Yap Dian Hao - UML diagrams

1. Storage Class Diagram

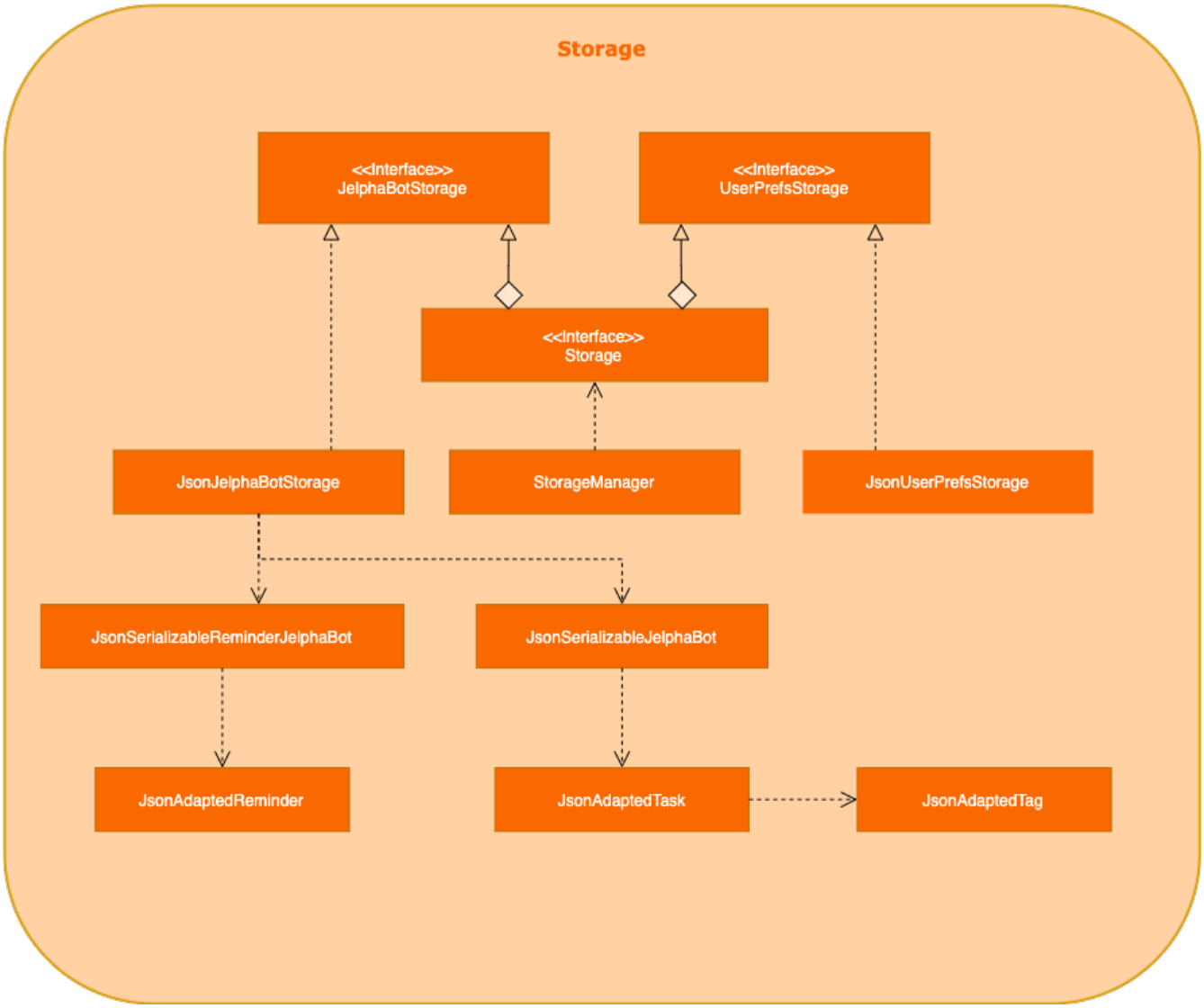

Figure 1. Structure of the Storage Component

`StorageManager` is the main functional class in the `Storage` component, extending the interfaces `JelphaBotStorage` and `UserPrefsStorage`. `JsonUserPrefsStorage` can save and read `UserPref` objects in .json format, while `JsonJelphaBotStorage` can save `JelphaBot` 's data and read them. `Reminder` and `Task` are converted to the json-friendly `JsonAdaptedTask` and `JsonAdaptedReminder`, then saved by `JsonJelphaBotStorage`.
This class diagram enables developers to have an overarching view of the `Storage` component and its interactions with the json files. It is crucial due to certain aspects such as `JsonAdaptedReminder` are not found in the original `Storage` component.

