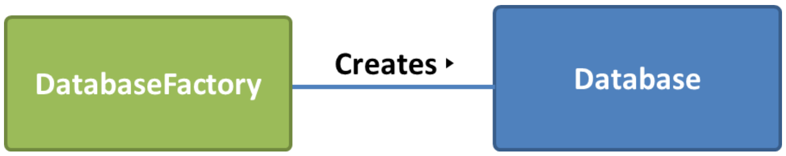


Figure 14 Database creation

Database Overview

Database interfaces persistent data storage operations on local disk as well as remote storage (Google Calendar Service) and synchronizing.

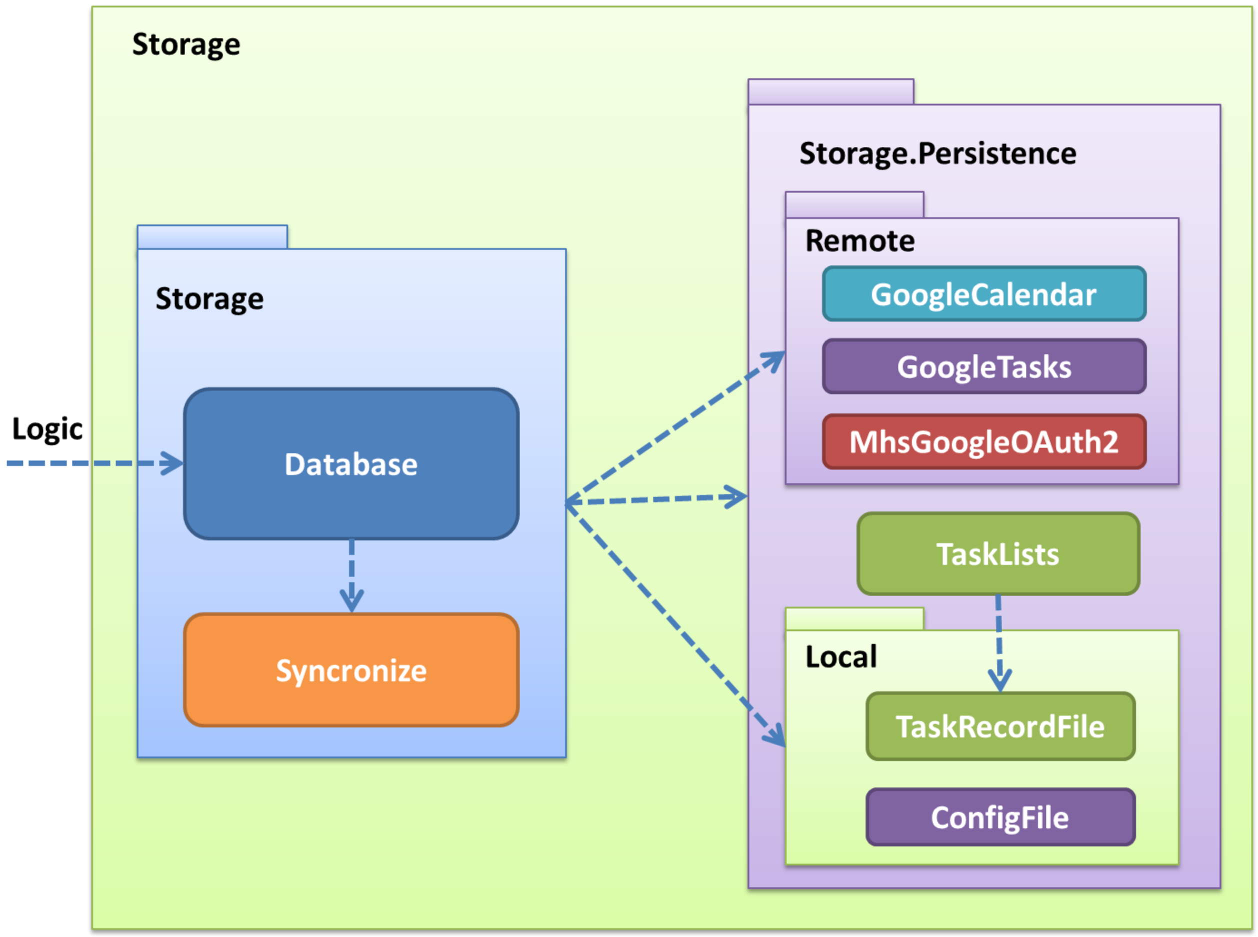


Figure 15 Storage package structure

Façade

The **Database** Class acts as a façade for storage packages Storage, interfacing operations on local and remote storage with a set of publicly available API.

Clients

The **Logic Tier** is using the Façade Pattern to access resources from the abstracted packages.

Database Policies

Database public methods that have return type of tasks / list of tasks returns clones of task(s) so that any modification on those task(s) do not change the original task.

Adding new task

Creates new task and assigns a new unique task id, created time and updated time.

InvalidTaskFormatException thrown when task specified does not meet the required format.

Updating Task

Updates existing task's editable fields and updates created time and updated time.

Non-editable fields: Google Ids and task last sync date times are preserved so as not to interfere with synchronization.

