





Kairos

Making Life simple! One task at a time!

USER GUIDE

			
Nguyen Hoang Thanh Duc, David	Hari Krishna Vetharenian	Karthik Rajasekaran	Bathini Yatish
Team Leader + Task, TaskList implementation	Repo Manager + UI implementation	Architect + Logic implementation	Tester, Debugger + Controller implementation

INTRODUCTION

Kairos is a open-source tool that helps you keep track and stay ahead of all of life's happenings all in one interactive app that is both easy-to-use and packed with features. It helps you meet deadlines, finish work and lets you achieve greater productivity. With a strong ***flexi command capability***, Kairos is designed to put user experience at the forefront during task handling through the ease of Text Command Line Interface. So let Kairos help you make every second count.

Let's get started.

Autostart - Boot up your system and Kairos is there waiting for you to start the tasks for the day.

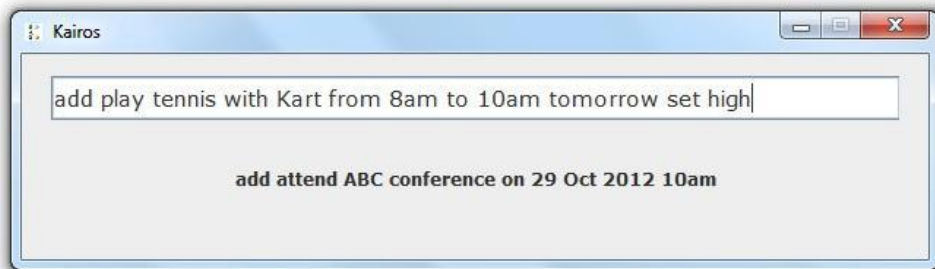
Shortcut – Hit **WINDOWS+A** at any point to minimise or maximise your Kairos application from the notification tray

FEATURES

1. Have some important deadline?

Just type in what you would normally write down and voila! Kairos understands and updates your task list. Kairos have an exhaustive add command which is capable of handling all events possible with a few short keywords. It's as intuitive as it can get.

Intelligent - Auto complete helps you save typing a few more keystrokes. The text also is colored based on the priority of the task, highlighting to you various parts of the input. Kairos is able to understand and accept different ways to create new task. You can start with various keywords like **add, +, create, new** in any case just to add in a new task into the application.



Task with time happening on the same day

add meeting with John at 5pm

Task with time happening on tomorrow

add submit report by 10am tomorrow

Task with start and end time

add play tennis from 2pm to 6pm this Saturday

Task on specific date with time

add attend ABC conference on 29 Oct 2012 10am

Task with priority: all the tasks are default to have no priority. However, you can change the task priority by simply typing “**set high/low**” at the end of your command.

add attend CS2103 lecture on 14 Sep 2012 2pm set high (*high priority*)

add attend ECE sharing session 10 Sep 2012 set low (*low priority*)

add re-organize room (*floating tasks with no priority*)

Task with start and end date, and location

add camping at National Park from 10 Oct 2012 10am to 11 Oct 2012 5pm

Floating task

add reading The Mythical Man Month

2. There is a change in the schedule. Need to update some details of a task?

The edit command has got you covered. It's able to quickly find the task you need to modify and enable you to change its description. You are able to use various command like **edit**, **update**, **change**, **fix**, **postpone**. With the autocomplete feature, you will be able to navigate the task which you wish to change and update with the new info easily.

You can easily find the task you want to update by simply giving Kairos any keyword which can be part of the task. For example, you can give the description of the task (“meeting”, “project”), location (“NUS”, “Expo”) or any timing component (“today 7pm”, “9am 12 Nov”, “tomorrow from 9am to 11am”). Kairos will automatically search and display all the possible tasks fit your keyword.

(What do the color mean? Check out Color Decoding Session)



[Type text]

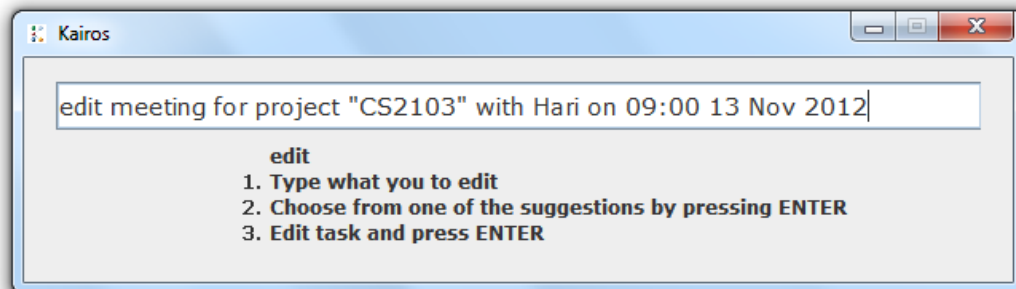
[F12-3J][V0.5]

[Type text]

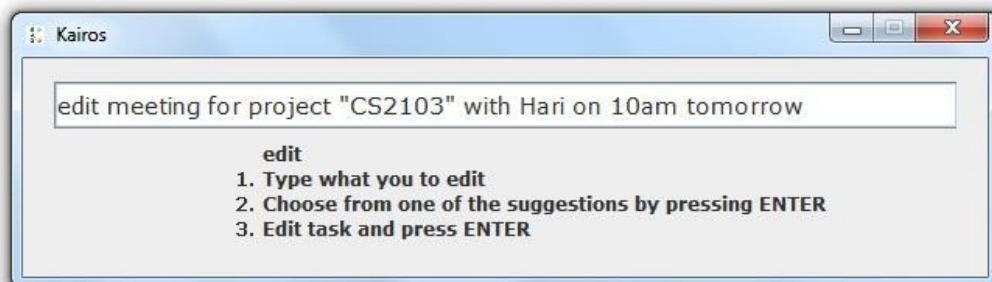
Navigate to choose the task you want to update.



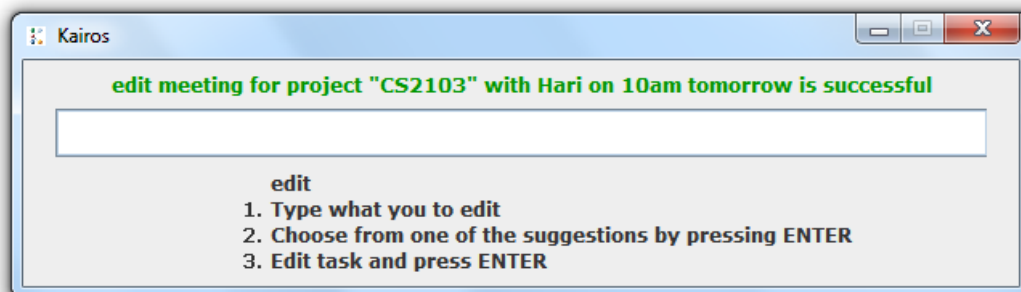
Hit enter to start editing the information of the task



Make any change you wish to the task. If you dont wish to update this task anymore, cancel the Edit command by hitting Esc.

