Shaun Teo - Project Portfolio

PROJECT: El Duque

Overview

El Duque is a desktop task manager application for computer science student. The user interacts with it through a CLI and is written in java with over >10 kLOC. It is enhanced from the original Duke application.

Summary of contributions

- **Enhancement**: added **ability to edit tasks**
  - What it does: allows the user to edit every parameter of the task. It has support for filtered indexing as well.
  - Justification: This feature improves the product significantly because a user can make mistakes in creating a task and now they can rectify those mistakes.
  - Highlights: This feature cuts across several layers of the project, hence designing it in such a way to work with existing classes was a very important step.
- **Enhancement**: added **pomodoro timer**
  - What it does: The Pomodoro feature is an inbuilt timer based on the pomodoro workflow. It allows the user to run a timer based on the 25-5 work-rest work ratio and reminds users to manage their work-rest ratio properly.
  - Justification: During work sessions, this provides the users the ability to only rely on one app to manage their work session.
  - Highlights: This feature is implemented as a Singleton as there is only one instance of this object. Supports adding and completing of tasks from the task list which required a dependency with the task list class.
- **Code contributed**: [Code]
- **Other contributions**:
  - Project management:
    - Managed issues and milestones for v1.1 and issues for v1.4.
    - Lead the team to ensure tasks are assigned properly and project requirements met.
  - Enhancements to existing features:
    - Wrote test code for existing commands.
    - Enforced code quality.
  - Documentation:
    - Converted UserGuide from a Google Doc to an AsciiDoc to improve the formatting.

- Tools:
  - Implemented and managed checkstyle plugin.

Contributions to the User Guide

Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

El Duque relies on a flexible command line syntax to parse in commands. Here is the general form of that said syntax.

```
[-FILTER] COMMAND [-MODIFICATIONS]
```

There are 3 main parts to the syntax and filter and modifications have to be prefixed with `-`. Filter and modifications are optional. They serve to add extra depth to the basic commands. You can think of filters as sublists. One very important point to note is that `INDEX` is relative to `FILTER`.

For example:
`edit 3 -description do laundry` can be doing exactly the same thing as `-housechores edit 1 -description do laundry` if the first task in housechores filter tasklist is the same as the third task in the overall tasklist.

Command Format

- Words in `UPPER_CASE` are the parameters to be supplied by the user e.g. in `task DESCRIPTION`, `DESCRIPTION` is a parameter which can be used as `task example`.
- Items in square brackets are optional e.g `DESCRIPTION [-p PRIORITY]` can be used as `task example -p high` or as `task example`.
- All optional modifications except FILTER can be in any order e.g. if the command specifies `task DESCRIPTION [-p PRIORITY] [-t DATETIME]`, `task DESCRIPTION [-t DATETIME] [-p PRIORITY]` is also acceptable.

Editing a task : `edit`

El Duque supports the editing of any parameters of a given task. The parameters to be edited must begin with `-KEYWORD` followed by the edit.

```
[-FILTER] edit INDEX -KEYWORD [MODIFICATION]
```

Editing the description and recurrence of the first task of the list is as follows.

edit 1 -desc gym -r daily

NOTE The order of the parameters does not matter!

Shown below are the list of keywords and the respective attributes that they edit.

- `-f` : filter
- `-des` / `-desc` / `-descript` : description
- `priority` : priority
- `t` : date and time
- `d` : duration of the task
- `r` : recurrence

Pomodoro Timer: `pomo`

El Duque includes a pomodoro timer to complement the pomodoro workflow. This timer has 3 states, **work**, **short break** and **long break**. A work cycle is followed by a **short break**, every 4th break will be a **long break**.

Pomodoro supports the adding of task to a temporary pomodoro task list for you to keep track of tasks you wish to complete this pomodoro work session.

Starting a pomodoro: `start`

Starts the timer for the current cycle that it is in. First instance of this would be the work cycle.

```
pomo start
```

El Duque will subsequently update you on the time remaining for the current state every 5 minutes.

NOTE Once the timer has ended, the next call of `pomo start` will be the next state. If the current state is **work**, the next call will be a **break**.