2. Reminder Class Diagram

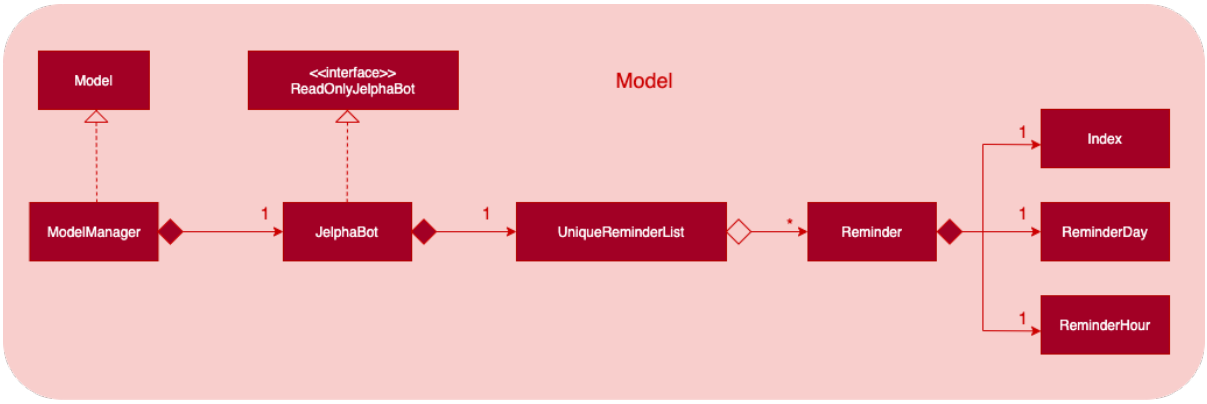

Figure 2. Reminder Class Diagram in the Model component

`Reminder` is a newly introduced class to the product, and has the same significance of a `Task`. `ModelManager` manages the adding and removing of a `Reminder` by calling the `addReminder` method of `JelphaBot`, which adds a `Reminder` into its `UniqueReminderList`. A `UniqueReminderList` can store arbitrary amount of `Reminder`, in which each `Reminder` has an `Index` of the task that it needs to remind, a `ReminderDay` which refers to the day the `Task` will be reminded, and a `ReminderHour`.
This class diagram plays a crucial role in summarizing the `Reminder` object and its relations with the `Model` component.

3. Reminder Sequence Diagram

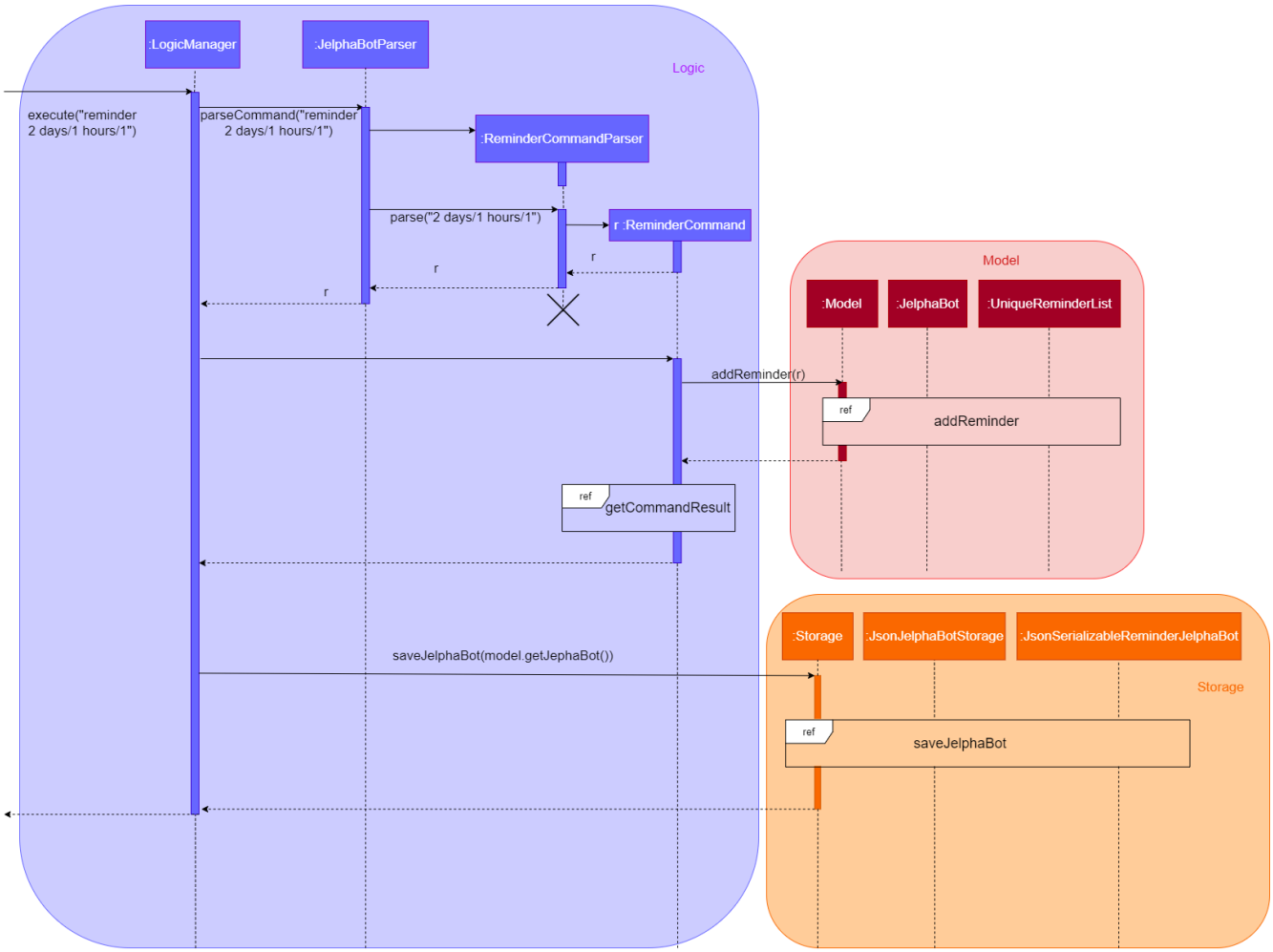

Figure 3. Sequence Diagram after running reminder 2 days/2 hours/1

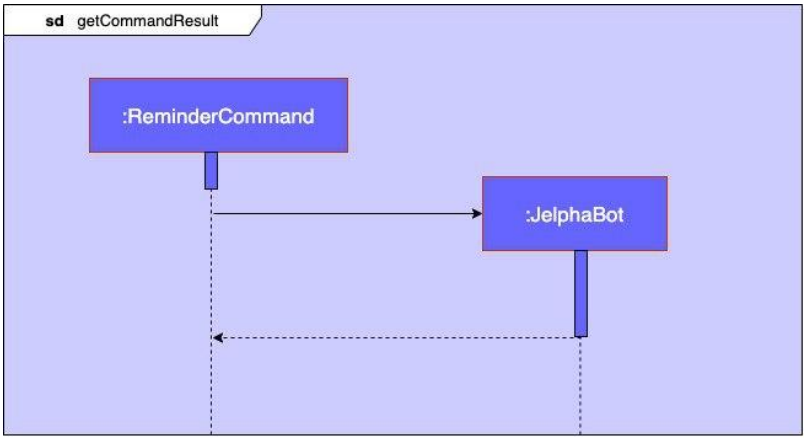

Figure 4. The reference frame of getting the `CommandResult` in the `Logic` component.

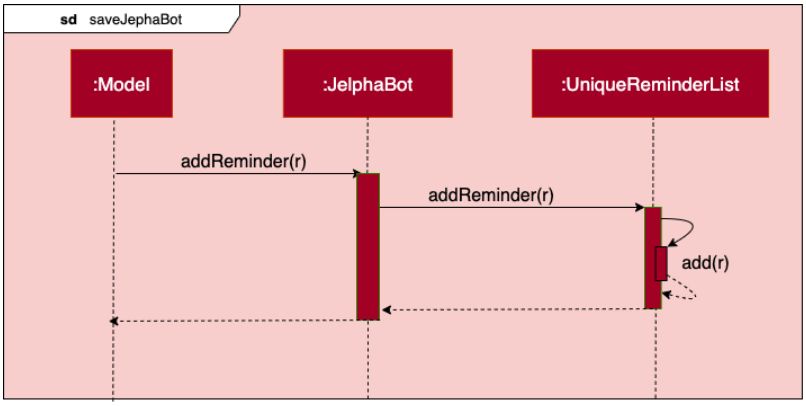

Figure 5. The reference frame of adding the `Reminder` in the `Model` component.

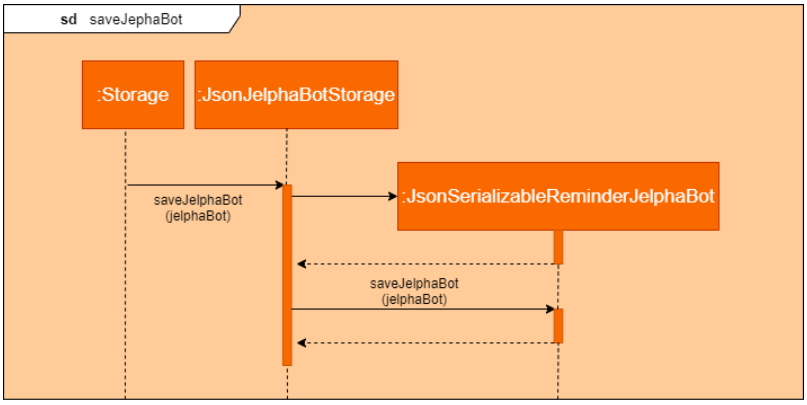

Figure 6. The reference frame of saving a `Reminder` by the `Storage` component.

This sequence diagram provides a summary of the process from the user enters the command to the storing of the `Reminder` object. Next, The `Logic execute()` method creates a `ReminderCommand` from the input string by parsing the input according to the command word and several other attributes. Next, the input string is converted into `Index`, `ReminderDay`, `ReminderHour`, and a `Reminder` object with these properties are forwarded to `Model`. The `Model` first checks the validity of the attributes respectively. The valid `Reminder` is then added to the `UniqueReminderList`. Lastly, the `Logic` fires the `Storage` to save the `Reminder`.

This diagram is the most important diagram for the `Reminder` feature as it provides a complete picture from front-end to back-end. Since the process is not trivial, several reference frames are required for developers to view in isolation of components for a clearer picture, and to avoid tiny fonts if all the processes are combined in a sequence diagram.