



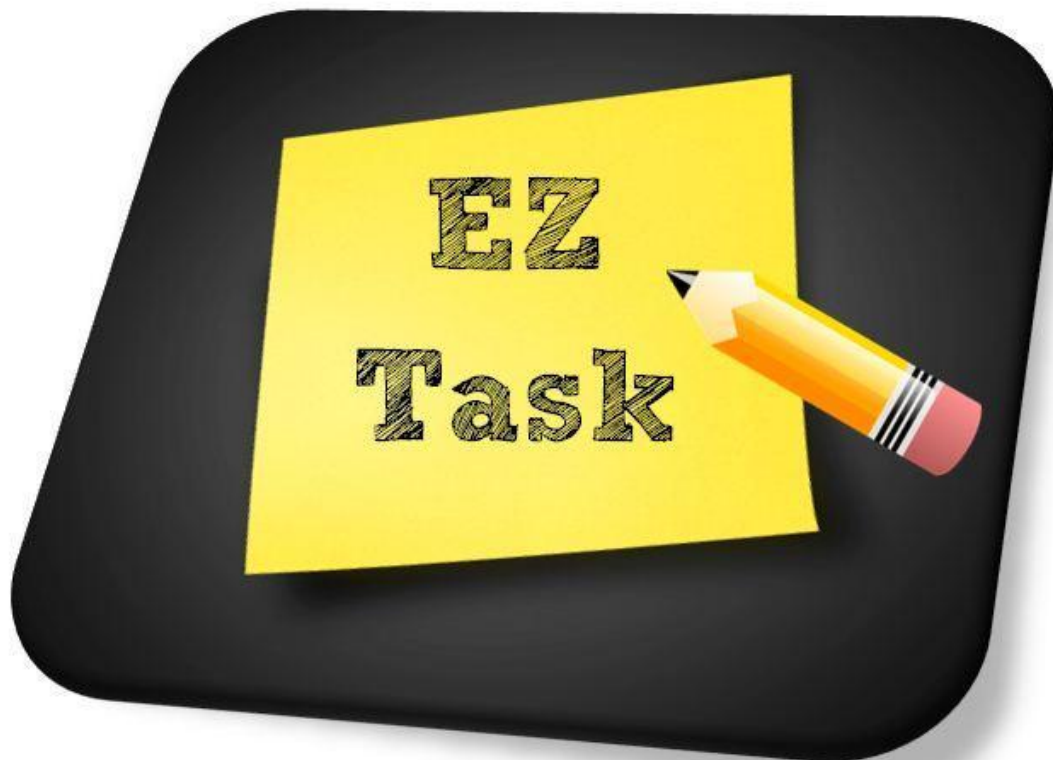


EZTask

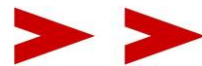
 <p>Raghav Ramesh Team lead, GUI, Testing,Parser</p>	 <p>Rajalakshmi Ramachandran Database Management(Storage),Logger, Documentation(manuals)</p>	 <p>Erik Bodin Architecture,TaskOrganizer,Logic</p>	 <p>Santosh Vadivelu Logger,video,Parser</p>
---	---	---	---





Too many tasks? Too much to handle on your own?

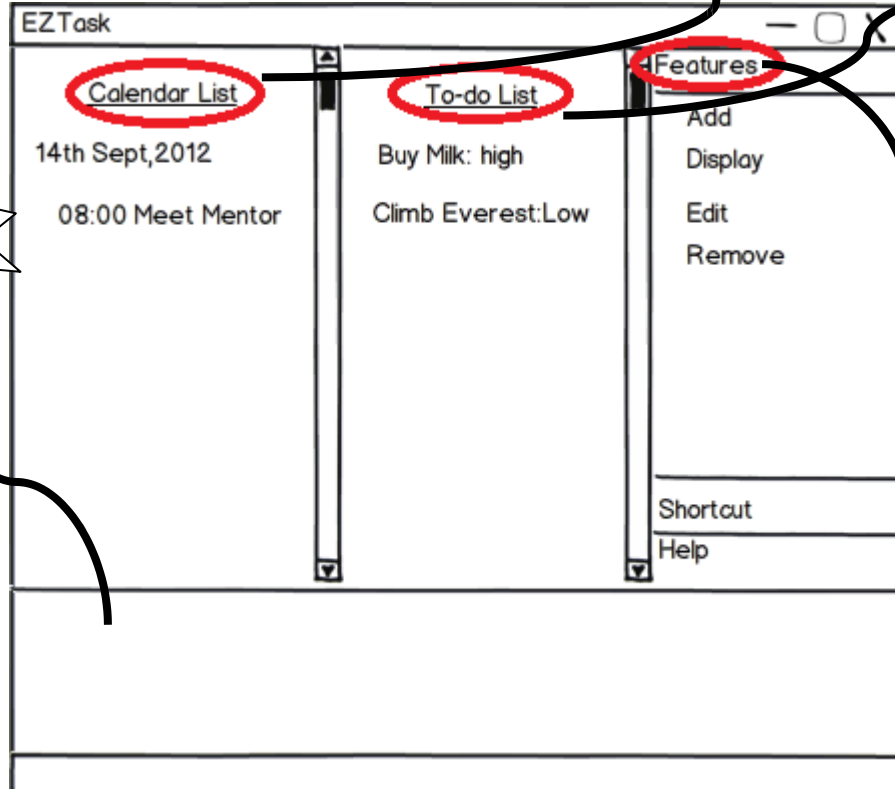
Add "We offer various facilities"
Delete "To handle tasks in day to day life"
Display "without internet being a necessity"
Update "Using our software"