TaskNotFoundException thrown when task specified by taskId does not exist.
InvalidTaskFormatException thrown when task specified does not meet the required format.

Deleting Task

Tasks are marked as deleted and not removed for synchronization purposes.

TaskNotFoundException thrown when task specified by taskId does not exist.
InvalidTaskFormatException thrown when task specified does not meet the required format.

Querying Task(s)

IllegalArgumentException thrown when parameter is illegal.
TaskNotFoundException thrown when task specified by taskId does not exist.

Syncronize

Syncronize handles synchronization operations between local and remote storage.

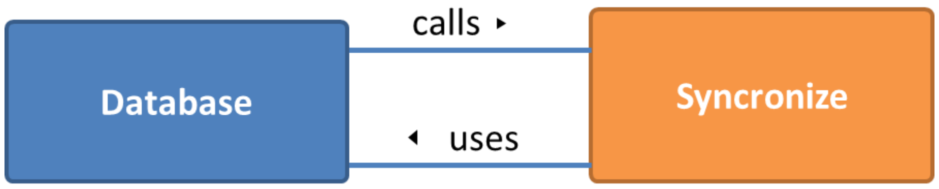


Figure 16 Database and synchronization association

Synchronization operations are done on a separate thread to prevent communication with remote storage classes **GoogleCalendar** and **GoogleTask** will not block the application.

Syncronization Flow

Sync Task Operations follows a command pattern.

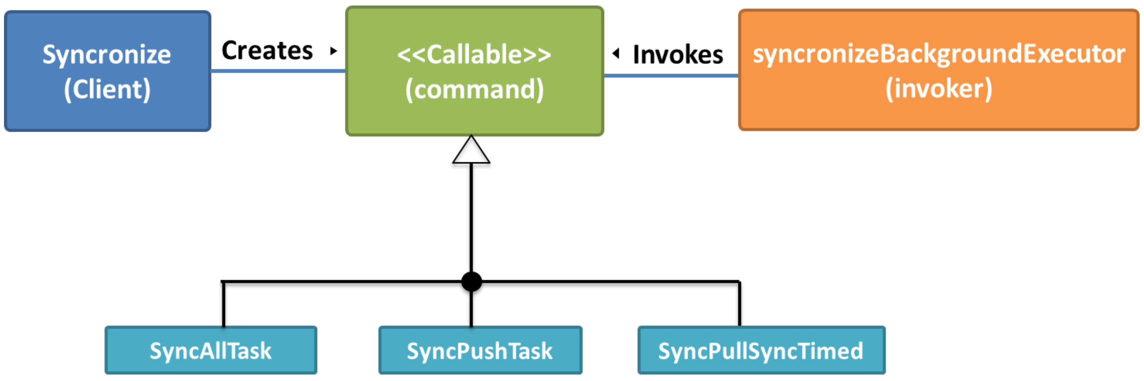


Figure 17 Syncronization command pattern

Client

Synchronize instantiates Sync Task Operations that is submitted to executor.