Answers to brain teaser: `answer`

El Duque contains a bank of brain teasers that will be shown every time a break has started. We hope this gives you something to take your mind of work. You can use the `answer` keyword to reveal the answer to the brain teaser.

```
pomo answer
```

NOTE Brain teaser bank is randomised, you might not get the same question the next break. Make sure to type `pomo answer` before you start your next break to view the answer to this current brain teaser!

Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

TaskList component

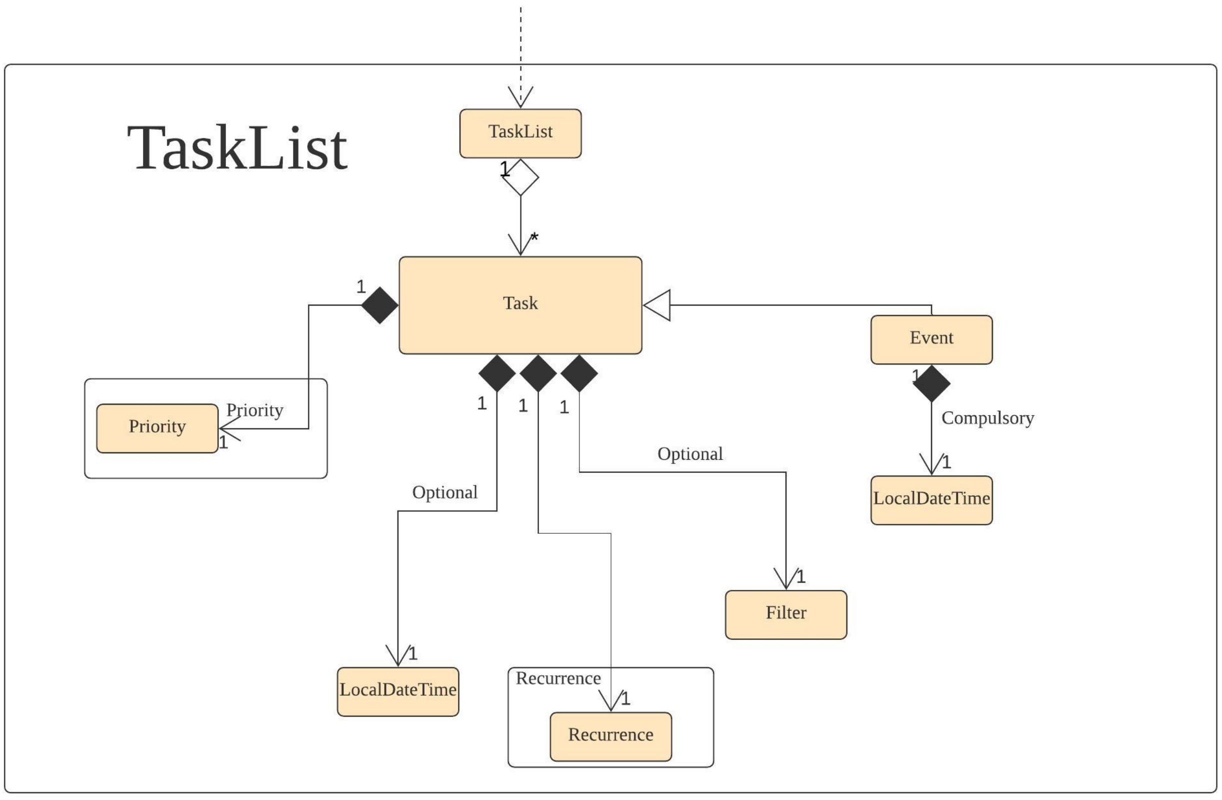

Figure 1. Structure of the TaskList Component

The `TaskList`:

- Stores an array list of tasks that represents tasks that the user wishes to store.
- Event object extends tasks and enforces a compulsory `LocalDateTime` object denoting the time of the event
- `TaskList` does not depend on other components

Editing

Implementation

The edit feature extends `Command` class. This commands takes in the filter and a string containing the attributes and it’s update field.

