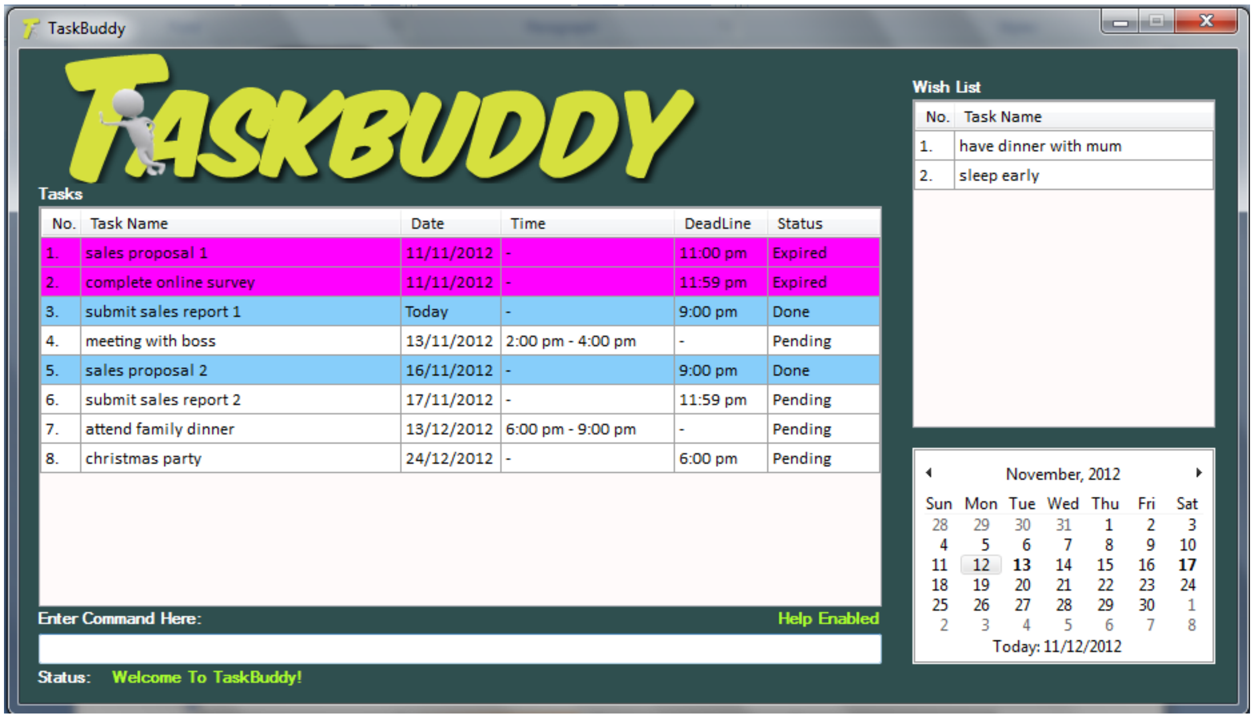
TASKBUDDY



TaskBuddy

Wish List

No.	Task Name
1.	have dinner with mum
2.	sleep early

Tasks

No.	Task Name	Date	Time	DeadLine	Status
1.	sales proposal 1	11/11/2012	-	11:00 pm	Expired
2.	complete online survey	11/11/2012	-	11:59 pm	Expired
3.	submit sales report 1	Today	-	9:00 pm	Done
4.	meeting with boss	13/11/2012	2:00 pm - 4:00 pm	-	Pending
5.	sales proposal 2	16/11/2012	-	9:00 pm	Done
6.	submit sales report 2	17/11/2012	-	11:59 pm	Pending
7.	attend family dinner	13/12/2012	6:00 pm - 9:00 pm	-	Pending
8.	christmas party	24/12/2012	-	6:00 pm	Pending

November, 2012

Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1
2	3	4	5	6	7	8

Today: 11/12/2012

Enter Command Here:

Help Enabled

Status: Welcome To TaskBuddy!

Ivan Chin Team leader, testing, UI implementation, coding	Yu Qing Documentation, Coding, UI design	Kar Wai Coding, Testing	Ming Show Coding, Testing

Contents

User Guide

What is TaskBuddy?

TaskBuddy allows you to quick add “To Do” tasks (tasks with a specific deadline), “Event” tasks (e.g. meeting 5-6pm) and “Wish-List” tasks (tasks with no specific deadline). Without having to use a mouse, you can easily update your scheduler on the go. You do not need Internet and all you have to do is to key in the keywords that are required.

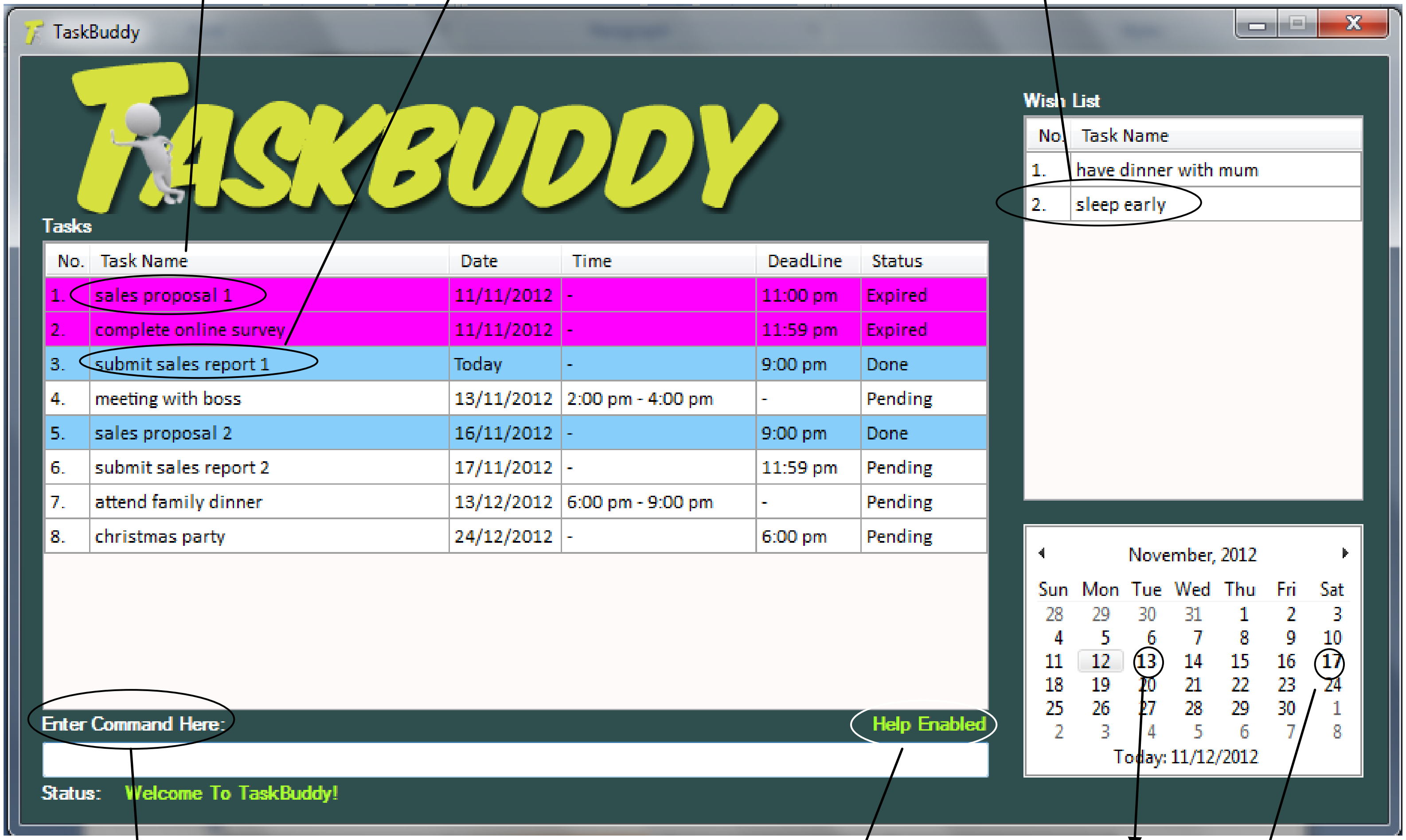
TaskBuddy User Interface will give you a clear view of all the tasks you have. You can also see a calendar at the right bottom corner of the User Interface and days which you have tasks due will be bolded.