edit	add	remove	lay
display	update	ete	edit
remove	update	delete	
edit	display	add	update
delete	remove	add	

COMMANDS - the keywords that help in adding, deleting or updating tasks.

NEW EXCITING FEATURES



Lists

Features Window

UNDO

LOG

How to use EZTask?

The following table tell you how to use EZTask.

Keyword	Format
Add	<ul style="list-style-type: none">• add <task>;<date>;<time>• add <task>;<date>;<start-time>:<end-time>• add <task> ;<priority>
Delete	<ul style="list-style-type: none">• delete <index>*
Edit	<ul style="list-style-type: none">• Edit <index> ;<changed/same task description>;<changed/same date>;<changed/same time>• Edit <index>;<changed/same task description>;<changed/same priority>
Search	<ul style="list-style-type: none">• Search <substring>• Search <substring> (+/*) <substring>
Complete	<ul style="list-style-type: none">• Complete c <index>• Complete p <index>
Goto	<ul style="list-style-type: none">• Goto p <priority>• Goto c<date>• Goto c today
Clear	<ul style="list-style-type: none">• Clear
Undo	<ul style="list-style-type: none">• Undo

*index- Each task has an index associated with it which is displayed in the respective lists. This index is used in remove and update.

ADD:

There are three types of tasks: deadline tasks(fixed time to end by), timed tasks(having start and end time) and floating tasks(open-ended tasks). The deadline and timed tasks are added to calendar list whereas floating tasks are added to priority List.

SEARCH:

Search can be done using substring of a string. The '+' symbol between two substrings is to search for either one of the substrings whereas '*' symbol between two substrings is to search for a task with both substrings in it.

COMPLETE:

Complete helps in marking a task as completed or done. The c or p in the syntax refers to calendar or priority list respectively. It helps in marking the task in that particular list as completed.

GOTO:

Goto displays the events that the user wants to view. It can display events of a particular date or priority. While displaying based on a particular date, it displays all the events following the date."Goto c today" displays all events from today.

CLEAR:

Clear clears all the completed tasks.

What are the features of EZTask?