Given below is an example usage scenario of an edit that includes a filter and how it behaves at each step. The command given is:

```
-cs edit 5 -priority 2
```

1. `Parser` separates the full command into a filter, `-cs` and the string, `edit 5 -priority 2`. Parser subsequently returns an `EditCommand` with these two strings as parameters.
2. `Duke` calls `Command.execute()`, which runs the overridden function in `EditCommand` and takes in `TaskList`, `UI` and `Storage`.
3. `Edit` command will call 3 functions to get the relevant fields. `getParameters()` which splits the command by regex `▯ -▯` and returns a string array, in this case `▯priority▯`. `getIndexFromCommand()` returns an integer of the index in the command, `5`. `getIndexFromTaskList()` takes in the filter, index from command and the list of tasks. It will loop through `TaskList` and finds tasks with the priority filter and matches the index specified by the user to the actual index in the task list.
4. `EditCommand` will access the specified task using the index and calls `setPriority` to update the priority of the task.

The following sequence diagram summarises how the edit operation works:

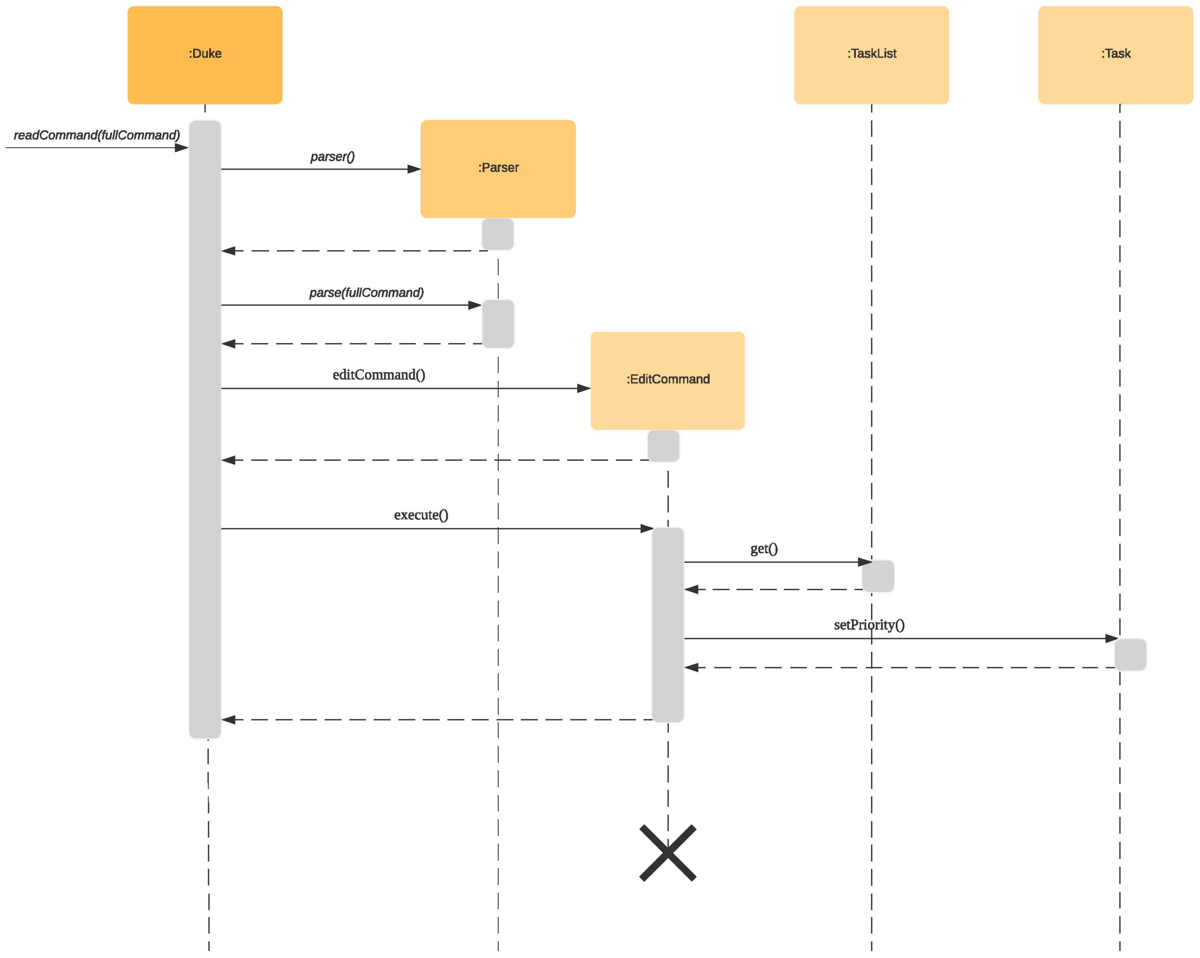

Figure 2. Sequence Diagram for Edit Command

Design Considerations

Aspect: How actual index of task is found when given an index based on the filtered list

- **Alternative 1 (current choice):** Using 2 separate counters to keep track of actual index versus filtered index
    - Pros: Easy to implement.
    - Cons: Harder to understand.
- **Alternative 2:** Saves a copy of the filtered list
    - Pros: Clearer to understand.
    - Cons: Have to deal with multiple lists.

Aspect: How the task are updated

- **Alternative 1 (current choice):** Updating each attribute of a task separately.
    - Pros: Reduces the dependency needed, `EditCommand` only needs to call a setter function instead of a getter for existing attributes followed by `AddCommand`.
    - Cons: Have to create a setter for each attribute of task
- **Alternative 2:** Create a new task with updated fields.