General User Interface Guideline

Purple highlighted box:
To show expired tasks, which you have not completed

Blue highlighted box:
Tasks which you have marked as done

Reminder for tasks that you always have to do



TaskBuddy

TASKBUDDY

Tasks

No.	Task Name	Date	Time	DeadLine	Status
1.	sales proposal 1	11/11/2012	-	11:00 pm	Expired
2.	complete online survey	11/11/2012	-	11:59 pm	Expired
3.	submit sales report 1	Today	-	9:00 pm	Done
4.	meeting with boss	13/11/2012	2:00 pm - 4:00 pm	-	Pending
5.	sales proposal 2	16/11/2012	-	9:00 pm	Done
6.	submit sales report 2	17/11/2012	-	11:59 pm	Pending
7.	attend family dinner	13/12/2012	6:00 pm - 9:00 pm	-	Pending
8.	christmas party	24/12/2012	-	6:00 pm	Pending

Wish List

No.	Task Name
1.	have dinner with mum
2.	sleep early

November, 2012

Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1
2	3	4	5	6	7	8

Today: 11/12/2012

Enter Command Here:

Help Enabled

Status: Welcome To TaskBuddy!

Status message

Help bubble enabled for new users who are unsure of the commands

Dates which you have tasks due will be bolded on the calendar

General information regarding command format

- Commands are not case sensitive.

- If you wish to enter a full date:
    - 14 june 2012
- Dates entered without specifying year will be registered as the day of the current year (Note: dates that have already passed are considered as invalid dates)
    - 14 june
- To indicate a particular day or month, you can enter either full spelling or the first 3 letters of the day/month.
    - Monday or mon
    - 6 september or sep
- To indicate a day in the current week in your command, you can just enter the day.
    - Friday
- To indicate time, commands can be in a.m/p.m format.
    - 230pm
    - 10am
    - 230pm-330pm

A Quick Start

Thank you for choosing TaskBuddy. We have provided the most user-friendly experience by enabling help bubble at default when you run our program.

Hence, if you are lazy to read through the chunks of words and simply wish to have a quick start, try typing in the key words in the table below and you will be automatically prompted on what to type next.

Common Commands	Use
help	To activate the help bubble displaying all possible commands
add	To add a task on main task table
add wl	To add a task on wish list
back	To go back to main page after display of search results
delete	To delete a task
delete wl	To delete a task on wish list
disable help	To disable help bubble
enable help	To enable help bubble
exit	To exit the program
mark	To mark a task as done
search	To search for a task
undo	To undo an action
update	To update details of a task

Shortcut Keys	Use
ctrl + m	To minimise window
up/down Arrow Keys	To scroll up/down on the main task window
ctrl + up/down arrow keys	To scroll up/down on the wish-list task window

A Clearer Picture

For detailed explanation on how to use our program, you can also read the following explanation.

How do I add a task?

1. To add a deadline task, simply type "add [Task Name] [Time]" in the command bar.

For example, if you want to add a task called "submit report", you can enter in the following forms:

- If you enter "add submit report Friday" without indicating a specific time, it will mean the deadline of the task is registered as the end of that day (11.59pm of Friday).
- If you enter "add submit report Friday 230pm", it will mean the deadline is registered as the Friday, 2.30pm of current week.

2. To add a timed event, simply type "add [Event Name] [Date] [Duration]" in the command bar.

For example, if you want to add an event called "attend meeting", you can enter "attend meeting 10 Dec 2012 11am-2pm", the event will be registered.

3. To add a wish-list task, type "add [wl] [Task Name]" in the command bar.

For example if you want to add a wish-list task called "sleep early", you can enter "sleep early" and the task will be registered.

4. A green highlight will be seen on the most recently added task.

*For alternative command formats on dates/timing, refer to "general information regarding command format" section.

How do I search for a particular task?

There are 2 ways in which you can search for tasks, by name or by date.

1. To search by name, you can type in the keyword(s) of the name of the task which you wish to search for in the command bar with "search" command.
    - For example, type "search report" if you wish to search for a task with name report
2. To search by date, you can type "search" followed by the date/day command formats as stated in the earlier section.
    - For example, type "search 31 october" or "search Friday" for a task that falls on 31 October, or Friday of the week.

List of relevant search results will be displayed.

To return to the main tasks page after you have done searching, simply type "back".

How do I delete a task?

1. To delete a task in the "Tasks" table, you can simply type "delete [Task No.]" in the command bar.

For example, you can type "delete 2" in the command bar to delete the task with the No. 2 as shown in Tasks table.

2. To delete a task in the "Wish-List" table, you can simply type "delete [wl] [Task No.]" in the command bar.

For example, you can type "delete wl 2" in the command bar to delete a Wish-List task ranked No.2 in the Wish-List table.

3. A red highlight will be seen on the most recently deleted task.

How do I update a task?

To update a task, you can type “update [No.]” followed by “[Task Name] [Time]” which is the exact same format as how you add a task/event. (Please refer to **How do I add a task?** on how to add a task/event)

For example, if you want to update the details of the task with the No. 2 on the Tasks display table, you can type “update 2 submit lab report saturday 2pm”. The updated details will be saved automatically.

A yellow highlight will be seen on the most recently updated task.

How should I undo an operation?

You can undo by typing “undo” in the command bar and the previous operation will be undone.

How should I mark a task as done?

You can mark a task as done by typing “mark [Task No.]”.

You can also mark several tasks by typing “mark [Task No.] [Task No.] [Task No.]”.

How should I enable/disable help?

The help bubble is enabled by default as you run TaskBuddy. To disable it, simply type “disable help”.

To enable it again, simply type “enable help”.

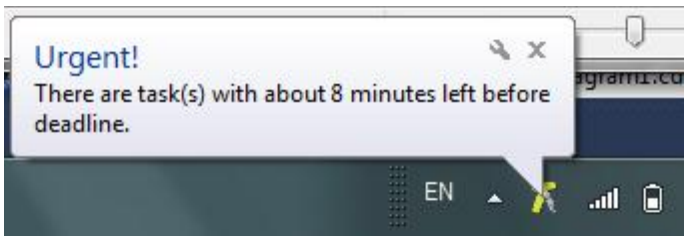
Colour Coding

The colour coding used is for following purposes:

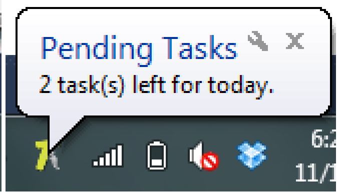
Colour	**To indicate:**
Blue	Done Tasks
Red	Newly deleted task
Yellow	Newly updated task
Purple	Expired Task
Green	Newly added task

Reminder Feature

We have also provided you with a reminder feature, which will remind you of the number of pending tasks that you have for the current day in intervals of 1 hour. It will also warn you when you have tasks that are due urgently.



Urgent!
There are task(s) with about 8 minutes left before deadline.



Pending Tasks
2 task(s) left for today.

Developer Guide

1. Project Vision

Long term goal: Become the **most user-friendly scheduler implemented in C++.**

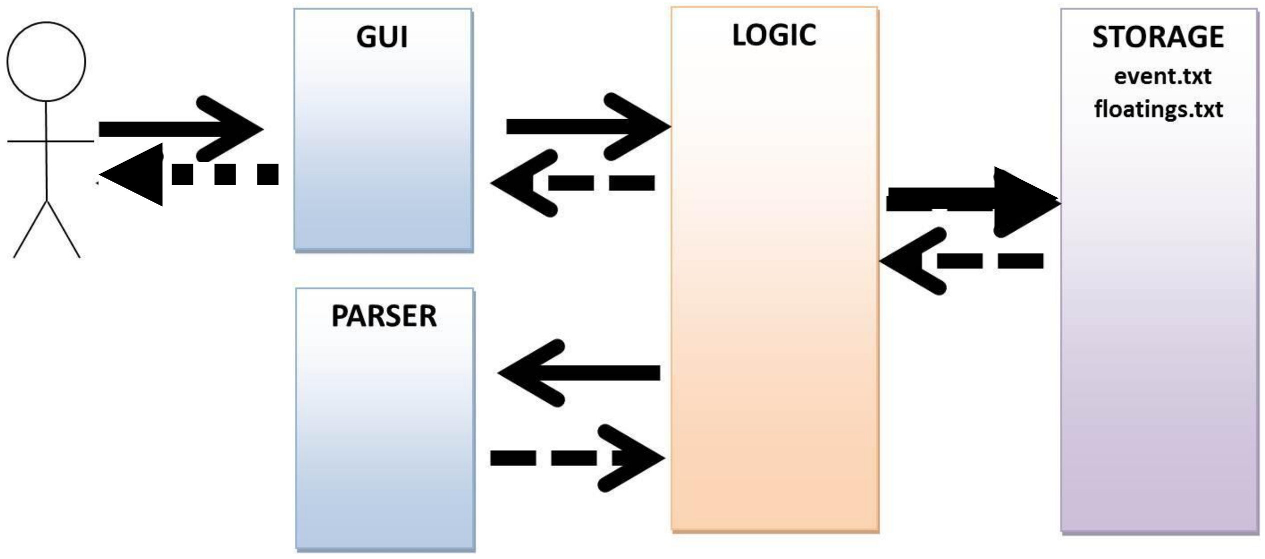
This project also serves as a god model and training ground for Software Engineering students who want to learn Software Engineering skills in the context of a real software product which will bring benefits to users.