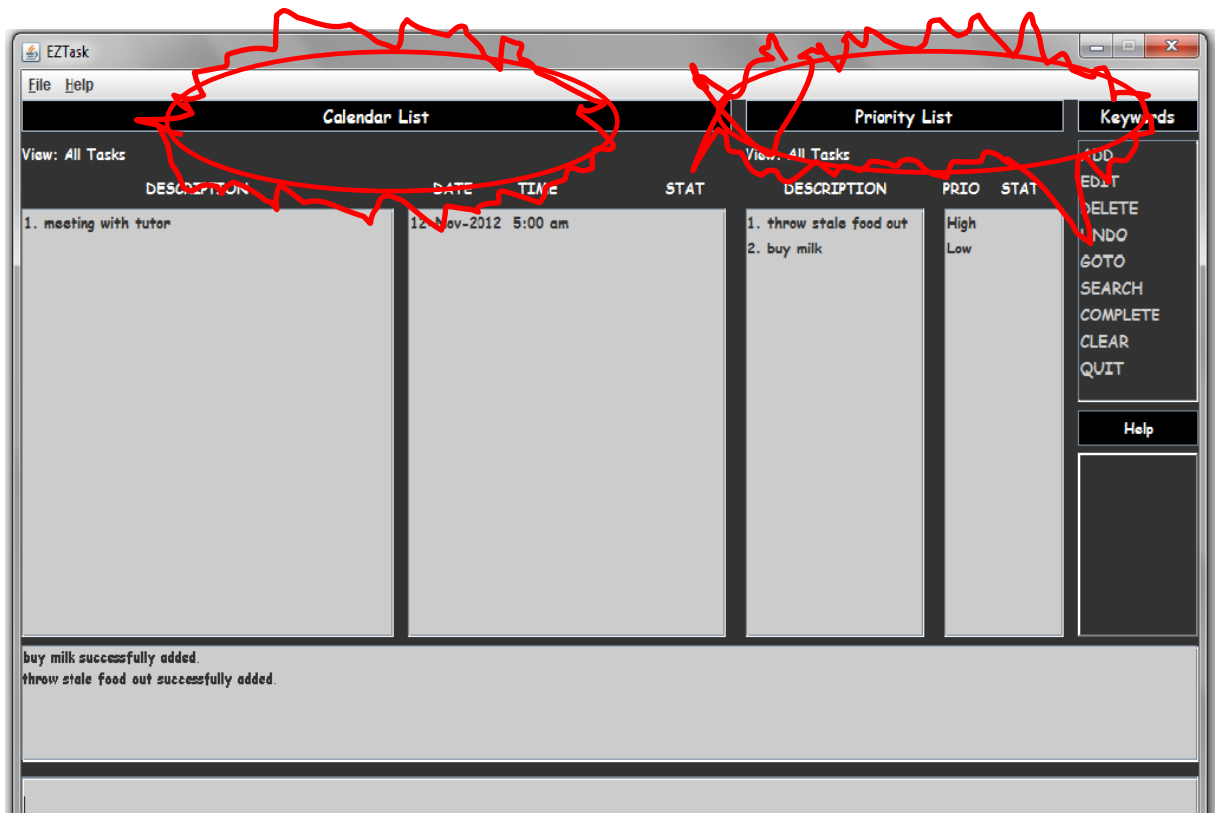
1. Calendar and Priority lists:

EZTask allows the user to add event for a particular date and time(Calendar) or an event with indefinite time or no deadline(Priority).

Calendar list displays all the events entered by the user in an orderly function. All events are displayed in an increasing order based on the date/time slot. The deadline and times tasks are displayed in this list.

Priority list displays the events that do not have a deadline and are to be completed whenever the user prefers to. All floating tasks are displayed in this list.

We can prioritise the Priority List events on the basis of the importance they hold. There are three levels of priorities: low, medium and high represented by 1, 2 and 3. The tasks are sorted based on their priority in descending order with the highest priority tasks at the top.



2. Flexible commands:

All the commands have a flexible feature whereby the command is performed with any of the substrings of the command keyword.

Special flexi features:

ADD:

Date can have any of the following formats:

- Yyyymmdd
- Now/today
- Dd <substring of month in string> <year> (in any order)

Time can have:

- Now
- Hhmm
- Hh (am/pm)