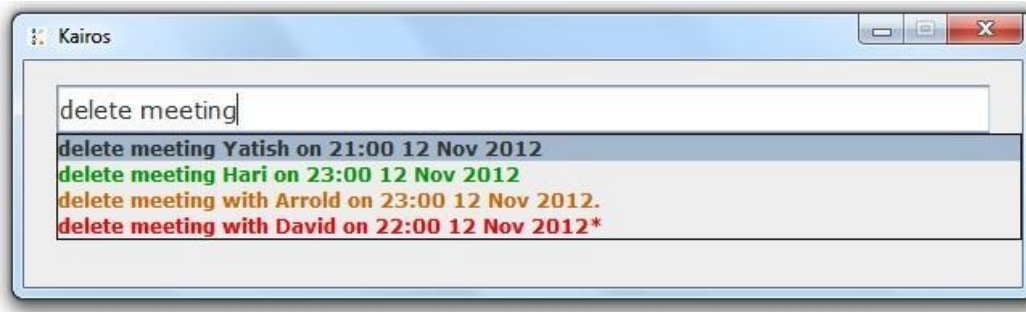


Hit Enter to confirm the change and Kairos will respond to you if the change is successfully made.



3. Your schedule entry is cancelled. Need to delete it?

Just as simple as updating task, you can give Kairos any keyword to search for the task. Simply typing any command like **delete**, **remove**, **-** (minus sign), you then can search any keyword and delete the task easily.



Navigate to the task you want to delete or cancel.



Hit enter to confirm. Kairos will respond if the task is deleted.



4. You have so many tasks now. How to view the details of them?

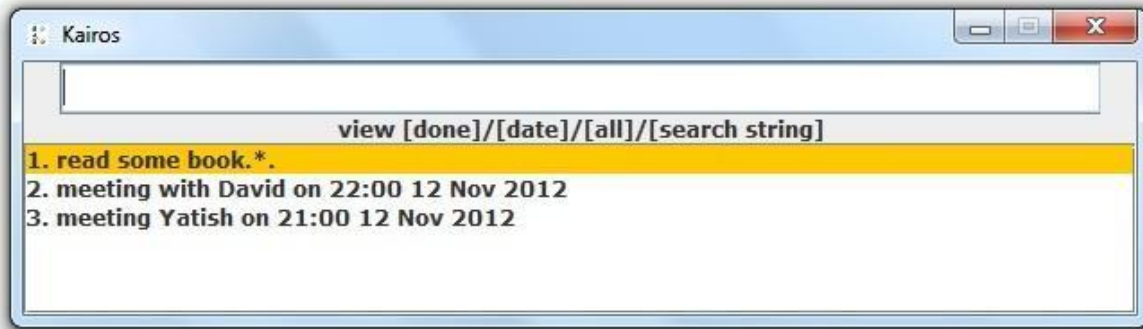
Kairos lets you to view the details of task easily without having to remember about their description. Kairos can support various command words like **search**, **view**, **find**, **display**, **show**, **see**. You can

[Type text]

[F12-3J][V0.5]

[Type text]

type any of these command words, following by your keyword. The result will be shown at the bottom of Kairos. By default, Kairos only shows the incompleted tasks.



To view all the incompleted tasks

view

view all

view undone

To view all the incompleted tasks with some keyword

view [keyword]

To view all the incompleted tasks of today

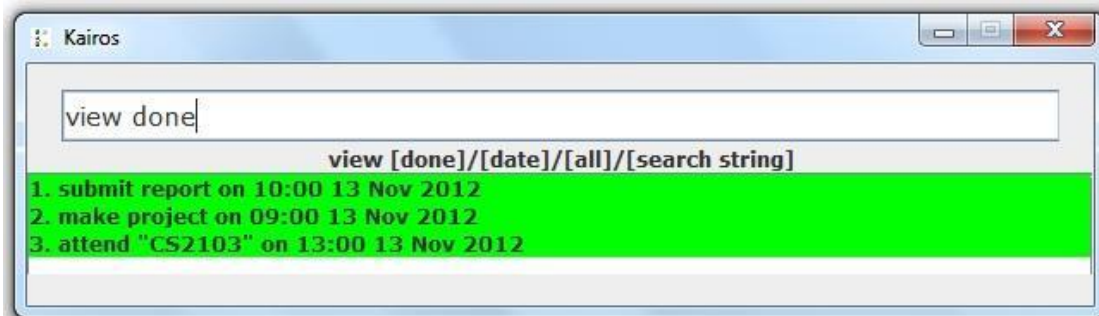
view today

To view all the incompleted tasks within a time range

view 8am 12 Nov to 10pm 14 Nov

To view all the completed tasks just enter the simple keyword

view done

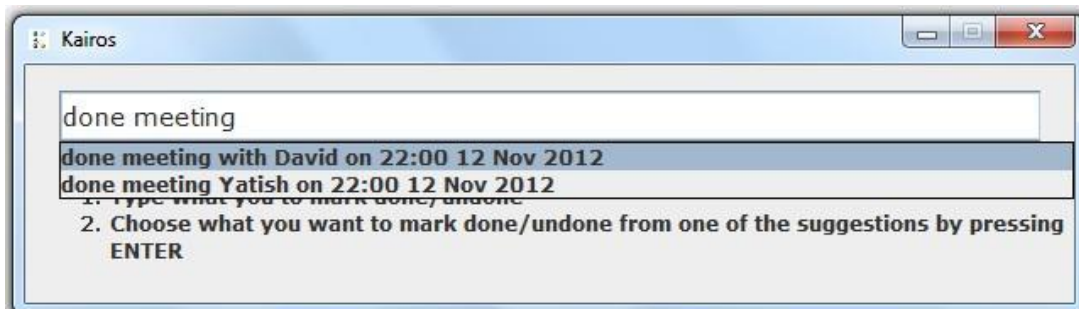


To view all the completed tasks with some keyword

view done [keyword]

5. You have finished some task. Good job. Let us mark that task as done.

Implemented with the autocomplete feature, Kairos allows you to search for any task and mark it as done easily. Simply typing any of these words: **done**, **completed**, **finished**, following by your keyword. Kairos will only show incomplected tasks associating with the keyword.



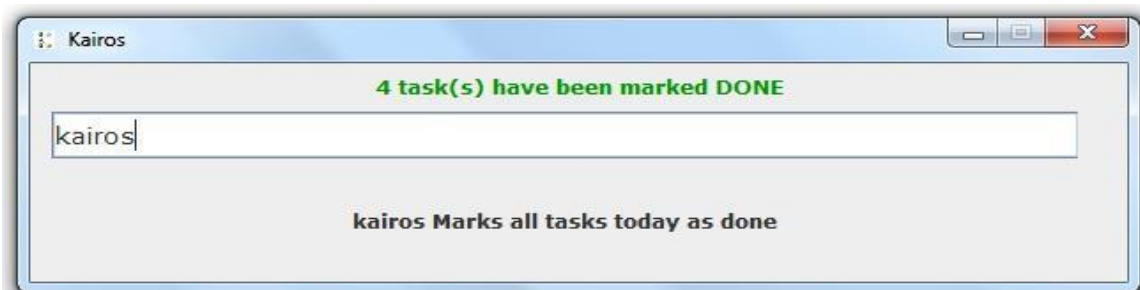
6. You thought you have done that task, but things change and you need to undone it?