2. Challenges

The project faces challenges in following areas.

- **Developers**: All developers are novices and their involvement with the project is short term and part time.
- **Code**: The program is coded from scratch so it takes more time and effort.
- **Testing**: No dedicated QA team.
- **Platform**: TaskBuddy is a Microsoft Visual C++ application.
    - It imposes various restrictions on the application, such as less easily customizable GUI
    - Developers are novices and there are many areas which more research has to be done to come up with certain codes.
- **Software Engineering**: As TaskBuddy serves as a model system for training students, it should also focus on applying good ‘software engineering’ techniques.

3. Basic Architecture



TaskBuddy allows users to quick add and delete deadline tasks (tasks with a specific deadline), timed events (e.g. meeting 5-6pm) and floating tasks (tasks with no specific deadline). Here is an overview of the main components:

- **GUI**: The GUI seen by users consists of 2 tables of contents. A bigger table is shown on the left which consists of a list of all the deadline tasks and timed events. A smaller table is show on the right which consists of a list of floating tasks to be completed. This GUI is generated using windows form and the contents are updated from Storage.
- **Logic**: The main logic of the program is to connect Storage and Parser so as to return the database to be displayed on the GUI.
- **Storage**: Storage uses containers to store data entered by user and implement the necessary commands entered by user.
- **Parser**: Parser helps to analyse user input which could then be implemented by storage.
- **Test Driver:** TaskBuddy makes use of manual testing for GUI and for automated regression testing, TaskBuddy makes use of G-Test to conduct unit testing.

4. Important Classes and APIs

4.1 GUI Class

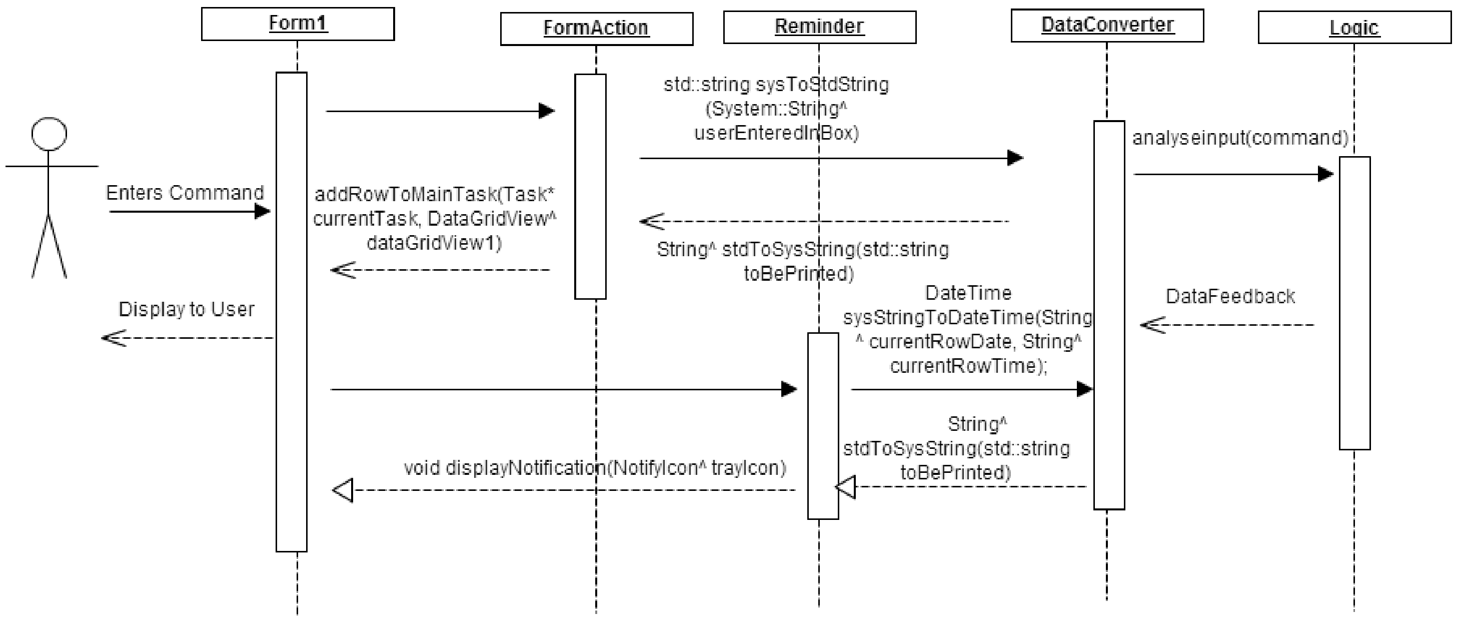
The GUI is created through windows form. The related files are:

FormAction.h/.cpp, Form1.h, AssemblyInfo.cpp: To allow for GUI design and messages shown to user and link it to the other Classes

DataConverter.h/.cpp: To convert data entered by user by returning the necessary "Static String" type for GUI to work and take in the right information from User

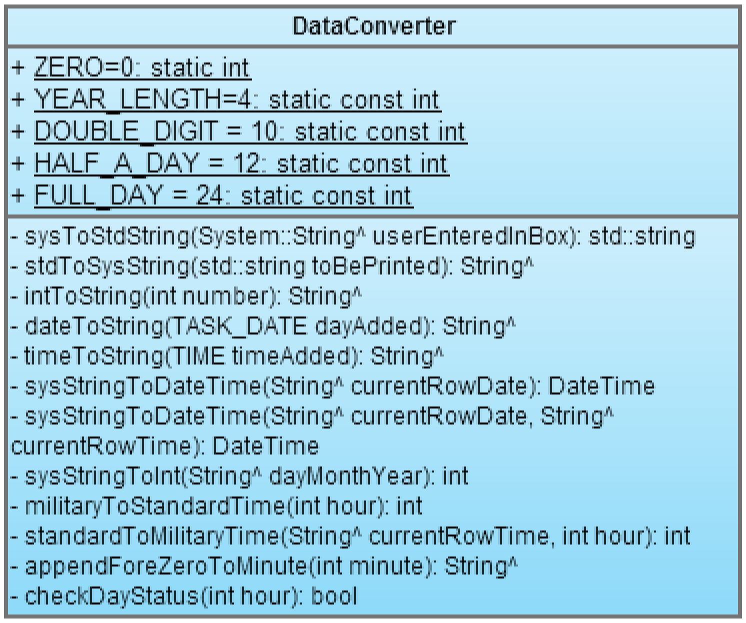
Reminder.h/.cpp: This class is written to provide reminder services for user. It will allow our .exe icon to be displayed at task bar telling users of the number of pending tasks for the current day. It will also send an urgent reminder if the task time is due in less than 2 hours.