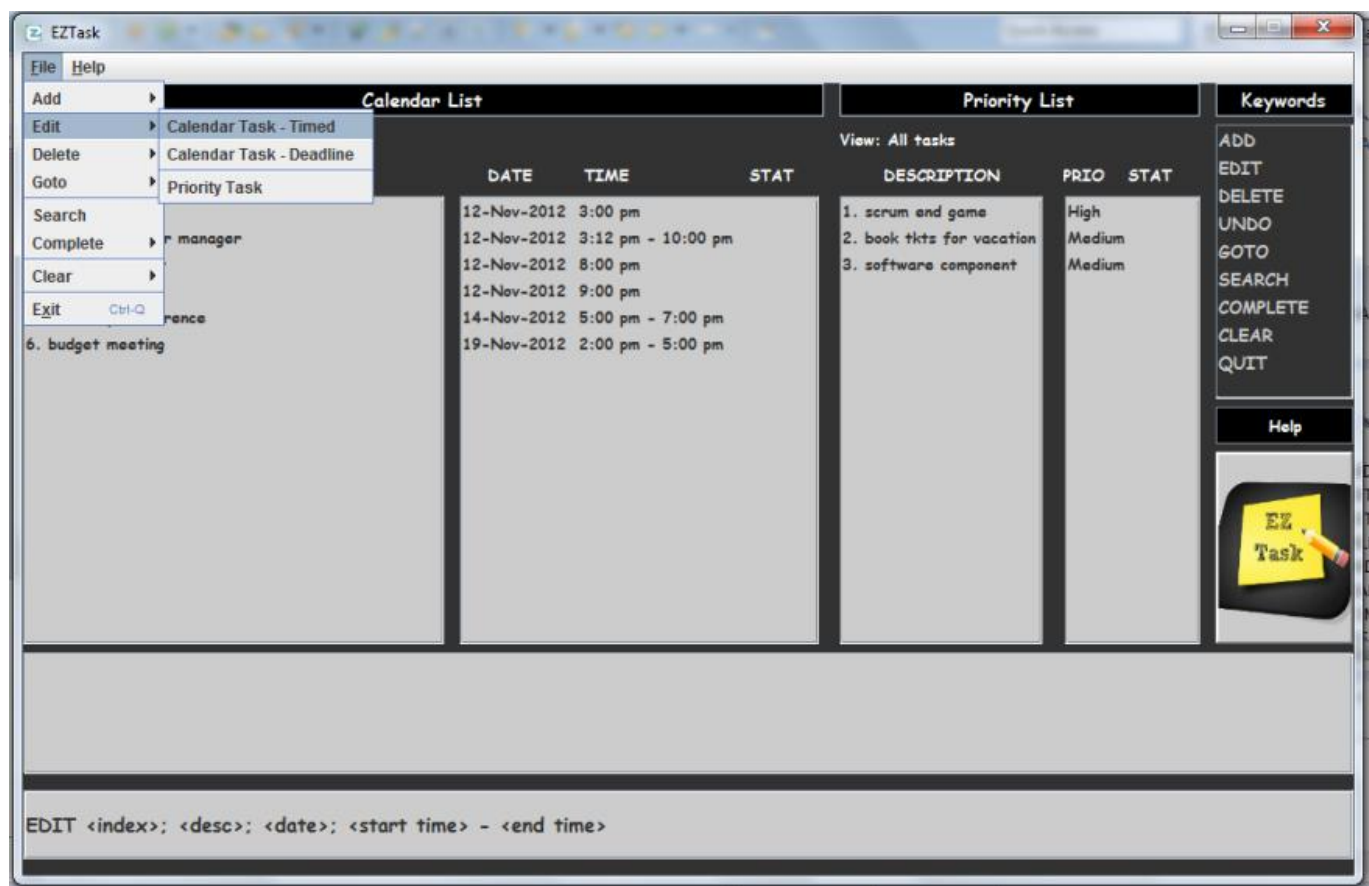
Time follows a 24-hour clock. But is displayed in the 12-hour format.

3. Feature Window/File menu:

EZTask has a features window which gives details on the formats for each of the commands. This acts like a tutorial for a first time user before he gets accustomed to the format of the commands.

The user can also use the help menu/button for further aid. which will lead to our user/developer guide.

The file menu has a number of sub-menus for all the commands which will automatically display the syntax of the command.



4. Log:

Stores the commands entered by the user in past transactions for the five most recent ones and displays it in a text area which has a scroll pane enabled to view the past transactions.

DEVELOPER'S GUIDE

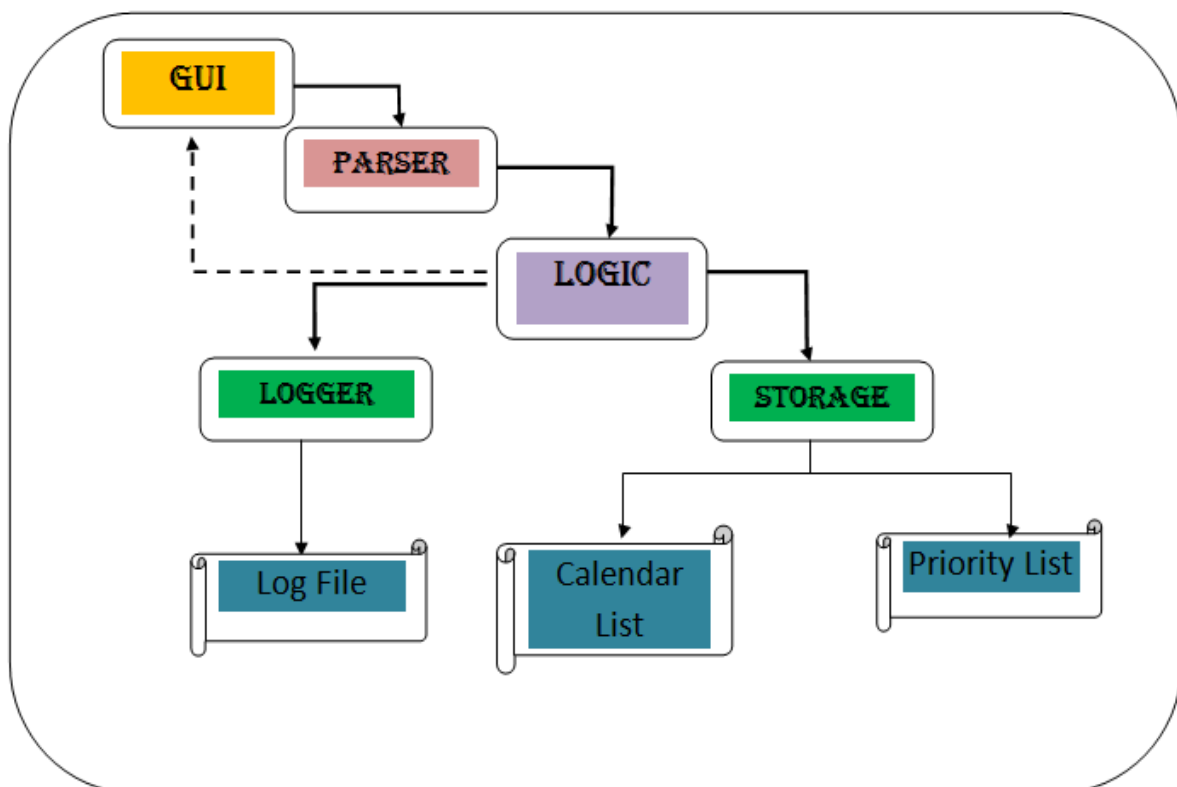
The four most important components of the program are the GUI, parser, logic and storage components. The TaskLogic and TaskOrganiser fall under the logic category and the logger is part of the storage component.