Just as simple as “**done**”, to undone some task, simply type any command words like **undone**, **incomplete**, **unfinished** following by your keyword to mark a task as undone. By default, Kairos will display all the completed tasks associating with the keyword.



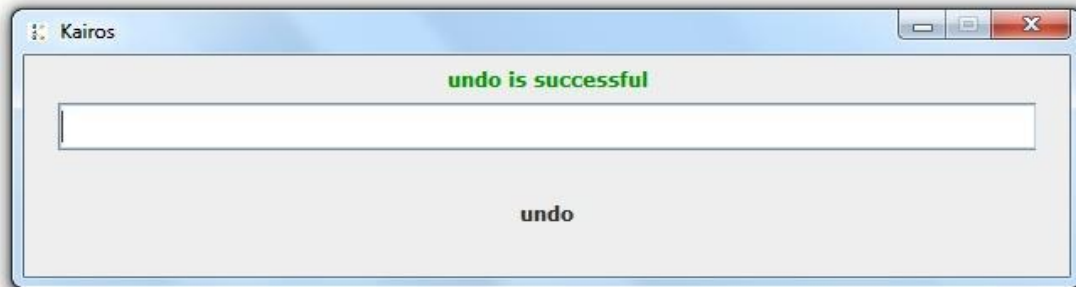
7. You have done so many tasks today. You have to mark one by one task as done?

Of course not. Kairos itself supports a feature which will mark all the today tasks to be completed by 1 command. Simply type **kairos**, it will respond to you how many tasks have been made to be done.



8. Made a mistake?

With undo, you can remove the changes you may have previously made. By simply type **undo**, Kairos will respond to you if the undo command is successful. Kairos is able to undo till the moment you first opened Kairos.



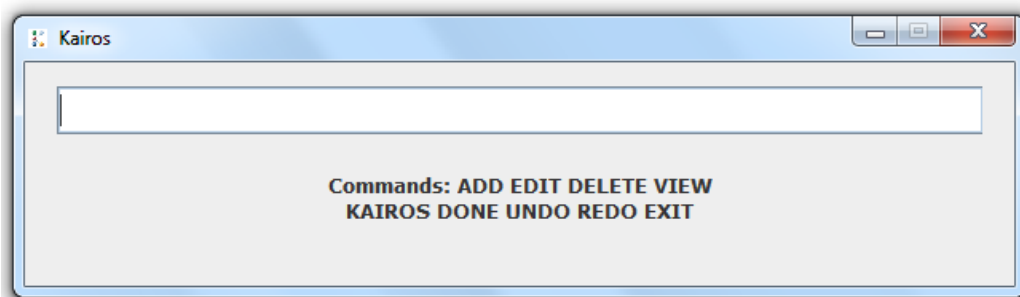
9. What if you undo by mistake? How about undo the undo?

This is basically the redo feature. If you did not do undo before, the **redo** command will fail. You can undo and redo as many times as you wish to.



10. Help!?

Stuck? Forgot syntax? Worry not because Kairos has an inbuilt help feature. Kairos helps you in your venture step by step by providing command hinting as you type your commands. These commands hinted change from time to time dynamically providing you with real time help.



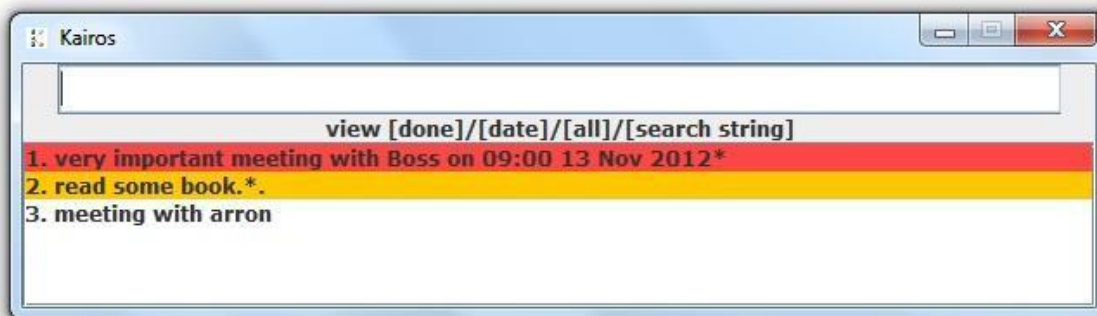
11. Ok, it is time to leave, Simply type **exit**, **quit** or **leave**.

12. One more last thing, what do all the colors mean?

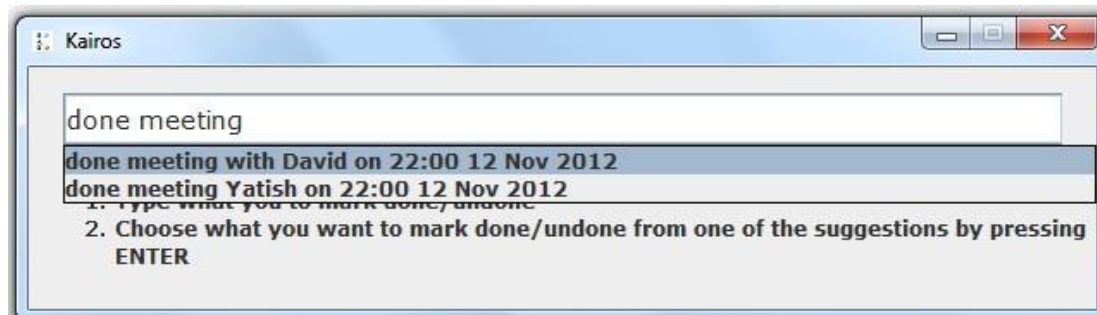
By default, any task will have no priority. Hence, in **view** or **autocomplete** feature, it will appear to be in back color.



You can set any specific task to have either high or low priority. It will then appear in red if it has high priority. It will appear in orange if it has a low priority.



If a task is marked as done, it will appear in green.



Thank you for choosing Kairos to be your daily task manager!
Make every second count.

Kairos

DEVELOPER GUIDE

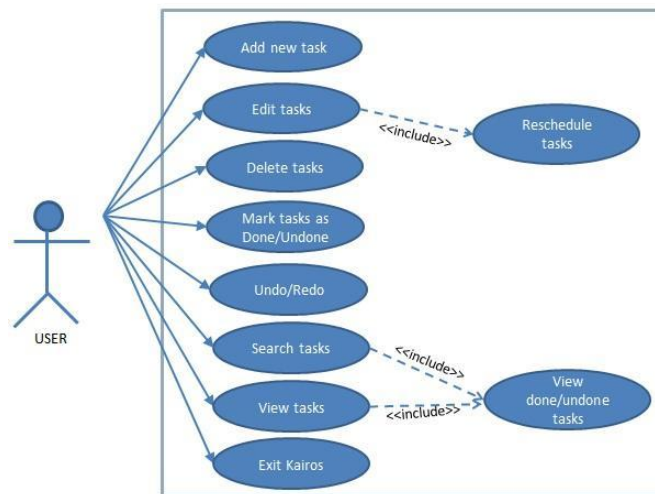
1 Design and implementation:

There are 9 main classes which are:

- Controller: plays a role as a container which will create objects of other classes and passing parameter between functions and objects.
- UI: interacts with User such as taking User command and display message to User.
- Logic: interprets the command from User. It also analyzes and provides necessary information to Kairos. It also creates a dummy Task which will contain information from user's input.
- TaskList: acts as a buffer which contains a list of Task. It also has 2 Log stack for Undo and Redo feature.
- Task: is the fundamental object of Kairos. It contains all the information of a task which User is trying to manage.
- KairosDictionary: a class which contains all the dictionary words used in Kairos. It contains the list of the alternate command types that Kairos allows.
- Storage: is the main database which will store all the task even when the application is closed or terminated. There are two files KairosTasks and Archive.
- Logging: is a common logging class to create a single logger instance for all the classes.
- SystemTrayIntegration: is to create an instance of Kairos on the windows task bar.