    - Pros: Won’t need to call individual setters for each attribute.
    - Cons: Have to deal with AddCommand class to create a new task.

Pomodoro feature

Implementation

The Pomodoro feature is an inbuilt timer based on the pomodoro workflow. It allows the user to run a timer based on the 25-5 work-rest work ratio and reminds users to manage their work-rest ratio properly.

This feature is implemented using the Singleton design pattern with only one instance of the Pomodoro object. Accessing this instance is through the getInstance() method.

The Pomodoro object instantiates a java.util timer and the pomodoroTimerTask class and manages the starting and stopping of the timer. The class is configured as follows:

- The timer is configured to run the pomodoroTimerTask.run() every minute.
- pomodoroTimerTask takes in an integer parameter minutesRemaining and decrements it every minute. This determines how many minutes to run this timerTask for. After it reaches 0, the timer task will cancel itself.

Pomodoro States The states are configured based on the pomodoro workflow and are as follows:

- Work: Set to 25 minutes intervals, once completed will switch to a break.

- Short Break: Set to 5 minutes interval, runs after ever 1st, 2nd and 3rd work cycle.
- Long Break: Set to 15 minutes interval, runs after every 4th work cycle.

Commands This feature uses the keyword pomo followed by a few commands to control it's actions as listed below.

```
pomo [MODIFICATION]
```

`start`: Begins the timer.
`status`: Outputs the current status of the pomodoro, minutes remaining and current cycle.
`reset`: Resets the timer back to the previous state.
`restart`: Restarts the pomodoro to the 1st work cycle.
`stop`: Stops the timer.
`add INDEX`: Assigns a task to the current pomodoro and prints out during status updates.
`done INDEX`: Marks a pomodoro task as complete and removes it from the pomodoro list.
`answer` : Shows the answer to the brain teaser.

Given below is the sequence diagram of how the pomodoro feature works.