The sequence diagram below shows how the different classes of GUI class interact with one another:

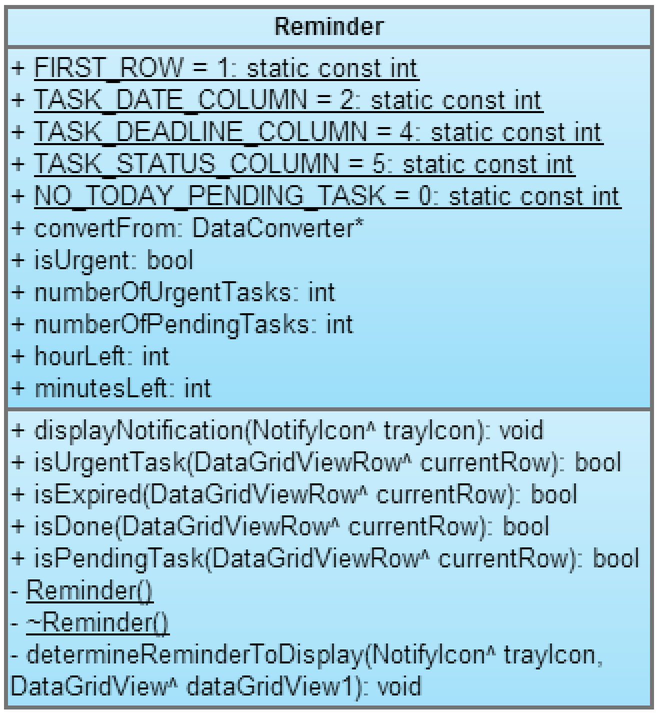

Form1
FormAction
Reminder
DataConverter
Logic
std::string sysToStdString (System::String^ userEnteredInBox)
analyseinput(command)
Enters Command
addRowToMainTask(Task* currentTask, DataGridView^ dataGridView1)
String^ stdToSysString(std::string toBePrinted)
DateTime sysStringToDateTime(String ^ currentRowDate, String^ currentRowTime);
DataFeedback
Display to User
String^ stdToSysString(std::string toBePrinted)
void displayNotification(NotifyIcon^ trayIcon)

Explanation: The GUI works when user enters a command which will be taken in by form1. FormAction class will take in this system string entered by the user and make use of DataConverter which will convert this system strings to standard strings for logic to use. Logic will then take in these strings and pass it to parser. After the logic has been done, it will be passed back to DataConverter in the form of DataFeedback and DataCconverter converts this information back to systemString for FormAction, which will then execute the necessary command, such as adding to main task, adding to wish list. This will eventually be displayed to user using Form1. This works the same for Reminder and DataConverter classes, whereby Reminder Class will receive the system string of date and time entered by using through Form1. Reminder Class then makes use of DataConverter to convert this system strings to Date and Time respectively, then pass this to Logic. After the logic has been done, it is then passed back to DataConverter which after the necessary conversions will return Reminder class information such as whether a task is due soon so that this reminder can be displayed on task bar accordingly.

Class diagrams for DataConverter and Reminder:



```
DataConverter
+ ZERO=0: static int
+ YEAR_LENGTH=4: static const int
+ DOUBLE_DIGIT = 10: static const int
+ HALF_A_DAY = 12: static const int
+ FULL_DAY = 24: static const int
- sysToStdString(System::String^ userEnteredInBox): std::string
- stdToSysString(std::string toBePrinted): String^
- intToString(int number): String^
- dateToString(TASK_DATE dayAdded): String^
- timeToString(TIME timeAdded): String^
- sysStringToDateTime(String^ currentRowDate): DateTime
- sysStringToDateTime(String^ currentRowDate, String^
currentRowTime): DateTime
- sysStringToInt(String^ dayMonthYear): int
- militaryToStandardTime(int hour): int
- standardToMilitaryTime(String^ currentRowTime, int hour): int
- appendForeZeroToMinute(int minute): String^
- checkDayStatus(int hour): bool
```

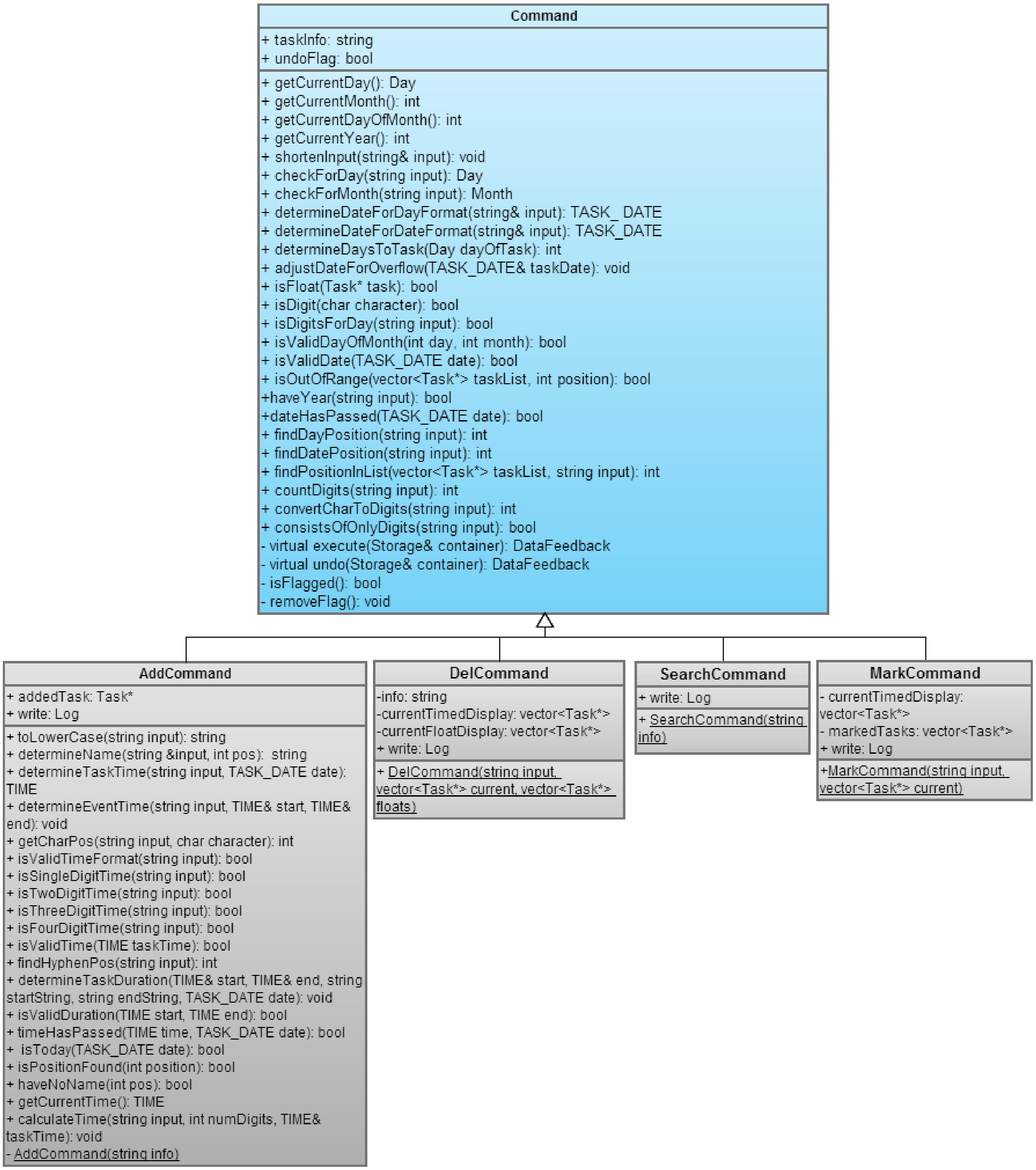


```
Reminder
+ FIRST_ROW = 1: static const int
+ TASK_DATE_COLUMN = 2: static const int
+ TASK_DEADLINE_COLUMN = 4: static const int
+ TASK_STATUS_COLUMN = 5: static const int
+ NO_TODAY_PENDING_TASK = 0: static const int
+ convertFrom: DataConverter*
+ isUrgent: bool
+ numberOfUrgentTasks: int
+ numberOfPendingTasks: int
+ hourLeft: int
+ minutesLeft: int
+ displayNotification(NotifyIcon^ trayIcon): void
+ isUrgentTask(DataGridViewRow^ currentRow): bool
+ isExpired(DataGridViewRow^ currentRow): bool
+ isDone(DataGridViewRow^ currentRow): bool
+ isPendingTask(DataGridViewRow^ currentRow): bool
- Reminder()
- ~Reminder()
- determineReminderToDisplay(NotifyIcon^ trayIcon,
DataGridView^ dataGridView1): void
```