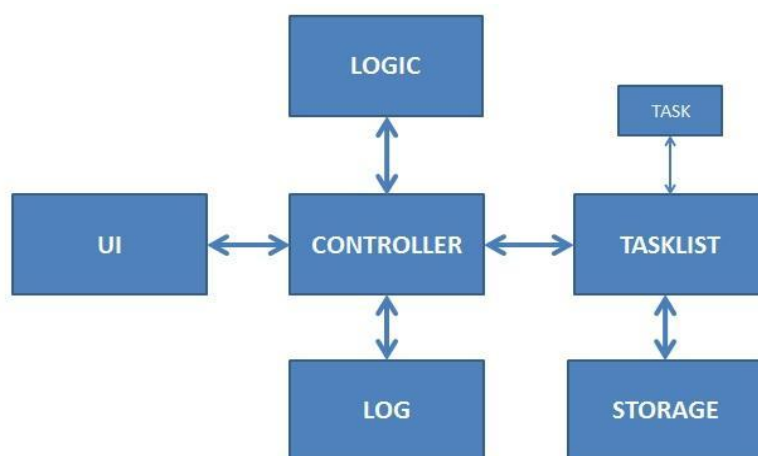
2 Descriptive Diagrams

Use Case Diagram



Users are able to add new tasks to database using UI. For editing and deleting tasks, autocomplete feature will help them to operate easily. Users can make use of this to postpone/delay a schedule entry. Kairos will search for the task and let users choose task. Users can mark the tasks as done or undone in the same way. Users also can undo the previous command and redo the following command. Users can search/view tasks following some keyword. By using this, users can view done/undone task at the same time. Finally, they can minimize as well as exit Kairos.

Architecture Diagram



The Architecture of Kairos consists of :

UI: Classes that interact directly with user, in our case with Keyboard. They pass inputs and display autocomplete search results as well as integrate with system tray and provide help messages as well. UI in Kairos consists of KairosGUI and SystemTrayIntegration as well as helper classes inside KairosGUI that help implement listeners and renderers for the elements in the GUI.

Controller: Class that contains the main, is the starting point of the program and acts as a relay between different classes that exist in the program. Provides validating functions to be used by other classes. In Kairos, Controller is implemented by its namesake.

Logic:

It is responsible for comprehending user commands. It is also responsible for creating dummytasks that are used by the TaskList for updating tasks. It also maintains a dictionary of words that can be easily expanded to be more flexible in the future. In Kairos Logic is implemented by both the Logic and KairosDictionary classes.

TaskList:

Is responsible for maintaining a temporary database of tasks and also the updation of the storage. It also contains stacks for implementing undo and redo. In Kairos, TaskList is implemented by its namesake.

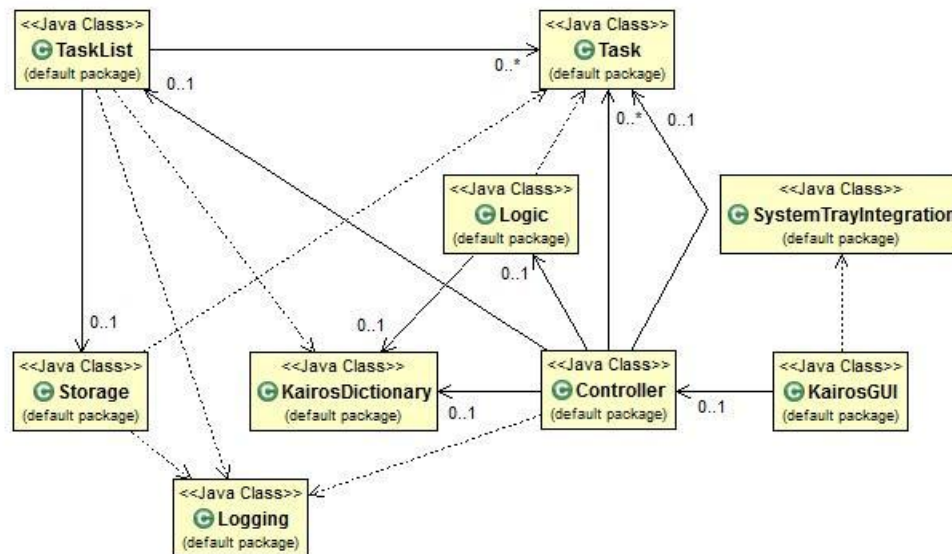
Storage:

Is responsible for updation of tasks in the tasklist onto an external file and loading of the file contents onto the tasklist. It also ensures that the program does not break if the external file is corrupted. In Kairos, it is implemented by its namesake.

Log:

Is responsible for logging all transactions between classes and exporting it to external file. In Kairos, Log is implemented by Logging class.

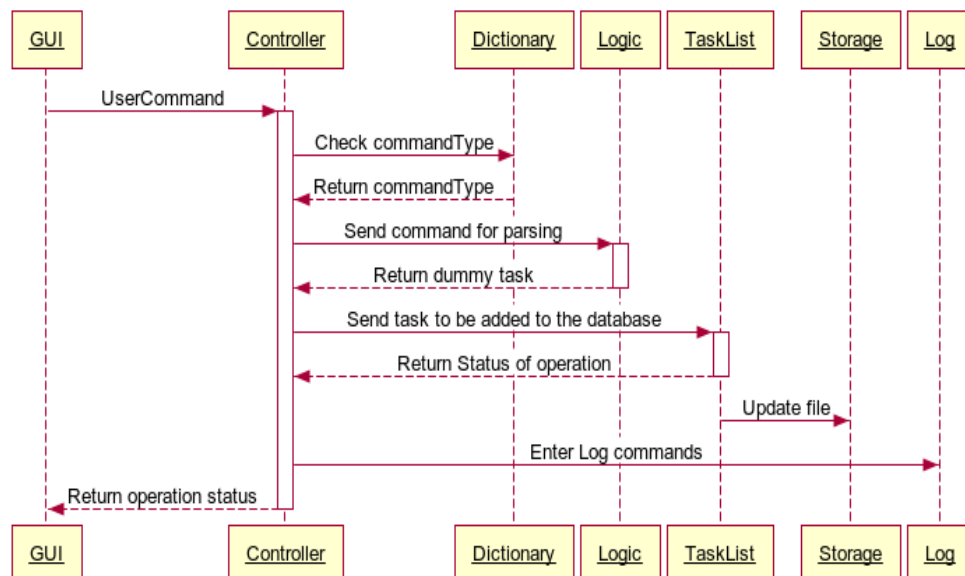
Class Diagram

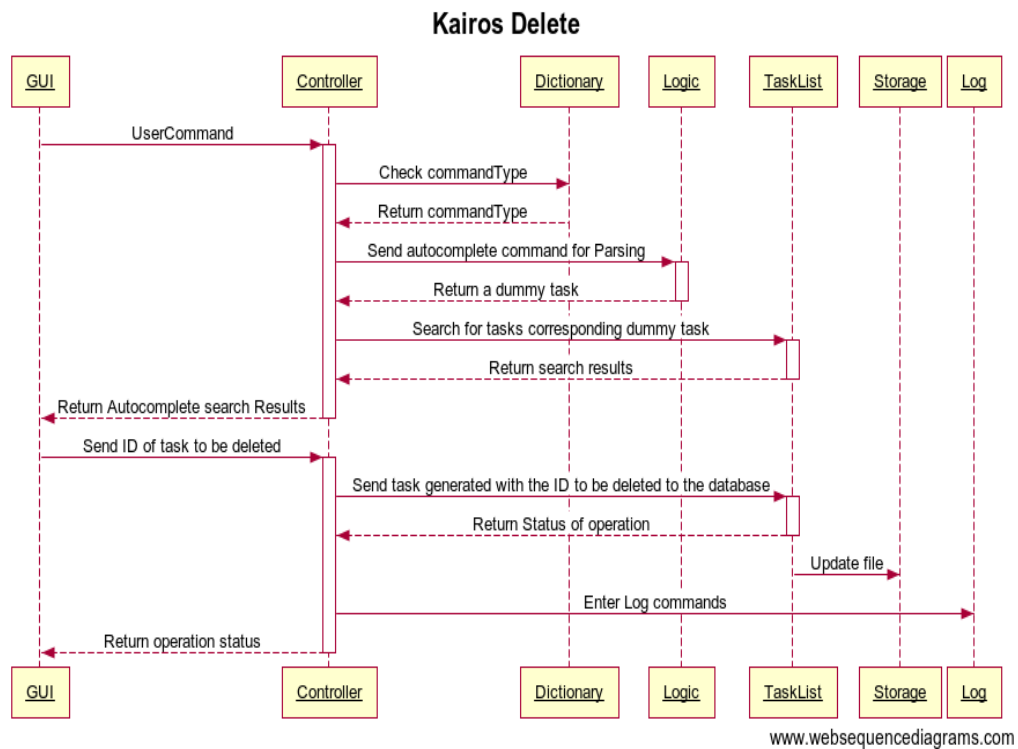
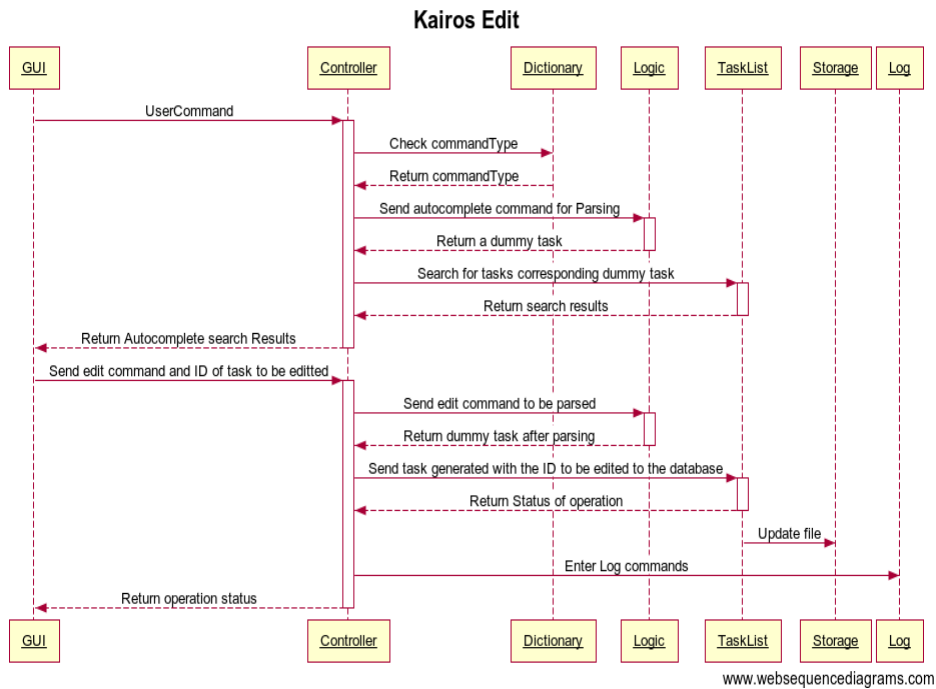


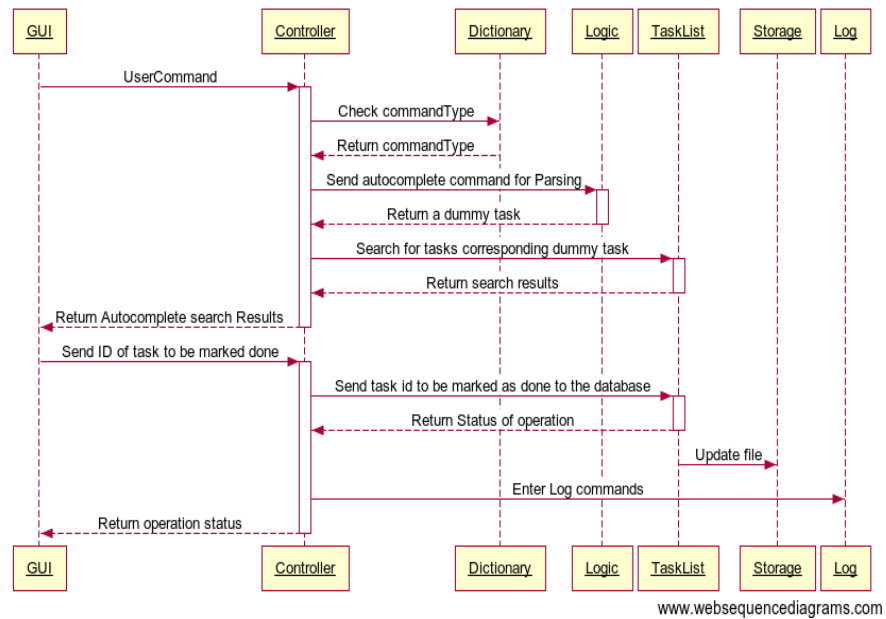
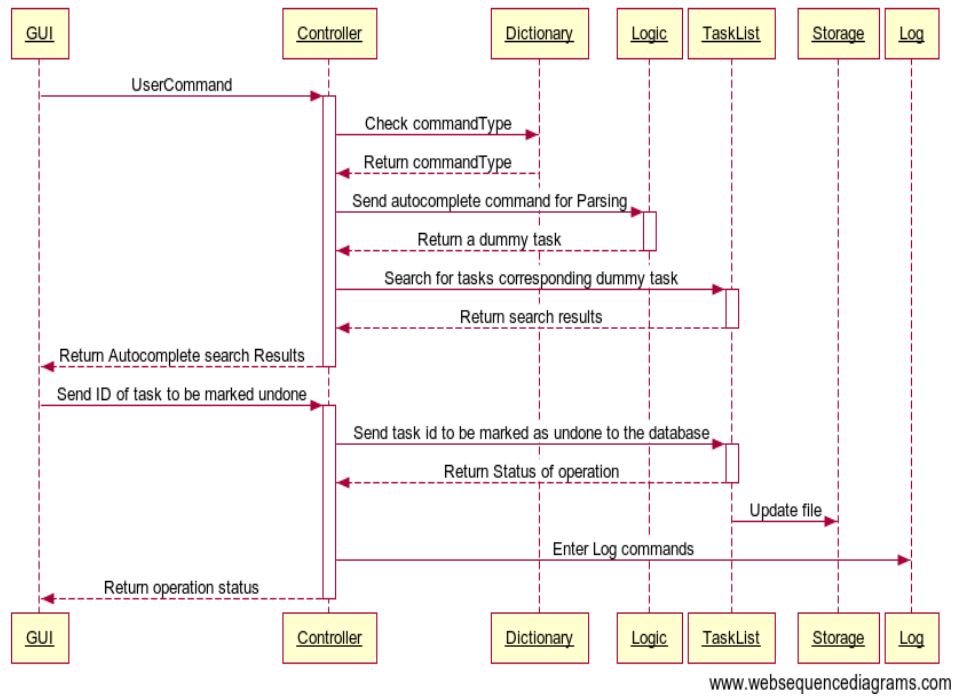
The class diagram depicts the extent of cohesion and coupling of the various classes. Coupling has been kept to a minimum in this architecture as far as it was seen fit. Each class has associations and members pertaining to that particular class, maintaining its cohesion. The facade class Controller has coupling with a single KairosGUI object, a TaskList object and a Logic class. The KairosGUI class is responsible for maintaining the SystemTrayIntegration object. There is a singleton class KairosDictionary used by Controller to instantiate the Logic object. TaskList operates with the Storage and Logging class as it is responsible for functions performed on them.

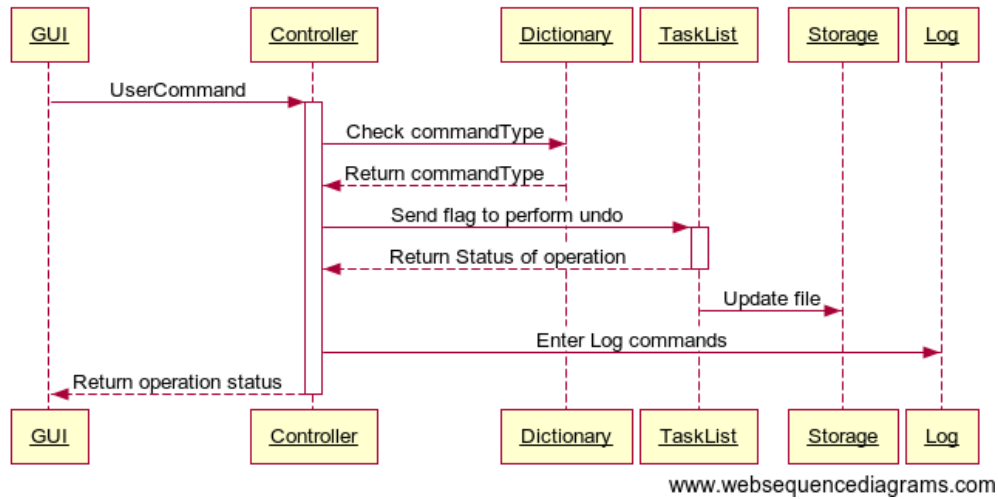
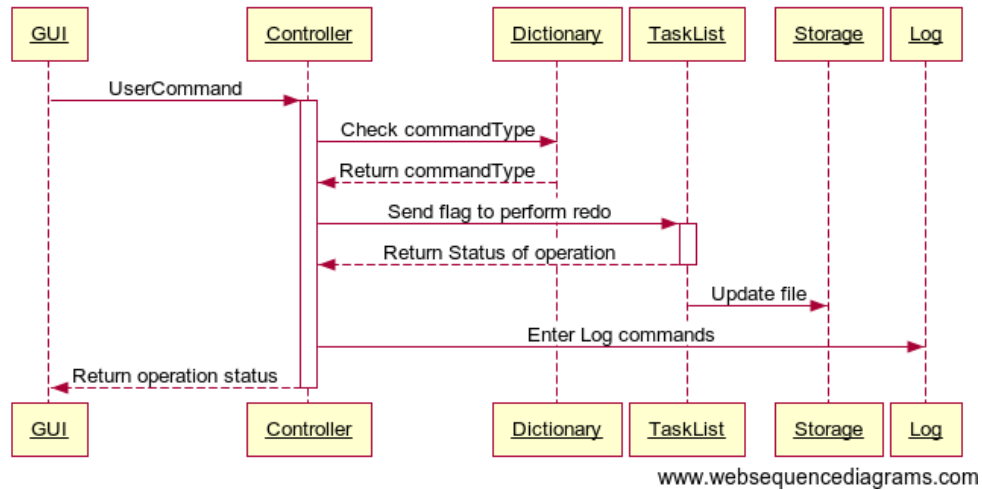
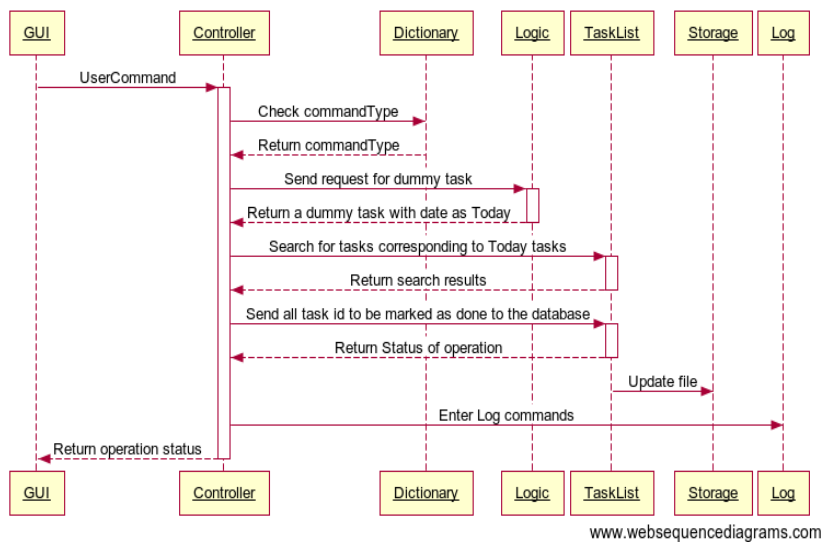
Sequence Diagrams