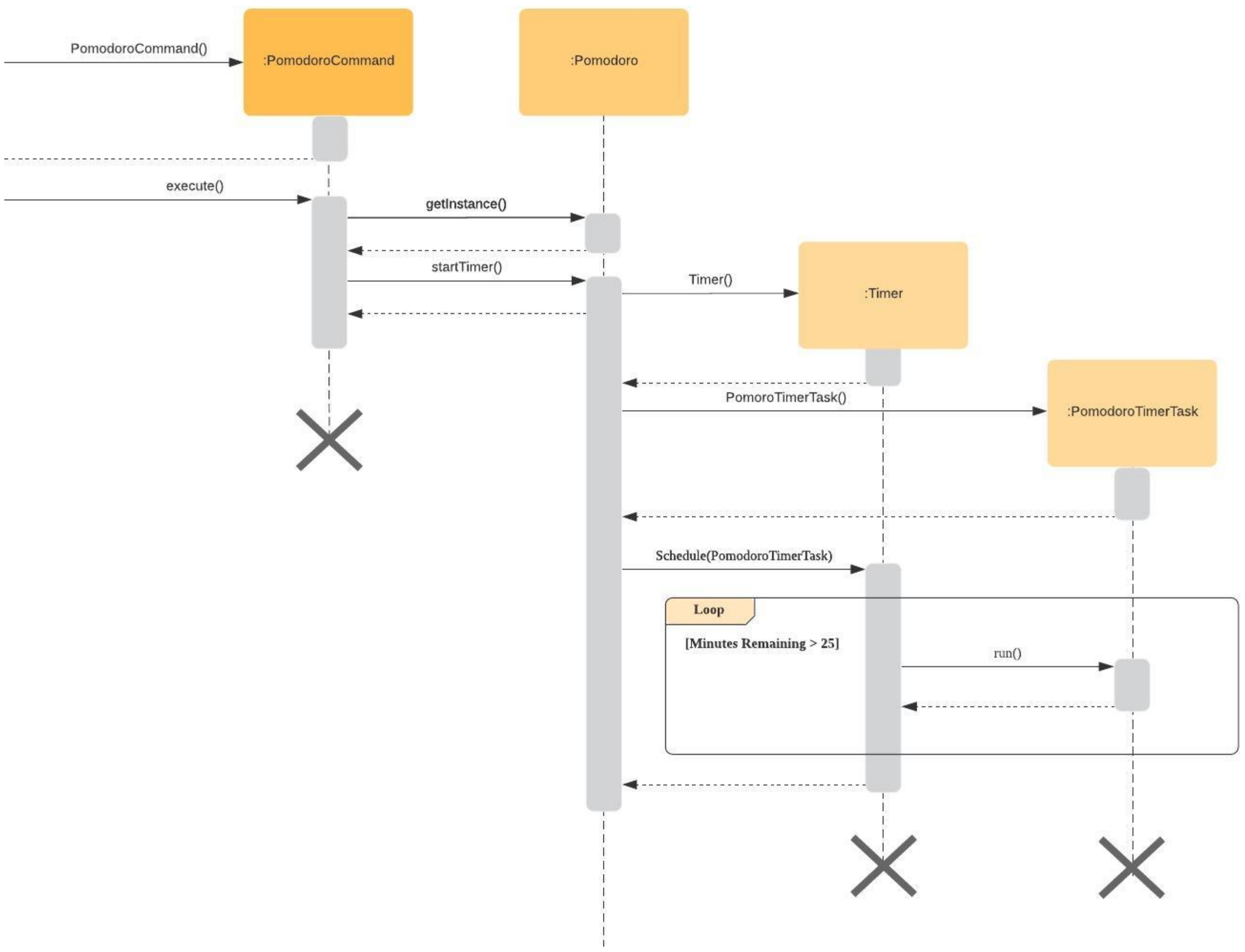

Figure 3. Sequence Diagram for Pomodoro Feature

Design Considerations

Aspect: Where to instantiate this pomodoro class

- **Alternative 1 (current choice):** Instantiate it as a singleton.
    - Pros: Easy way to access this pomodoro class from anywhere in the code base. There is only ever one instance of this class, multiple objects of this class will result in incorrect timer control.
    - Cons: Acts like a global variable, increasing coupling accross the code base.
- **Alternative 2:** Instantiate it in Duke and pass it into PomodoroCommand.
    - Pros: Easier to test, Singletons carry data from one test to another.
    - Cons: Have keep passing this object to commands that require it.

Aspect: How pomodoro tasks are stored

- **Alternative 1 (current choice):** Create a new array list of tasks and add tasks into it.
    - Pros: Reduces the dependency needed, PomodoroCommand does not need to call TaskList when it needs to run getDescription().
    - Cons: Have to write more code to maintain this new array list.
- **Alternative 2:** Store indexes of tasks that are added.
    - Pros: Won’t need to maintain another array list in the Pomodoro Class.
    - Cons: Increases coupling, have to constantly reference TaskList class when managing tasks.