4.2 Command class:

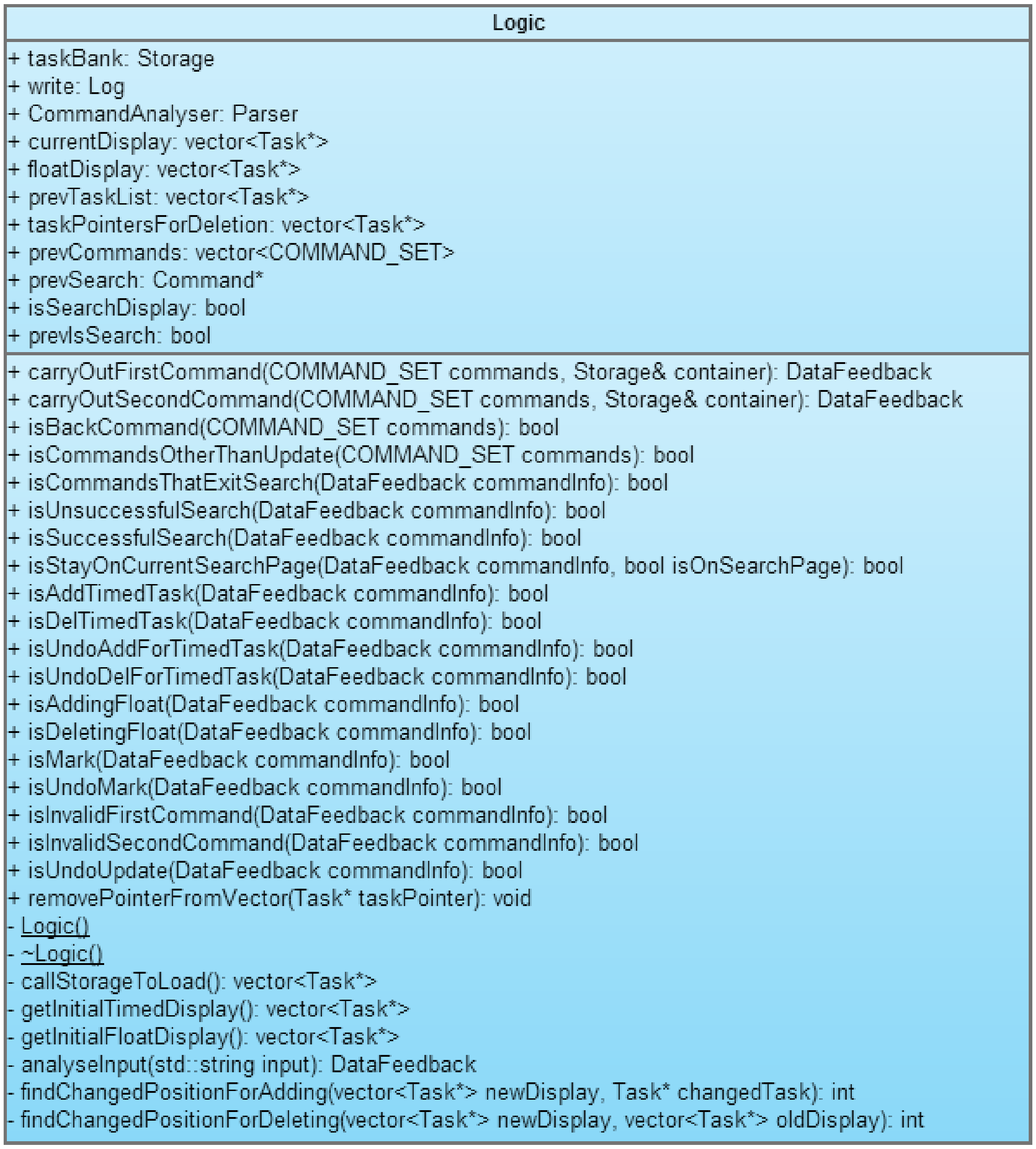
The Command Class is able to implement “add”, “delete”, “search”, “undo” and “mark” functions by analysing the user input and passing this information to the Parser Class. There are three classes that inherit from Command Class: **“AddCommand”**, **“DelCommand”** and **“SearchCommand”** and **“MarkCommand”.**

Class & Inheritance Diagram is in the next page(static const int variables have been excluded from class diagram, refer to codes in .h/.cpp files):


Command
+ taskInfo: string
+ undoFlag: bool
+ getCurrentDay(): Day
+ getCurrentMonth(): int
+ getCurrentDayOfMonth(): int
+ getCurrentYear(): int
+ shortenInput(string& input): void
+ checkForDay(string input): Day
+ checkForMonth(string input): Month
+ determineDateForDayFormat(string& input): TASK_ DATE
+ determineDateForDateFormat(string& input): TASK_DATE
+ determineDaysToTask(Day dayOfTask): int
+ adjustDateForOverflow(TASK_DATE& taskDate): void
+ isFloat(Task* task): bool
+ isDigit(char character): bool
+ isDigitsForDay(string input): bool
+ isValidDayOfMonth(int day, int month): bool
+ isValidDate(TASK_DATE date): bool
+ isOutOfRange(vector<Task*> taskList, int position): bool
+haveYear(string input): bool
+dateHasPassed(TASK_DATE date): bool
+ findDayPosition(string input): int
+ findDatePosition(string input): int
+ findPositionInList(vector<Task*> taskList, string input): int
+ countDigits(string input): int
+ convertCharToDigits(string input): int
+ consistsOfOnlyDigits(string input): bool
- virtual execute(Storage& container): DataFeedback
- virtual undo(Storage& container): DataFeedback
- isFlagged(): bool
- removeFlag(): void
AddCommand
+ addedTask: Task*
+ write: Log
+ toLowerCase(string input): string
+ determineName(string &input, int pos): string
+ determineTaskTime(string input, TASK_DATE date): TIME
+ determineEventTime(string input, TIME& start, TIME& end): void
+ getCharPos(string input, char character): int
+ isValidTimeFormat(string input): bool
+ isSingleDigitTime(string input): bool
+ isTwoDigitTime(string input): bool
+ isThreeDigitTime(string input): bool
+ isFourDigitTime(string input): bool
+ isValidTime(TIME taskTime): bool
+ findHyphenPos(string input): int
+ determineTaskDuration(TIME& start, TIME& end, string startString, string endString, TASK_DATE date): void
+ isValidDuration(TIME start, TIME end): bool
+ timeHasPassed(TIME time, TASK_DATE date): bool
+ isToday(TASK_DATE date): bool
+ isPositionFound(int position): bool
+ haveNoName(int pos): bool
+ getCurrentTime(): TIME
+ calculateTime(string input, int numDigits, TIME& taskTime): void
- AddCommand(string info)
DelCommand
-info: string
-currentTimedDisplay: vector<Task*>
-currentFloatDisplay: vector<Task*>
+ write: Log
+ DelCommand(string input, vector<Task*> current, vector<Task*> floats)
SearchCommand
+ write: Log
+ SearchCommand(string info)
MarkCommand
- currentTimedDisplay: vector<Task*>
- markedTasks: vector<Task*>
+ write: Log
+MarkCommand(string input, vector<Task*> current)