Invoker

Sync Task Operations are invoked via a single-threaded **executor** on a FIFO or timed basis.

Receiver

Sync Tasks Operations form the receiver, encapsulating information needed about the task.

Sync Task Operations Overview

1. *SyncAllTasks* – Performs full push-pull sync between local storage and remote storage.
2. *SyncPushTask* – Performs a push sync for a single task.

3. *SyncTimedPullTasks* – Performs a timed pull sync from remote storage. This operation is timed and scheduled to execute at a fixed rate.

FailSyncSilentlyPolicy: Exceptions are caught and silently handled during sync operations. Remote Sync is disabled on catching UnknownHostException.

TaskValidator

TaskValidator performs validation and check on Tasks' format and sync status.

Persistence Package

TaskLists

TaskLists abstracts CRUD operations on tasks in all task lists.

Persistence.Local Package

TaskRecordFile

TaskRecordFile handles CRUD operations of tasks on the task record file as well as file operations, such as creation, saving and loading of tasks

Calls to save to File are synchronized to prevent File I/O exceptions.

ConfigFile

ConfigFile handles CRUD operations of configuration parameters on the configuration file as well as file operations such as creation, saving and loading.

Persistence.Remote Package

Remote Package contains classes pertaining to remote storage.

MhsGoogleOAuth2

MhsGoogleOAuth2 interfaces the OAuth2.0 authentication mechanism to allow users to grant MHS access to their remote storage services and data.

GoogleTask

This class provides methods for the application to create, retrieve, update and delete tasks a user's Google tasks. It utilizes the Google Tasks API and java code provided from:

https://developers.google.com/google-apps/tasks/

GoogleCalendar

This class provides methods for the application to create, retrieve, update and delete events in a user's Google Calendar. It utilizes the Google Calendar Data API and java code provided from:

http://code.google.com/p/gdata-java-client/downloads/list

GoogleCalendarMhs

This class provides the methods to interface Google Calendar with the classes of MHS, it also handles two of the user's calendars: the default calendar and the MHS completed tasks calendar

Common Package

HtmlCreator

HtmlCreator contains the constants and methods to create basic HTML for MHS

MhsLogger

MhsLogger is a singleton class providing a logger instance (java.util.logger) for application-wide usage.

MhsGson

MhsGson is a singleton class providing a *Gson* instance configured to work with Tasks (Timed, Deadline, and Floating) and JodaTime DateTime objects.

DateTimeConverter

Gson Converter Class for Joda DateTime to json and vice-versa.

TaskTypeConverter

Gson Converter Class MHS Tasks to json and vice-versa.

DateTimeHelper

Converts DateTime object to a standardized string that is used throughout the program.

Testing

Logging

My Hot Secretary implements logging for debugging purposes.

Log Levels Usage

Log Level	Purpose
SEVERE	• Messages explaining cause of application failure
WARNING	• Warning messages signifying cause for concern
INFO	• Application messages at key points
FINER	• Tracing messages for debugging purposes • Tracking messages for exceptions and throws

