

# Design Documentation

This design document presents an overview of "A Simple Task Tracker". Within these pages, you will find a detailed exposition of my design philosophy, the architectural framework of the application, and a comprehensive guide to its behavior and user interactions. Furthermore, this document delves into the rationale behind the selection of specific technologies and methodologies. Most importantly, it highlights the unique features and distinguishing aspects of this app.

## Data Design

The log file in "A Simple Task Tracker" is divided into two key sections:

1. **\*\*Operation Log\*\***: This section records every modification made to the tasks, including the specific time of each operation. It provides a chronological account of all changes, additions, and deletions, offering a clear history of task modifications.
2. **\*\*Task Log\*\***: This part of the log captures all essential attributes of a task, presenting a detailed overview of each task's current state. It is a comprehensive log that includes all relevant details about the tasks. To be specific, these attributes are task name, task size, task start time, task end time, and task description.

Both sections have unique titles and formats, established at the program's start, ensuring organized and straightforward tracking of tasks.

The Operation log is designed with human readability in mind. It offers clear historical records for users to review. This feature ensures that users can easily revisit and check past details, such as previous task names or other historical data, especially after performing operations like renaming tasks.

The Task Log is primarily designed for machines to read. It contains all the vital information necessary for each task. When initializing a task, fields such as task size or description may initially be unset. In such cases, the program assigns the value "UNDEFINED" to these fields. This approach ensures clarity and consistency in data representation.

To facilitate operations like summary, which rely on task name or size, the system employs specific rules for user inputs. Users are restricted from naming tasks with terms like "S", "M", "L", "XL", and "UNDEFINED". This restriction is crucial to prevent the program from misinterpreting these inputs while parsing data from the log file, particularly when distinguishing between a task name and its size.

Additionally, all time values in the Log are formatted as “yyyy/MM/dd-HH:mm:ss”. This standardized time format contributes to the log’s machine readability, ensuring that time data is consistently and accurately processed.

When a task's end time is first set up in the system, it defaults to a placeholder value of "2000/01/01-00:00:00". This specific value acts as an indicator that the task is currently active or running. It signifies an undefined end time. Once the task is stopped, this end time value is then updated to reflect the actual duration of the task. This is calculated based on the difference between the time the task is stopped and its start time, ensuring an efficient tracking of task duration.

However, it's important to note that manual modifications to the task's end time in the log can lead to inconsistencies. If a user changes the end time to a value that is neither the predefined minimum ("2000/01/01-00:00:00") nor greater than the task's start time, the system may calculate a negative duration value. Such scenarios underscore why direct editing of the time log by users is not advisable, as it can disrupt the accurate tracking and representation of task durations.

## Behavior Design

The behavior of an app is akin to a finely-tuned gear system. A minor misstep can lead to a complete redesign, making the definition of app behavior crucial before implementation. To ensure a seamless logic flow, I've outlined several key behavioral aspects.

1. Users are allowed to restart a task, but only after it has been stopped. This action will generate two time windows in the task log, enabling effective tracking.
2. The summary report is divided into two parts. The first part details the time spent on each task. The second part provides statistical data, including minimum, maximum, and average values, for the number of tasks of the same size when there are more than one.
3. Renaming a task will update all instances of that task's name in the task log. This feature prevents confusion and ensures that tasks can be interrupted, restarted, and renamed without losing coherence.
4. The operation log records only those actions that alter tasks. Since generating a summary report does not modify any task, this operation will not be logged in the operation log.

## Project Structure Design

In the preceding sections, we gained a clear insight into the problem domain and specific requirements. Now, it's an appropriate time to introduce a suitable structural design to further optimize the app's functionality.

The workload is bifurcated into two primary components. The 'Logger' class takes charge of creating, reading, and writing the log file. It interprets file contents using predefined formats and sequentially processes each line from both the operation log and task log into respective lists. For

effective management, the Logger class incorporates a list for the operation log, archiving each action for future reference, and a list for the task log, maintaining comprehensive data on all tasks for operational purposes. Upon initialization, the Logger class reads, interprets, and stores all information from the log file. A singleton pattern is employed to prevent redundant initializations, thereby conserving resources. The Logger class is equipped with interfaces for various operations like start, stop, summary, etc., which can be directly invoked from the main program. These operations may allow the Logger class to write new log information to the file.

The second crucial component of the system is the 'Task' class, which is responsible for representing each individual task. This class allows for the initialization of a task by specifying essential parameters such as task name, task size, start time, end time, and description. These attributes encompass all the necessary information for task operation. The Task class is designed with interfaces for various actions like stop, summary, rename, etc. These interfaces enable the Logger class to execute methods on a task object. This design ensures that each task is fully encapsulated, allowing for efficient and effective functionality within the system.

## Design Highlights

```
// Constants class
class Constants{

    // Log file name
    protected static final String LOG_FNAME = "TM_log.txt";

    // log file section names
    protected static final String OP_LOG = "Operation Log:";
    protected static final String TASK_SUMMARY = "Task Summary:";

    // DateTime format
    protected static final DateTimeFormatter FORMATTER
    = DateTimeFormatter.ofPattern(pattern:"yyyy/MM/dd-HH:mm:ss");

    // Minimum time which used to compare
    protected static final ZonedDateTime MIN_TIME
    = ZonedDateTime.parse(text:"2000/01/01-00:00:00", FORMATTER.withZone(ZoneId.systemDefault()));

    // Printing tasks formats
    protected static final String PRINT_FORMAT = "%-22s";
    protected static final int PRINT_GAP = 22;
    protected static final String LABEL =
        String.format(Constants.PRINT_FORMAT, ...args:"Task Name")
        + String.format(Constants.PRINT_FORMAT, ...args:"Task Size")
        + String.format(Constants.PRINT_FORMAT, ...args:"Start Time")
        + String.format(Constants.PRINT_FORMAT, ...args:"End Time")
        + String.format(Constants.PRINT_FORMAT, ...args:"Description");
```

A 'Constants' class is implemented to centralize the management of format and string values, such as log names and error messages. This ensures that these elements are consistently controlled from a single location.