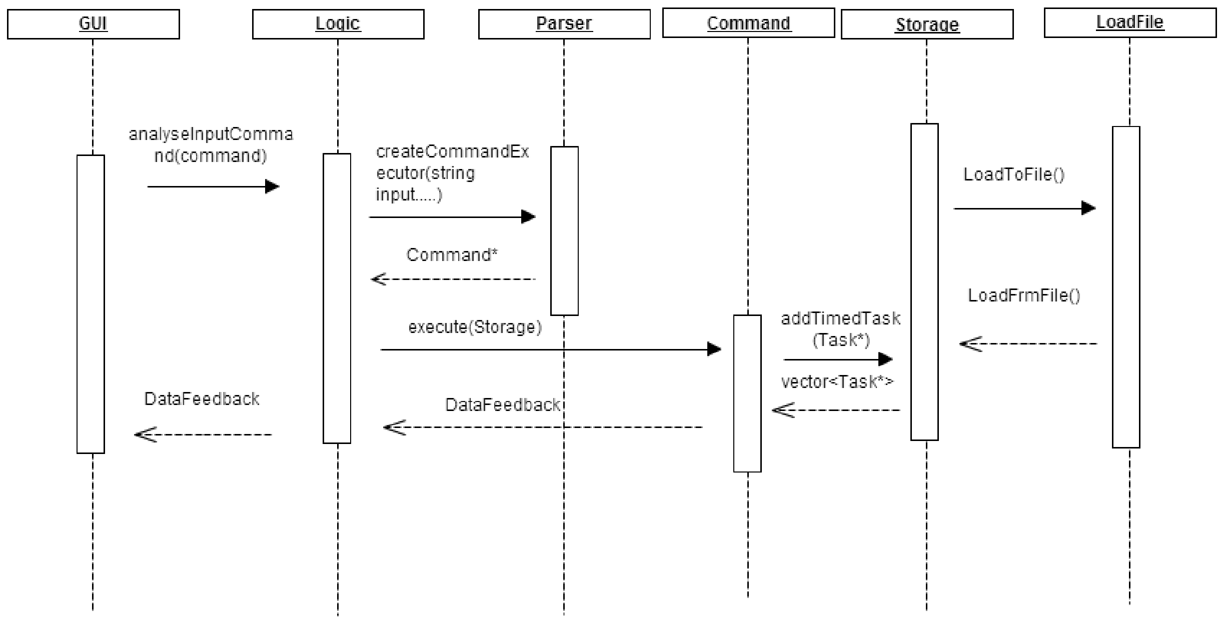
4.3 Logic Class

This class acts as a controller and links the “Parser Class” and “Storage class”. It gets Parser to disseminate the information and return the command which it will execute through “Storage Class”. This class also saves the tasks that are currently being displayed on the GUI.



```
Logic
+ taskBank: Storage
+ write: Log
+ CommandAnalyser: Parser
+ currentDisplay: vector<Task*>
+ floatDisplay: vector<Task*>
+ prevTaskList: vector<Task*>
+ taskPointersForDeletion: vector<Task*>
+ prevCommands: vector<COMMAND_SET>
+ prevSearch: Command*
+ isSearchDisplay: bool
+ prevIsSearch: bool
+ carryOutFirstCommand(COMMAND_SET commands, Storage& container): DataFeedback
+ carryOutSecondCommand(COMMAND_SET commands, Storage& container): DataFeedback
+ isBackCommand(COMMAND_SET commands): bool
+ isCommandsOtherThanUpdate(COMMAND_SET commands): bool
+ isCommandsThatExitSearch(DataFeedback commandInfo): bool
+ isUnsuccessfulSearch(DataFeedback commandInfo): bool
+ isSuccessfulSearch(DataFeedback commandInfo): bool
+ isStayOnCurrentSearchPage(DataFeedback commandInfo, bool isOnSearchPage): bool
+ isAddTimedTask(DataFeedback commandInfo): bool
+ isDelTimedTask(DataFeedback commandInfo): bool
+ isUndoAddForTimedTask(DataFeedback commandInfo): bool
+ isUndoDelForTimedTask(DataFeedback commandInfo): bool
+ isAddingFloat(DataFeedback commandInfo): bool
+ isDeletingFloat(DataFeedback commandInfo): bool
+ isMark(DataFeedback commandInfo): bool
+ isUndoMark(DataFeedback commandInfo): bool
+ isInvalidFirstCommand(DataFeedback commandInfo): bool
+ isInvalidSecondCommand(DataFeedback commandInfo): bool
+ isUndoUpdate(DataFeedback commandInfo): bool
+ removePointerFromVector(Task* taskPointer): void
- Logic()
- ~Logic()
- callStorageToLoad(): vector<Task*>
- getInitialTimedDisplay(): vector<Task*>
- getInitialFloatDisplay(): vector<Task*>
- analyseInput(std::string input): DataFeedback
- findChangedPositionForAdding(vector<Task*> newDisplay, Task* changedTask): int
- findChangedPositionForDeleting(vector<Task*> newDisplay, vector<Task*> oldDisplay): int
```

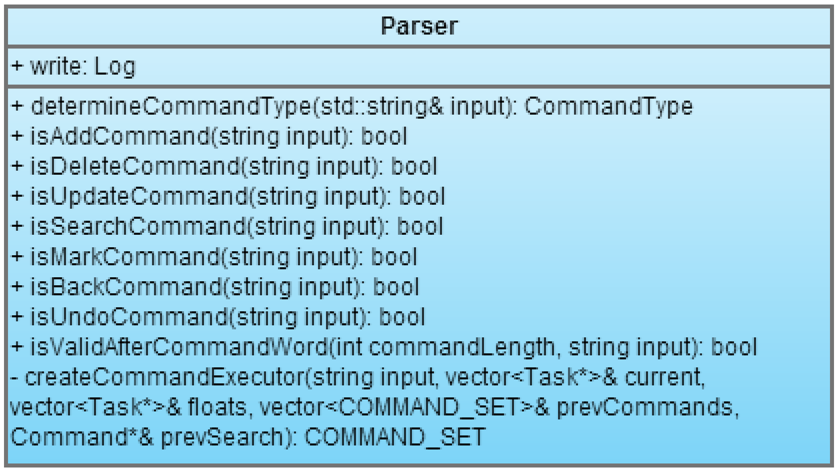
The following sequence diagram shows how the Logic Class interacts with the rest of the classes, when a user enters “add” command:



Explanation of diagram: As we can see from the diagram, when logic class first receives an user input from GUI, it will analyse the user input and call Parser to create the proper Command. Parser returns the appropriate Command which the Logic will use to execute the required action(could be add, delete, search depending on type of command returned) on the storage. At the start of program, storage will also call LoadFile to load any existing tasks that a user has saved in file and also load to file after the user exits the program. All of this will then be sent back in the form of DataFeedback to GUI.

4.4 Parser Class

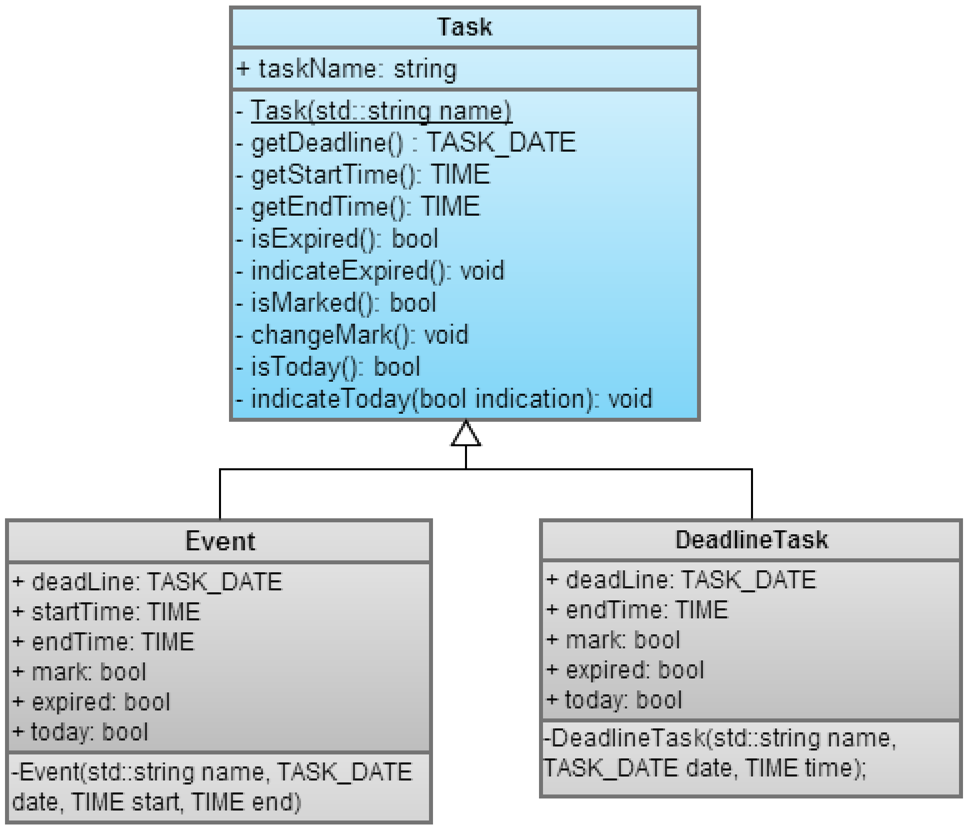
Parser Class analyses the command from the input string by accessing Command Class and returns the respective commands to Logic class where it will be executed.


Parser
+ write: Log
+ determineCommandType(std::string& input): CommandType
+ isAddCommand(string input): bool
+ isDeleteCommand(string input): bool
+ isUpdateCommand(string input): bool
+ isSearchCommand(string input): bool
+ isMarkCommand(string input): bool
+ isBackCommand(string input): bool
+ isUndoCommand(string input): bool
+ isValidAfterCommandWord(int commandLength, string input): bool
- createCommandExecutor(string input, vector<Task*>& current, vector<Task*>& floats, vector<COMMAND_SET>& prevCommands, Command*& prevSearch): COMMAND_SET

4.5 Task Class

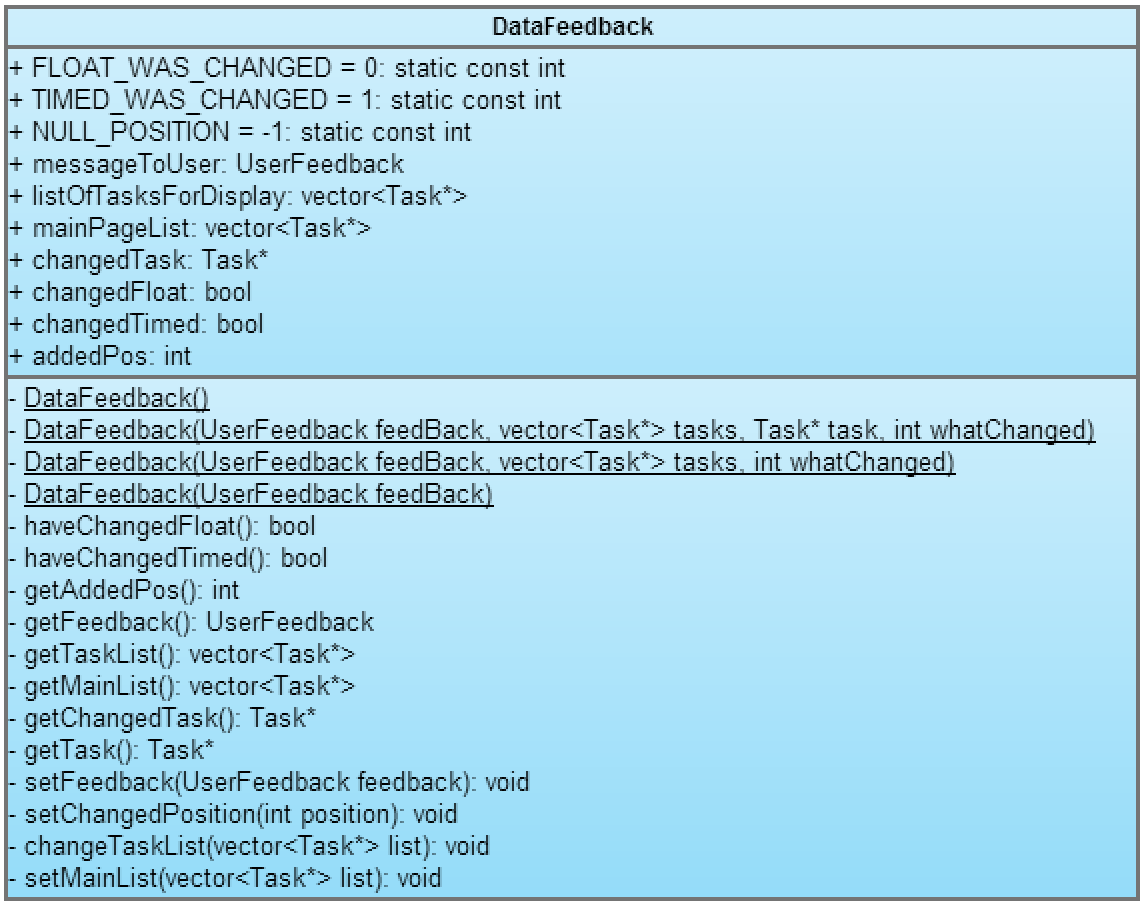
The Task Class mainly extracts information of different kinds of tasks and sends it to Storage for execution.
Task is a super class with “Floating Tasks” being the “parent” and DeadlineTasks(i.e Tasks with a deadline) and Events(i.e Timed tasks) inheriting from it.

Class & Inheritance diagram:



4.6 Datafeedback Class

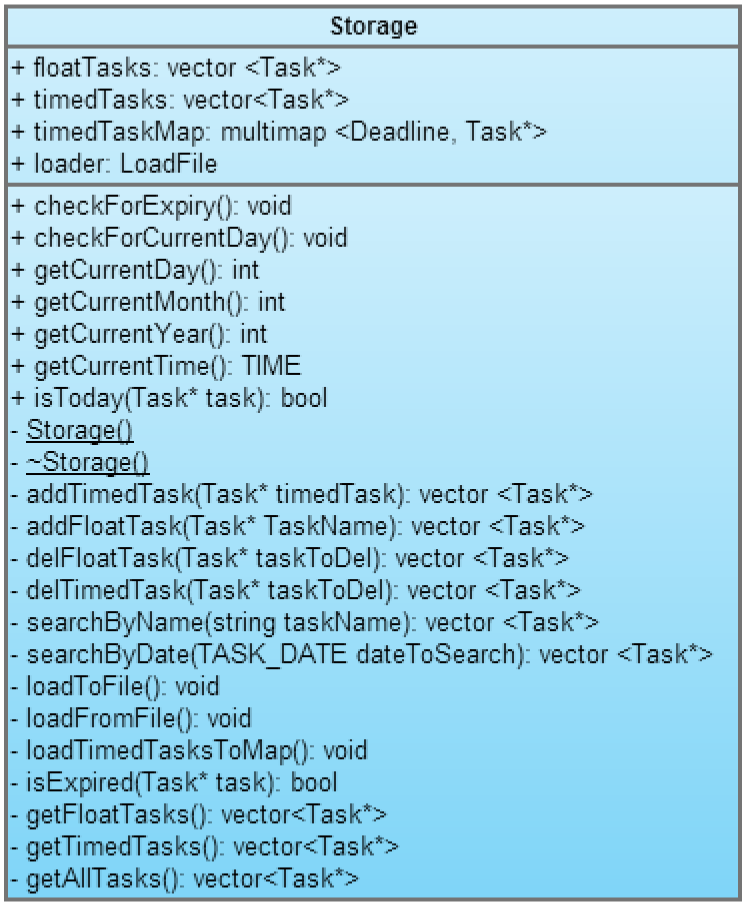
This class contains information required by GUI class that are required for display.
The information of this class includes: information about the users actions and any invalidities for message displays, if there is a need to change display for timedtasks or floating tasks, the vector of tasks to be displayed, position of tasks in the vector that are to be highlighted if applicable.
The above information is determined in the Logic class whereby Logic will analyse and set the information accordingly.
The following is the class diagram for DataFeedback:



```
DataFeedback
+ FLOAT_WAS_CHANGED = 0: static const int
+ TIMED_WAS_CHANGED = 1: static const int
+ NULL_POSITION = -1: static const int
+ messageToUser: UserFeedback
+ listOfTasksForDisplay: vector<Task*>
+ mainPageList: vector<Task*>
+ changedTask: Task*
+ changedFloat: bool
+ changedTimed: bool
+ addedPos: int
- DataFeedback()
- DataFeedback(UserFeedback feedBack, vector<Task*> tasks, Task* task, int whatChanged)
- DataFeedback(UserFeedback feedBack, vector<Task*> tasks, int whatChanged)
- DataFeedback(UserFeedback feedBack)
- haveChangedFloat(): bool
- haveChangedTimed(): bool
- getAddedPos(): int
- getFeedback(): UserFeedback
- getTaskList(): vector<Task*>
- getMainList(): vector<Task*>
- getChangedTask(): Task*
- getTask(): Task*
- setFeedback(UserFeedback feedback): void
- setChangedPosition(int position): void
- changeTaskList(vector<Task*> list): void
- setMainList(vector<Task*> list): void
```

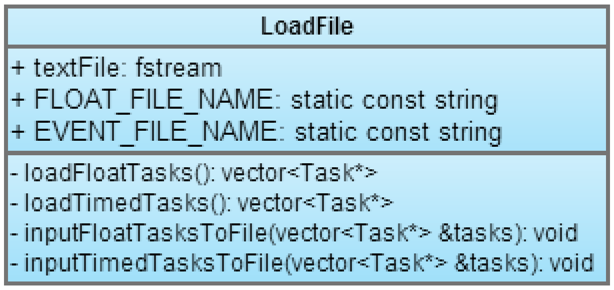
4.7 Storage Class

This class makes use of multimap to store all the DeadlineTask and Events using adding and deleting functions. This is to allow easy sort so that tasks can be properly sorted according to their finishing time and displayed according to urgency level on the GUI.