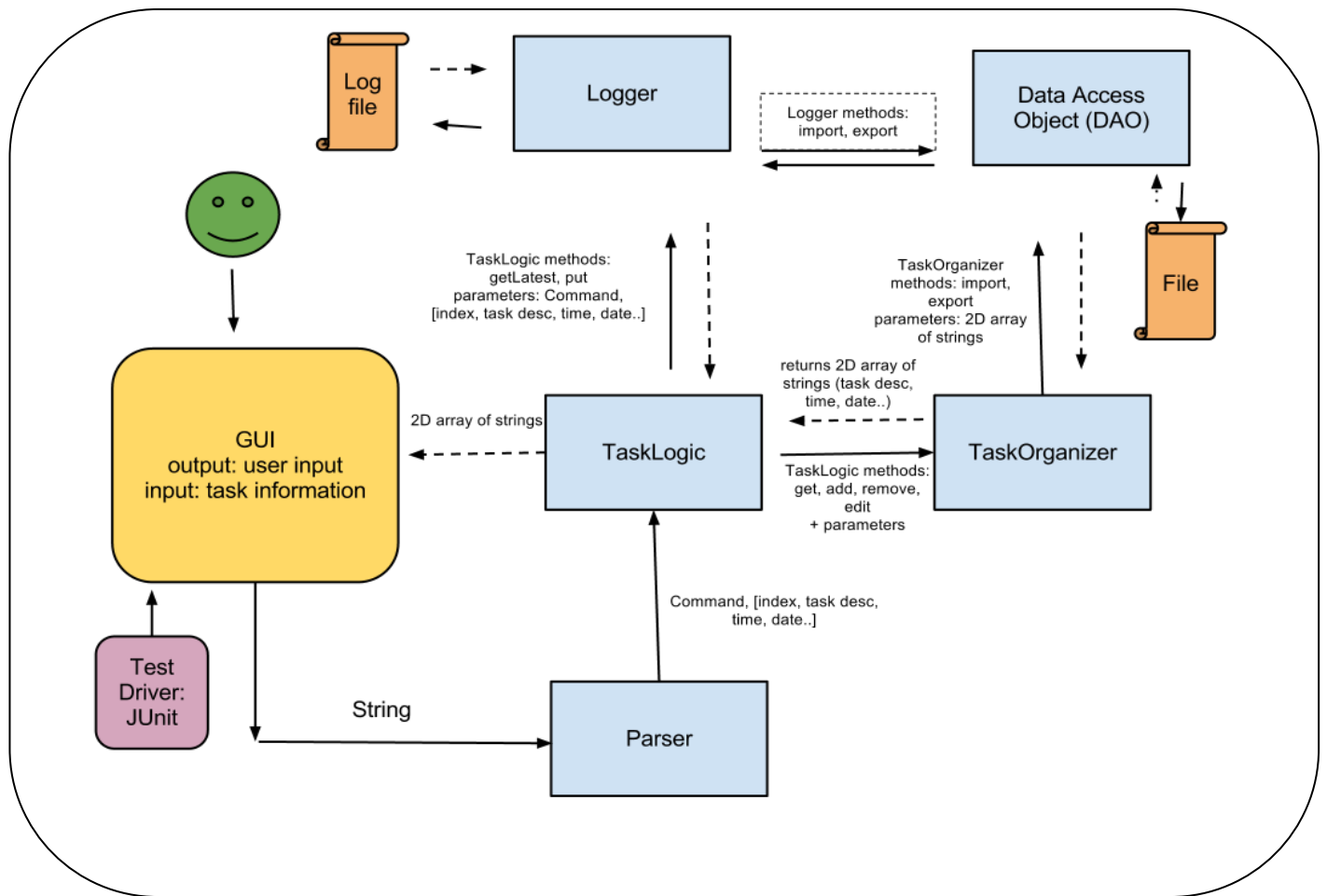
To give a brief description, the user view the GUI and enters data . This data is then passed to the parser which segregates the command into the respective keyword ,task name, priority number, date,etc and based on the keyword and function and passes the same to the respective methods/classes in the logic.