The system incorporates a well-structured exception control mechanism to ensure stability and prevent illegal arguments and operations. This approach is essential for maintaining the robustness of the application.

```
if (args.length == 2){
    logger.deleteTask(args[1]);
}else{
    throw new IllegalArgumentException(Constants.DELETE
        + ": " + Constants.ERR_ARGUMENT);
}
```

```
// Size values
enum TASK_SIZE{
    UNDEFINED, S, M, L ,XL
}
```

Enums, such as `TASK\_SIZE`, are utilized to represent values like task sizes. This makes the program more efficient and structurally appropriate.

The use of predicate and supplier functions is employed to enhance both readability and efficiency within the program.

```
// Check if name exists
protected Predicate<String> hasTask = name -> taskName.equals(name);

// Check if this task is still going
protected Supplier<Integer> isRunning = () -> taskEnd.compareTo(Constants.MIN_TIME);
```

```
// Class instance
private static Logger instance;

private List<String> operationLog = new ArrayList<>();
private List<Task> taskSummary = new ArrayList<>();

// Store cleaned tasks
Map<String, Duration> map = new HashMap<>();
```

The design ensures full encapsulation and information hiding, while utilizing maps to categorize and organize data effectively.

The Logger class is designed with a private constructor to prevent the creation of multiple instances, adhering to a singleton design pattern.

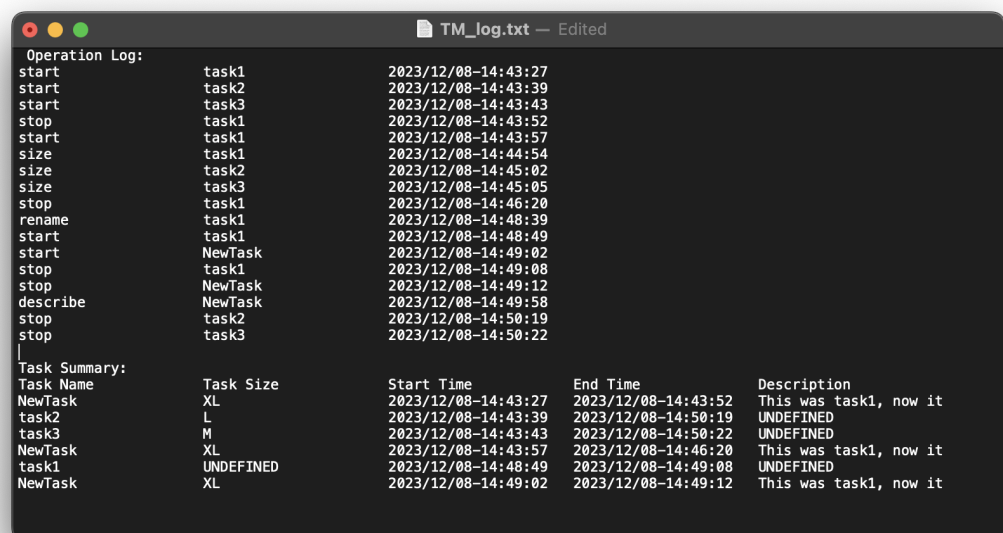
```
// Private constructor
private Logger(){
    try{
        if(!file.exists()){
```

```
tinypink@Andys-MacBook-Pro-5 TM % java TM.java summary
Task Name      Time Spent
task1          0 Hours, 1 Minutes, 53 Seconds
task2          0 Hours, 1 Minutes, 46 Seconds
task3          0 Hours, 1 Minutes, 42 Seconds

M:
Min: 0 Hours, 0 Minutes, 25 Seconds
Max: 0 Hours, 1 Minutes, 42 Seconds
Avg: 0 Hours, 1 Minutes, 11 Seconds
```

The system generates a clear summary report that includes computed statistical results.

The log file meticulously records every operation history and task information, with all types of information stated clearly.



```
TM_log.txt — Edited

Operation Log:
start task1 2023/12/08-14:43:27
start task2 2023/12/08-14:43:39
start task3 2023/12/08-14:43:43
stop task1 2023/12/08-14:43:52
start task1 2023/12/08-14:43:57
size task1 2023/12/08-14:44:54
size task2 2023/12/08-14:45:02
size task3 2023/12/08-14:45:05
stop task1 2023/12/08-14:46:20
rename task1 2023/12/08-14:48:39
start task1 2023/12/08-14:48:49
start NewTask 2023/12/08-14:49:02
stop task1 2023/12/08-14:49:08
stop NewTask 2023/12/08-14:49:12
describe NewTask 2023/12/08-14:49:58
stop task2 2023/12/08-14:50:19
stop task3 2023/12/08-14:50:22

Task Summary:
Task Name Task Size Start Time End Time Description
NewTask XL 2023/12/08-14:43:27 2023/12/08-14:43:52 This was task1, now it
task2 L 2023/12/08-14:43:39 2023/12/08-14:50:19 UNDEFINED
task3 M 2023/12/08-14:43:43 2023/12/08-14:50:22 UNDEFINED
NewTask XL 2023/12/08-14:43:57 2023/12/08-14:46:20 This was task1, now it
task1 UNDEFINED 2023/12/08-14:48:49 2023/12/08-14:49:08 UNDEFINED
NewTask XL 2023/12/08-14:49:02 2023/12/08-14:49:12 This was task1, now it
```