Junit testing

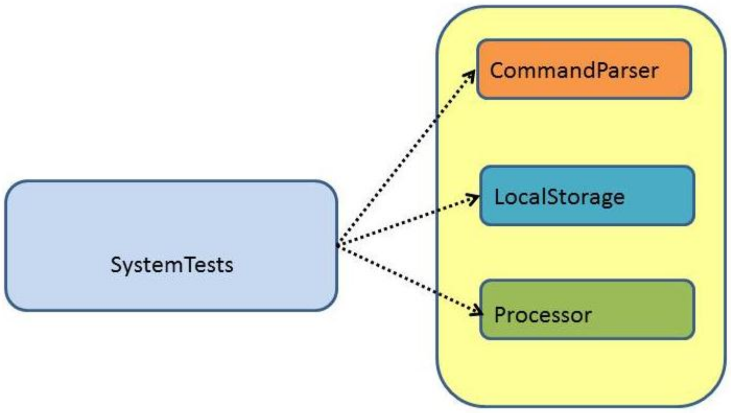


Figure 18 System tests

Tests are contained with the Test Package - mhs.test.

1. System Testing
    a. The system tests can be initiated by running the test class SystemTest under mhs.test package.
    b. The system takes a diff of the actual and expected output. If the output is 'ok' then all tests passed or else, you can find the test logs in the output files '*feedback<date>.html*' (Has all command feedback) and '*state<date>.html*' (has all the main screen outputs) in the folder – *System Test Files*.
    c. If the output does not match the tasks in the expected, the error is printed into an error.txt.
    d. The output files are html formatted so it can be viewed on any web browser.

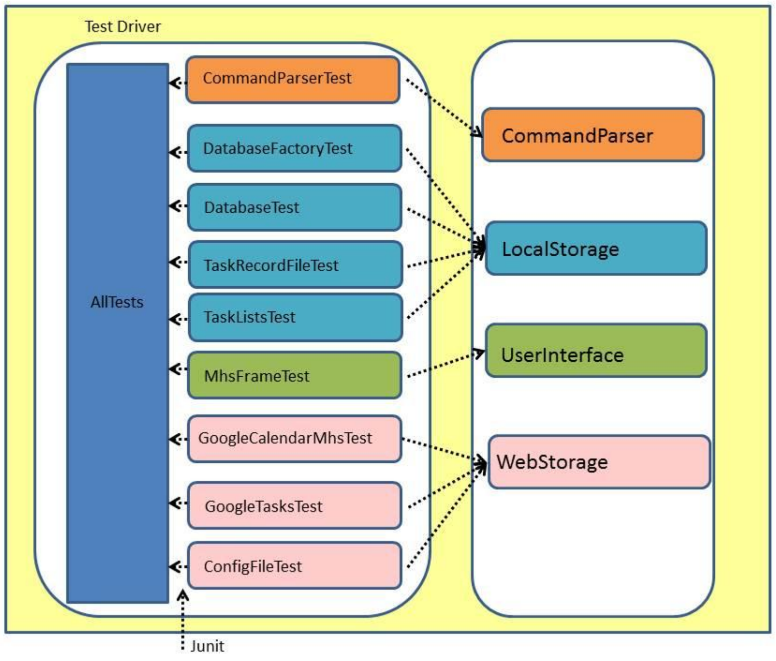


Figure 19 Unit Testing

2. Unit Testing
    a. Test Suite - All Tests
        i. Individual jUnit tests

Note: Database Test may have failed test cases if anti-virus is enabled since it is dependent on I/O operations, disable anti-virus before running tests.

Integration

New features should be implemented in a separate branch. The new branch should be merged with the trunk only after it is certain that the system with the new feature is stable. Assessment of stability is done through the testing process documented below.

Build Automation

Build automation is achieved using Apache Ant.

The following are built automatically:

- Jar distributable
- Javadocs
- Combined dependencies into a single jar file

Appendices

- MHS Javadoc - http://cs2103aug12-t14-2j.googlecode.com/hg/doc/index.html

Credits

1. Google API Client Java: http://code.google.com/p/google-api-java-client/
2. Date Time library : http://joda-time.sourceforge.net/
3. GSON: http://code.google.com/p/google-gson/

Future improvements

1. Multi user support.
2. Multi commands in 1 line.
3. Auto command recognition.
4. Recurring Tasks.
5. Notifications.