The logic then evaluates what the user wants and adds , deletes or edits the events according to the users' choice. The logic sends the logger the entire history i.e. every command that the user enters. The logger stores the entire history in a log file.

The storage uses files which will load data into respective classes in the logic , which then make the necessary changes and later store the data back into the file when all changes have been made.

Sequence flow of events in the program

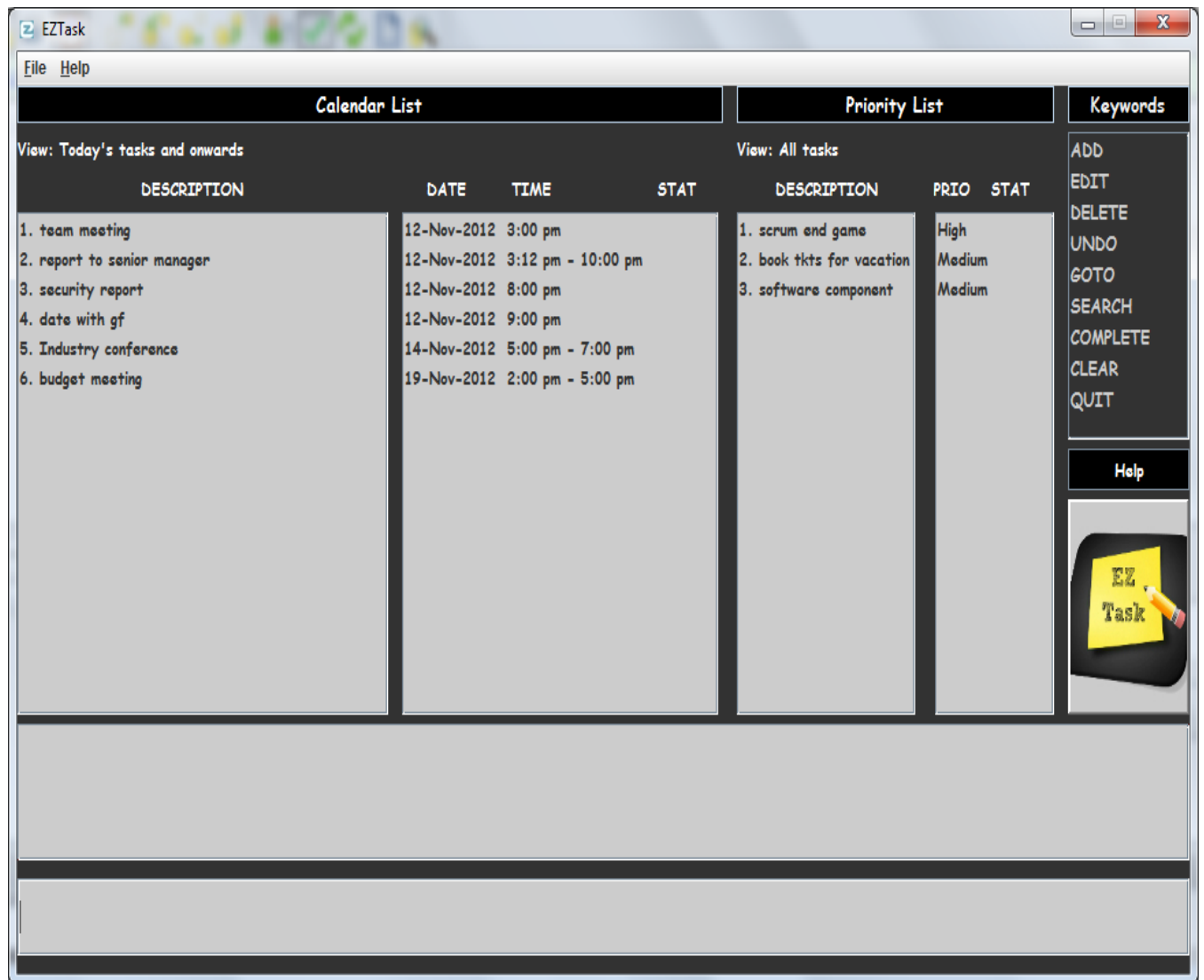




Detailed description :

GUI :

EZTask's most exciting feature is its GUI. Attractiveness and ease of use are the two most important goals that we focused upon. A sneak peak into the GUI is given below.