Kairos Add





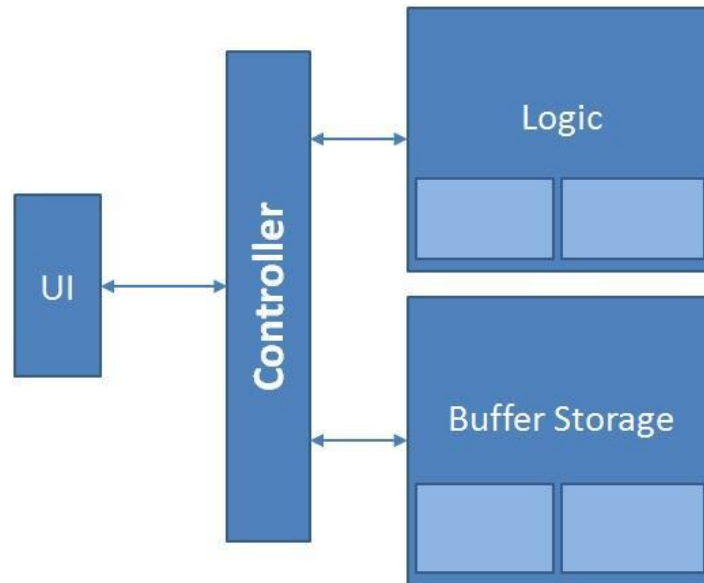
Kairos Done**Kairos Undone**

Kairos undo**Kairos Redo****Kairos kairos**

5. Principles and Patterns Used:

Facade:

The Controller class in our project acts as a Facade pattern. For example, the GUI requires to access the database for auto-complete search results, however we would not like the GUI accessing the database as it would allow the user to alter the database. In such cases, the GUI sends its requests to the Controller which searches the database using the classes Logic and TaskList and returns the search results.



6. **Important APIs:**

a *Controller:*

This is the main class of the project which initiates the application. This class acts as a facade class separating the GUI from the Logic and Storage. This class contains objects of the Logic, TaskList and uses the Task. It also invokes the KairosDictionary to verify the commands entered and performs the corresponding operations

boolean setUserCommand(String)	Function that interacts with the GUI and calls the corresponding functions to perform the operations relevant to the commands entered by the user
KairosDictionary.CommandType	Function that returns the command type that corresponds

getCommandType(String)	to the command passed by the user to it based on the existing Kairos Dictionary
boolean isAutocompleteable(String)	Checks if the command passed is a command containing a keyword that is autocompleteable. Returns true if the command type is either Edit, Delete, Done or Undone and false otherwise
boolean isAdd(String)	Checks if the command passed is a command containing a keyword that is ADD. Returns true if the command type is ADD and false otherwise
boolean isAdd(String)	Checks if the command passed is a command containing a keyword that is ADD. Returns true if the command type is ADD and false otherwise
boolean isEdit(String)	Checks if the command passed is a command containing a keyword that is EDIT. Returns true if the command type is EDIT and false otherwise
boolean isView(String)	Checks if the command passed is a command containing a keyword that is VIEW. Returns true if the command type is VIEW and false otherwise
boolean isDelete(String)	Checks if the command passed is a command containing a keyword that is DELETE. Returns true if the command type is DELETE and false otherwise
boolean isUndoOrRedo(String)	Checks if the command passed is a command containing a keyword that is Undo or Redo. Returns true if the command type is UNDO or REDO and false otherwise
boolean isDone(String)	Checks if the command passed is a command containing a keyword that is DONE. Returns true if the command type is DONE and false otherwise
boolean isKairos(String)	Checks if the command passed is a command containing a keyword that is KAIROS. Returns true if the command type is KAIROS and false otherwise
boolean isExit(String)	Checks if the command passed is a command containing a keyword that is EXIT. Returns true if the command type is EXIT and false otherwise
boolean isViewDone(String)	Checks if the command passed is a command containing a keyword that is View and if the view is for the keyword Done. Returns true if so and false otherwise
boolean getAutocompleteSearchResults(String)	Searches for other tasks in the database which match the task created upon parsing the String entered by the user.

	Returns true if search results have been found and false otherwise. Assertion: String entered is not null
Vector<Integer> getAutocompleteSearchDoneID()	Returns a vector of integers containing the indices of all tasks that have the status as COMPLETED in the searchResults array generated from getAutocompleteSearchResults()
Vector<Integer> getAutocompleteHighPriorityID()	Returns a vector of integers containing the indices of all tasks that have their priority set as high in the searchResults array generated from getAutocompleteSearchResults()
Vector<Integer> getAutocompleteLowPriorityID()	Returns a vector of integers containing the indices of all tasks that have their priority set as low in the searchResults array generated from getAutocompleteSearchResults()
boolean setUserCommand(String)	This function is used to perform the required operations for the input entered. The result of the operation is returned to the user. It also returns false if the input is invalid or no keyword could be formulated from it. Assertion: input command is not null. Exception: throws exception when an operation has ended without complete execution

b GUI:

void (Vector<Vector<String>>)	setViewList Creates the way the view list is displayed to the user. Takes the input from the user and formats this input in the form of a list throws: Exception e when there is a NullPointerException in viewListModel() or any of the input vectors.
void invokeGUI()	Sets up the GUI frame and starts it.
void displayText(String)	Function to print a feedback onto the display part of the GUI
void closeGUI()	Function that closes the GUI. throws: Exception when the GUI could not be closed due to other operations being performed

c Logic:

This class is responsible for interpreting all the commands entered by the user and it converts these commands onto a dummy task object passed to the controller. The command parsing here is supposed to be strong in nature and understands tasks entered by the user in simple language.

Task parseCommand(String command)	splits up the command into respective fields then combines them into one Task object which it then returns to the Controller
KairosDictionary.CommandType determineCommandType(String command)	Returns the CommandType of the command entered.

d Task:

The class whose objects represent each single task input by user. Each object of this class contains various information like title, location, the person with whom user may want to do this task, start time, end time. Moreover, each object also has a unique taskID, taskStatus (COMPLETE, INCOMPLETE) and taskPriority (HIGH, LOW, NULL).

public Task(String title, Date start, Date end, String location, String withSomeone, Priority priority)	Constructor for a Task object
void setTaskId(int id);	Assign ID of a task. Each task has a unique non-negative ID. This must be managed by the class which creates tasks. <u>Assertion:</u> when ID is negative, assert with message "ID must be non-negative".
void setTaskTitle(String title);	Assign Title of a task. This title will give a short description about the task. This field should not be empty.
void setTaskStartTime(Date startTime);	Assign start time of a task. This will indicate the start time of a task. When this is null, it implies that the task is not a time task (task with start time and end time).
void setTaskEndTime(Date endTime);	Assign end time of a task. This indicates the end time of a task. If this is null, it implies that the task is a floating task (task with no timing component).
void setTaskLocation(String location);	Assign location details of a task. It briefly describes the location where user wants to do this

	task. This can be empty.
void setTaskStatus(int status);	Update status of a task. This indicates the status of a specific task. There are 2 possible status: COMPLETE and INCOMPLETE. By default, when a task is created, its status is INCOMPLETE.
void setTaskPriority(int priority);	Assign priority of a task. This indicates level of importance of a task. This will be useful to remind user to do high priority tasks.
int getTaskId();	Return the unique non-negative ID of a task
String getTaskTitle();	Return the title of a task
Date getTaskStartTime();	Return the start time of a task
Date getTaskEndTime();	Return the end time of a task
String getTaskLocation();	Return the location of a task
int getTaskStatus();	Return the status of a task
int getTaskPriority();	Return the priority of a task
String getPersonAccompany();	Return the personAccompany of a task
String toString();	Return a string with all data of a task collated together
boolean isMatchedKeyword(String keyword)	Check if a task contains the input keyword. The keyword must not be null. If the length of the keyword is smaller than 3, this will only return those tasks starting with the keyword. If the length is longer than 3, this will return all the tasks which contains the keyword. <u>Assertion:</u> when the keyword is NULL, assert with message "Input string is null".
boolean isMatchedDate(Date date)	check if the timing info of a task is matched the input date. If the task does not have timing component, this will return false.
boolean isSameTask(Task task)	check if the input task has any same information as this object. This will check if the object contains any element in any field of the input task. This will also check the object has any timing element same as the

	input task. The input task must not be NULL. Unless, the assertion will be thrown. <u>Assertion:</u> when the input task is NULL, assert with message “Input task is null”.
boolean exactComparison(Task task)	check if the input task has exactly title, location, person accompany, start time, end time as the object. This will return FALSE if the input task is NULL or any field of the input task is different from the corresponding field of the object, else return TRUE.