Storage
+ floatTasks: vector <Task*>
+ timedTasks: vector<Task*>
+ timedTaskMap: multimap <Deadline, Task*>
+ loader: LoadFile
+ checkForExpiry(): void
+ checkForCurrentDay(): void
+ getCurrentDay(): int
+ getCurrentMonth(): int
+ getCurrentYear(): int
+ getCurrentTime(): TIME
+ isToday(Task* task): bool
- Storage()
- ~Storage()
- addTimedTask(Task* timedTask): vector <Task*>
- addFloatTask(Task* TaskName): vector <Task*>
- delFloatTask(Task* taskToDel): vector <Task*>
- delTimedTask(Task* taskToDel): vector <Task*>
- searchByName(string taskName): vector <Task*>
- searchByDate(TASK_DATE dateToSearch): vector <Task*>
- loadToFile(): void
- loadFromFile(): void
- loadTimedTasksToMap(): void
- isExpired(Task* task): bool
- getFloatTasks(): vector<Task*>
- getTimedTasks(): vector<Task*>
- getAllTasks(): vector<Task*>

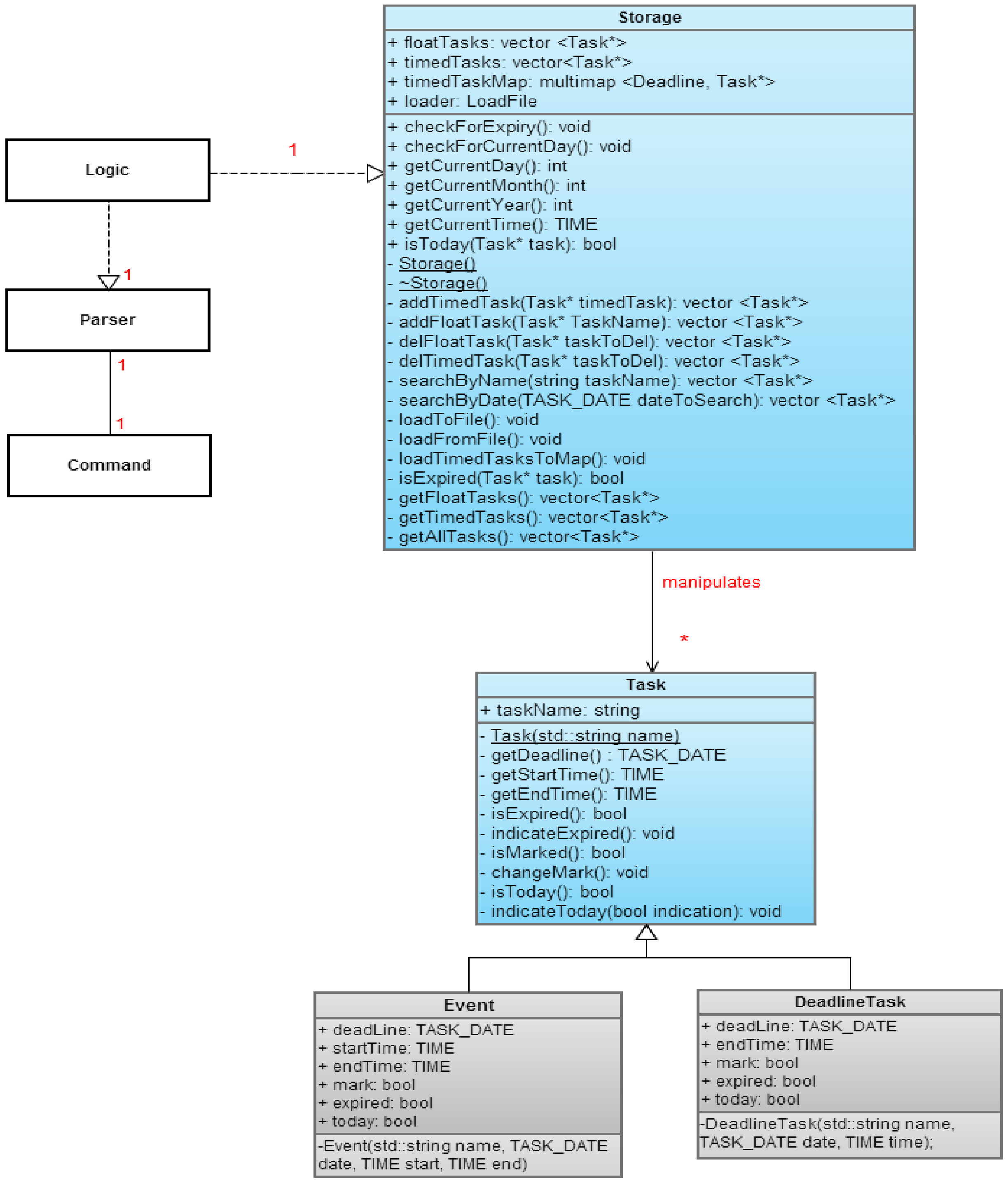
4.8 LoadFile Class

LoadFile class is able to conduct file loading so as to allow the saving and loading of tasks from a textfile. The loadfile class will first load the file from text file at the start of the program so as to display it in the GUI and it will then input everything the user has saved into the text file again after he has exited the program.


LoadFile
+ textFile: fstream
+ FLOAT_FILE_NAME: static const string
+ EVENT_FILE_NAME: static const string
- loadFloatTasks(): vector<Task*>
- loadTimedTasks(): vector<Task*>
- inputFloatTasksToFile(vector<Task*> &tasks): void
- inputTimedTasksToFile(vector<Task*> &tasks): void

5. Class Association Diagram

This is an example of how different classes interact with one another.


Logic
Parser
Command
1
1
1
1
manipulates
*
Storage
+ floatTasks: vector <Task*>
+ timedTasks: vector<Task*>
+ timedTaskMap: multimap <Deadline, Task*>
+ loader: LoadFile
+ checkForExpiry(): void
+ checkForCurrentDay(): void
+ getCurrentDay(): int
+ getCurrentMonth(): int
+ getCurrentYear(): int
+ getCurrentTime(): TIME
+ isToday(Task* task): bool
- Storage()
- ~Storage()
- addTimedTask(Task* timedTask): vector <Task*>
- addFloatTask(Task* TaskName): vector <Task*>
- delFloatTask(Task* taskToDel): vector <Task*>
- delTimedTask(Task* taskToDel): vector <Task*>
- searchByName(string taskName): vector <Task*>
- searchByDate(TASK_DATE dateToSearch): vector <Task*>
- loadToFile(): void
- loadFromFile(): void
- loadTimedTasksToMap(): void
- isExpired(Task* task): bool
- getFloatTasks(): vector<Task*>
- getTimedTasks(): vector<Task*>
- getAllTasks(): vector<Task*>
Task
+ taskName: string
- Task(std::string name)
- getDeadline() : TASK_DATE
- getStartTime(): TIME
- getEndTime(): TIME
- isExpired(): bool
- indicateExpired(): void
- isMarked(): bool
- changeMark(): void
- isToday(): bool
- indicateToday(bool indication): void
Event
+ deadLine: TASK_DATE
+ startTime: TIME
+ endTime: TIME
+ mark: bool
+ expired: bool
+ today: bool
-Event(std::string name, TASK_DATE date, TIME start, TIME end)
DeadlineTask
+ deadLine: TASK_DATE
+ endTime: TIME
+ mark: bool
+ expired: bool
+ today: bool
-DeadlineTask(std::string name, TASK_DATE date, TIME time);

6. Application of Patterns

4.1 Patterns used

4.1.1 Abstraction Occurrence Pattern

Our group has adopted the Abstraction Occurrence Pattern for Task Class and Command Class

For Task Class, the “abstraction” class is Task, which only includes the attribute that a float task has, which is the task name. The “occurrence” class is Deadline Task, which has additional attribute like end time and date. Another “occurrence” class is Event which has additional attribute like start time, end time and date.

For Command Class, the “abstraction” class is Command. The “occurrence” classes are AddCommand, DelCommand, MarkCommand and SearchCommand

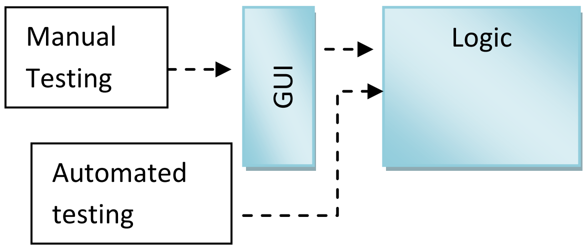
4.1.2 Facade Pattern

We have applied the Facade Pattern by creating the Facade Class called Logic, which acts as a controller between parser and storage. The Parser Class first analyses user input which then passes to Logic to execute the necessary commands. This class will be able to allow access from GUI without internal details of Storage.

4.1.3 Command Pattern

We have applied the Command pattern for command add, command delete and command search. By making the Command class an abstract class and the rest of the classes like AddCommand, DeleteCommand, SearchCommand and MarkCommand as inherited classes. All these commands are treated as a general Command type. Parser Class then issues a request to execute the required command.

7. Testing



Due to the presence of GUI, all product-level testing needs to be done using the GUI. However, GUI testing is much harder than CLI (command line interface) testing or API testing. To overcome the challenge of testing GUIs, we have minimized logic aspects in the GUI. Thus, we bypass the GUI to test the rest of the system using automated API testing. While this still requires the GUI to be tested manually, we can reduce the number of such manual test cases because most of the system has already been tested using automated API testing.

Resources

Gliffy for all the UML diagrams

http://balazshuszar.com/ for stickman figure for logo