The tasks are placed into two different lists: Calendar and Priority Lists.

The calendar lists contains tasks which need to adhere to a time constraint: timed tasks(start and end time) or deadline tasks. The tasks in calendar list are displayed based on their date/ time in the descending order of the same. The priority lists contains tasks which are open ended and do not have any time constraints. The tasks in this list are displayed in the descending order of their priority, the top most being the one with highest priority.

Apart from this the GUI has the features window which gives the user an insight into the keywords to be used for various functions. Once these buttons are pressed , the keyword automatically appears in the command display window facilitating the user.

A log window is available which contains the results of the top five commands entered by the user for easy viewing of past transactions. We also have a

helpline which leads the user directly to a user guide document. The user windows can also be scrolled down to view future events

PARSER :

It receives the user input from GUI and manipulates the input to give a 2-D array which will contain the keyword, task names, etc that has to be passed as parameters to the respective methods that will perform the add , delete or edit operations.

```
public InputData parseUserInput(String userInput) throws Exception {

    int sepMark = userInput.indexOf(" ");
    String commandString;

    if(sepMark!=-1){
        commandString = userInput.substring(0, sepMark);
    }
    else{
        commandString = userInput;
    }

    if (COMMAND_ADD.contains(commandString.toUpperCase())) {
        userInput = userInput.replaceFirst(commandString, COMMAND_ADD);
        return dataForAdd(userInput);
    }
    else if (COMMAND_EDIT.contains(commandString.toUpperCase())) {
        userInput = userInput.replaceFirst(commandString, COMMAND_EDIT);
        return dataForEdit(userInput);
    }
    else if (COMMAND_DELETE.contains(commandString.toUpperCase())) {
        userInput = userInput.replaceFirst(commandString,
COMMAND_DELETE);
        return dataForRemove(userInput);
    }
    else if (COMMAND_GOTO.contains(commandString.toUpperCase())) {
        userInput = userInput.replaceFirst(commandString, COMMAND_GOTO);
        return dataForGoto(userInput);
    }
    else if (COMMAND_UNDO.contains(commandString.toUpperCase())) {
        userInput = userInput.replaceFirst(commandString, COMMAND_UNDO);
        return dataForUndo();
    }
    else if (COMMAND_SEARCH.contains(commandString.toUpperCase())) {
        userInput = userInput.replaceFirst(commandString,
COMMAND_SEARCH);
        return dataForSearch(userInput);
    }
    else if (COMMAND_COMPLETE.contains(commandString.toUpperCase())) {
        userInput = userInput.replaceFirst(commandString,
COMMAND_COMPLETE);
        return dataForComplete(userInput);
    }
}
```

```

    }
    else if (COMMAND_INCOMPLETE.contains(commandString.toUpperCase())) {
        userInput = userInput.replaceFirst(commandString,
COMMAND_INCOMPLETE);
        return dataForIncomplete(userInput);
    }
    else if (COMMAND_CLEAR.contains(commandString.toUpperCase())) {
        userInput = userInput.replaceFirst(commandString,
COMMAND_CLEAR);
        return dataForClear();
    }

    throw new Exception("Invalid command");
}

```

Here parseUserInput splits the user input and returns an array of strings which will contain the keyword , the task name, date , etc . For the purpose of passing the right number of parameters and the right parameters(changing string to integer,etc) we call separate functions for add, delete and other operations. One example of the above:

```

private InputData dataForAdd(String userInput) throws Exception {
    String[] parts = userInput.split(";"); //seperate on ;
    String[] inputs = parts[0].split(" ", 2); //seperate command
from desc
    String commandString = inputs[0];
    String descString = inputs[1];
    int id = -1; //no id specified or needed
    boolean isTimed;
    InputData inputData;
    TaskData taskData = null;

    String desc = parseDesc(descString);
    Command command = parseCommand(commandString);

    if (parts.length == 3) { //if timed task
        String dateString = parts[1];
        String[] timeParts = parts[2].split("-");
        String startTimeString = timeParts[0];
        int date = parseDate(dateString);
        int startTime = parseTime(startTimeString);
        int endTime = startTime; // default value if not pecified.
makes it a deadline task

        if(timeParts.length>1){ // if a end time specified
            String endTimeString = timeParts[1];
            endTime = parseTime(endTimeString);
            if(endTime==startTime){
                endTime = (startTime + 100)%2400;
            }
        }
    }
}