e . *TaskList*:

This is a main data buffer for Kairos. It is the only contact point to the database. TaskList will retrieve data from Storage class every time Kairos starts and save data to Storage after every command successfully runs. It contains an ArrayList of tasks, an undoLog stack, an redoLog stack and a Storage object.

- Log class: the class which will record all the successful action on main data buffer. It has 3 object: previous task, command type and updated task.
- undoLog stack: a stack of Log objects which will keep records all action of main data buffer. Whenever an action has been successfully done, a new Log object is created and added into this undoLog stack.
- redoLog stack: a stack of Log objects which will be empty most of the time. Whenever an “undo” command is performed, the current Log object at the top of undoLog will be popped and pushed to redoLog. The reverse process is performed when a “redo” command is received from user.

boolean addTask(Task task);	Assign an ID to that task and add it into the ArrayList. If the same task is already inside the ArrayList, it will not be added. This task must not be null. <u>Assertion:</u> if the task is null, assert with message: “Task is null”
bool deleteTask(Task task);	Search if the task is inside the ArrayList and delete it from main buffer.
bool deleteTask(int taskId); (overloading)	Search a task by its ID and then delete it from the main buffer.
ArrayList<Task> searchTask(String keyword)	Search all the tasks which contains the keyword.

ArrayList<Task> searchDoneTasks(Task inputTask)	Search all the done tasks by some requirement set by inputTask
ArrayList<Task> searchUndoneTasks(Task inputTask)	Search all the undone tasks by some requirement set by inputTask
public ArrayList<Task> searchTask(Task targetTask)	search all tasks by some requirement set by inputTask
boolean editTask(Task updatedTask)	Search for a task which ID is the same as the updatedTask and replaced it by the updatedTask
int kairos(Task task)	Set all the tasks of the current date to be done and return the number of tasks which were changed.
boolean undo()	Undo the previous action on main buffer. It returns true if successful, else return false
boolean redo()	Redo the current action on main buffer. It returns true if successful, else returns false.
boolean isTaskListEmpty()	Check if the main buffer is empty.
String toString()	Collate all the tasks info and combine to a single string.
boolean isClashing(Task dummyTask)	Check if the dummyTask is clashing with any task in the main buffer
boolean writeToDataBase()	Try to update all the tasks into the main data buffer

f. *Storage:*

This class creates the files and the database which store the tasks in an XML format. Two files are created by the class called KairosTasks.xml and Archive.xml. KairosTasks contains all events upto one week old. Tasks that are older than one week are pushed onto Archive.xml and are cleared out when the tasks get more than one month old.

void writeToFile(ArrayList<Task>)	Writes the tasks from the arraylist what are not more than one week old onto an xml file.
void writeToArchive(ArrayList<task>)	Writes tasks that are more than one week old but less than one month old onto an xml file
ArrayList<Task> readFromFile()	Returns an arraylist of task objects that are stored in the file
ArrayList<Task> readFromArchive()	Returns an arraylist of all task objects in the file that are not more than one month old.

g. KairosDictionary:

This class contains the list of all the keywords supported by the application. This allows the flexibility in the commands entered by the user. Commands like 'add' can be alternatively entered as '+', 'create' or 'new'

CommandType getWord(String)	Returns the Dictionary equivalent word of the entered String
boolean isAutocompleteable(String)	Checks if the command passed is a command containing a keyword that is autocompleteable. Returns true if the command type is either Edit, Delete, Done or Undone and false otherwise. <u>Assertion:</u> User input is not null
boolean isAdd(String)	Checks if the command passed is a command containing a keyword that is ADD. Returns true if the command type is ADD and false otherwise <u>Assertion:</u> User input is not null
boolean isAdd(String)	Checks if the command passed is a command containing a keyword that is ADD. Returns true if the command type is ADD and false otherwise <u>Assertion:</u> User input is not null
boolean isEdit(String)	Checks if the command passed is a command containing a keyword that is EDIT. Returns true if the command type is EDIT and false otherwise <u>Assertion:</u> User input is not null
boolean isView(String)	Checks if the command passed is a command containing a keyword that is VIEW. Returns true if the command type is VIEW and false otherwise <u>Assertion:</u> User input is not null
boolean isDelete(String)	Checks if the command passed is a command containing a keyword that is DELETE. Returns true if the command type is DELETE and false otherwise <u>Assertion:</u> User input is not null
boolean isUndoOrRedo(String)	Checks if the command passed is a command containing a keyword that is Undo or Redo. Returns true if the command type is UNDO or REDO and false otherwise <u>Assertion:</u> User input is not null
boolean isDone(String)	Checks if the command passed is a command

	containing a keyword that is DONE. Returns true if the command type is DONE and false otherwise <u>Assertion:</u> User input is not null
boolean isKairos(String)	Checks if the command passed is a command containing a keyword that is KAIROS. Returns true if the command type is KAIROS and false otherwise <u>Assertion:</u> User input is not null
boolean isExit(String)	Checks if the command passed is a command containing a keyword that is EXIT. Returns true if the command type is EXIT and false otherwise <u>Assertion:</u> User input is not null

h. *Logging:*

This is a common logging class to create a single logger instance for all the classes. This logger returned by this class is used by all the classes for logging onto a single log file which makes it easy to debug any bugs in the application.

Logger getInstance()	Gets the instance of the logger to be used in all classes to maintain a single log file
----------------------	---

External Libraries:

Natty:

Used for natural language time-date parsing in this application

“Natty is a natural language date parser written in Java. Given a date expression, natty will apply standard language recognition and translation techniques to produce a list of corresponding dates with optional parse and syntax information.”

Developed by Joe Stelmach and his team

<https://github.com/joestelmach/natty>

Jintellitype:

Used for creating global hotkeys in the application.

“[JIntellitype](#) is a Java API for interacting with Microsoft Intellitype commands as well as registering for Global Hotkeys in your Java application. The API is a Java JNI library that uses a C++ DLL to do all the communication with Windows.”

<http://code.google.com/p/jintellitype/>

Tools Utilised:

- Eclipse
- Mercurial and TortoiseHg
- Google Code
- JUnit
- Object Aid UML Diagram
- websequencediagrams.com

Practises

Followed

Top Down Design, Agile Development , Pair Programming for certain classes

Create a local copy of our code : <https://code.google.com/p/cs2103aug12-f12-3j>