```

```

        isTimed = true;
        taskData = new TimedTaskData(isTimed, id, desc, date,
tartTime, endTime);
    }
    else if (parts.length == 2) { //if nontimed task
        String prioString = parts[1];
        int prio = parsePrio(prioString);
        isTimed = false;
        taskData = new NontimedTaskData(isTimed, id, desc, prio);
    }
    else{
        throw new Exception(MESSAGE_INVALIDINPUT_EXCEPTION);
    }
    inputData = new InputData(command, taskData);
    return inputData; // return exceptions if somethings wrong
}

```

UML Class Diagram:

<u>Parser</u>
-instance:Parser -Parser() +getInstance():Parser +parseUserInput(userInput: String):InputData -dataForAdd(userInput:String):InputData -dataForEdit (userInput:String):InputData -dataForRemove(userInput:String):InputData -dataForGoto(userInput::String):InputData -dataForComplete(userInput::String):InputData -dataForIncomplete(userInput::String):InputData -dataForClear(userInput::String):InputData -dataForUndo(userInput::String):InputData -dataForSearch(userInput::String):InputData -parseDate(dateString:String):int -parseTime(timeString:String):int -parseId(IdString:String):int -parseCommand(commandString:String):Command -parsePrio(prioString:String):int -parseDesc(descString:String):String -parseIsTimed(type:String):Boolean -getCurrentDate():int -getCurrentYear():int -getCurrentMonth():int -getCurrentDay():int -getCurrentTime():int -isInt(input:String):Boolean -isArray(string:String,array:String []):boolean)

TASK ORGANIZER:

A class called TaskOrganizer handles the tasks that go into the two different lists. It handles the operations of the two lists using array lists. It loads data from the files into these array lists and after the updation, stores it back in the files. This class performs all the major functions on the tasks. It also checks for validations, for example if the time and date are valid entries. Sample code:

```
private ArrayList<PriorityTask> priorityList;  
private ArrayList<CalendarTask> calendarList;  
private Storer storage;  
  
static private TaskOrganizer instance = null;  
  
private TaskOrganizer() {  
    priorityList = new ArrayList<PriorityTask>();  
    calendarList = new ArrayList<CalendarTask>();  
    storage = new Storer();  
    loadPrioTasks();  
    loadCalTasks();  
}
```

```

    public String addNontimedTask(String desc, int prio) throws
Exception {
    if (prio > 3 || prio < 1) {
        throw new IllegalArgumentException("Prio must be between 1 and 3");
    }
    PriorityTask task = new PriorityTask(desc, prio);
    priorityList.add(task);
    Collections.sort(priorityList);

    int index = priorityList.indexOf(task) + 1;
    String undoString = "DELETE p "+index;
    System.out.println(undoString);

    storePrioTasks();

    return undoString;
}

    public String addTimedTask(String desc, int date, int time)
throws Exception {
    if (!isValidDate(date)) {
        throw new Exception("Invalid date");
    }
    if (!isValidTime(time)) {
        throw new Exception("Invalid time");
    }
    CalendarTask task = new CalendarTask(desc, time, date);
    calendarList.add(task);
    Collections.sort(calendarList);

    int index = calendarList.indexOf(task) + 1;
    String undoString = "DELETE c "+index;
    System.out.println(undoString);

    storeCalTasks();
    return undoString;
}

```

UML Class Diagram:

UML Class Diagram:

<i><u>Logic</u></i>
-taskOrganizer: TaskOrganizer -ui: UI -parser: Parser -logger: Logger -startDate: int -startPrio: int -undoString: String
-executeInput(input: String): String +main(args: String [])

The Logic also calls the logger which keeps a log of all the commands that the user enters.

UML Class Diagram:

<i><u>Logger</u></i>
-writer: FileWriter; -buffWriter: BufferedWriter; -fileName: String
+logger(fn: String) +logString(string: String)

STORAGE:

It consists of two files: one for calendarlist and the other for prioritylist. Each of the file is used to load onto the lists in TaskOrganizer and save the tasks in the files again.

UML Class Diagram:

<i><u>Storer</u></i>
-readFile(filename: String): String[][] -writeFile(array: String[][], fileName: String)

Relationship between the classes:

