Chen Yi Jun - Project Portfolio

PROJECT: Plan²travel

The purpose of this document is to document the contribution that I have made in the project: ***Plan²travel*** as a student undertaking the module CS2103T under NUS School of Computing.

Plan²travel Github link: https://ay1920s1-cs2103t-t09-1.github.io/main/

---

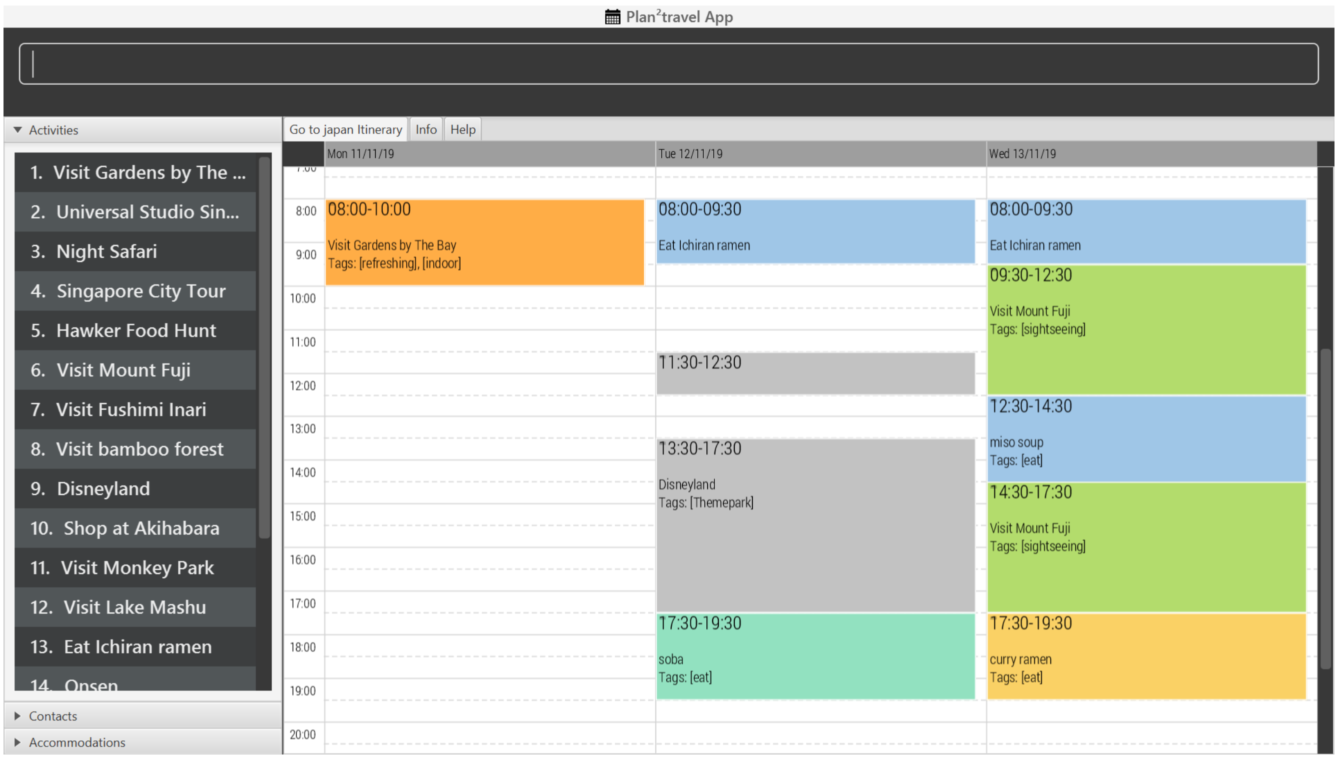
1. Introduction

Plan²travel is a desktop travel planning application. The application is targeted at fellow university students who travel infrequently and would greatly benefit from an application that helps organize their travelling information when they plan their own itinerary. Plan²travel has the ability to store and display information such as activities, accommodations, contacts and an itinerary which comprises of a list of day. Each day is displayed with a list of activities that is scheduled.

The user interacts with the application using a Command Line Interface (CLI), and it has Graphical User Interface GUI created with JavaFX.

It is cross-platform and can be compiled for both Windows and Mac OS.

Below is a screenshot of our application:



2. Summary of contributions

Below is a summary of the contributions that I have made to the project.

2.1. Major enhancement:

- **Added the ability to undo/redo previous commands**
  - **What it does:** allows the user to undo all previous commands one at a time. Preceding undo commands can be reversed by using the redo command.

    NOTE Only undoable commands can be undone by the 'undo' command.

  - **Justification:** This feature improves the product significantly because a user can make mistakes in commands and the app should provide a convenient way to rectify them.
  - **Highlights:** This enhancement affects existing commands and commands to be added in future. It required an in-depth analysis of design alternatives. The implementation too was challenging as it required changes to existing commands.
- **Functional Code Contributed**: Update 1 | Update 2 | Update 3 | Update 4 | Update 5 | Update 6 | Update 7 | Final Update
- **Test Code Contributed**: Test 1 | Test 2

2.2. Minor enhancement:

- **Morph original code base for Model and Storage package**
  - **What I did:** I modified the entire model package from the original code base and added new model class files, such as Accommodation, Activity and Contact. I also remove traces of unused codes from the previous code base. Aside morphing model, I also morphed the storage package, allowing the application's data to be saved into 4 different json data files instead.
  - **Justification**: Morphing has to be done to the old code base in order to fit the goals of our application. Once the code base is morphed, the team is able to proceed with implementing their individual features of the application.
  - **Highlights**: I gained a deeper understanding of how the different components of the code base worked, especially for storage component. With the insights gained, I was able to advise my teammates on how they should approach their implementations when working with these components.
- **Functional Code Contributed**: Update 1 | Update 2 | Final Update
- **Test Code Contributed**: Test

2.3. Other Contributions:

- Tools:

    - Set up build tools for maintaining and automation of team repository. travis, appveyor, coverall
- Documentation:
    - Contributed to the User Guide and Developer Guide for this project. See below for more details.
- Community:
    - Over 30 Pull Requests on Github
    - Over 15 Reviews on Github

Overall Code Contribution here

3. Contributions to the User Guide

Given below are sections I contributed to Plan²travel User Guide. They showcase my ability to write documentation targeting end-users.

3.1. Undo: `undo`

Allows user to undo by one action. Only Undoable commands executed previously by the user can be undone. Refer to the list below for all possible undoable commands.

Format: `undo`

NOTE undo command cannot be chained (i.e. 'undo undo undo' does not result in 3 undos applied), but typed one at a time.

List of UndoableCommand:

add activity/ accommodation/ contact/ days
delete activity/ accommodation/ contact/ day
edit activity/ accommodation/ contact
schedule
unschedule
autoschedule
optimise
clear

3.2. Redo: `redo`

Redo by one action. This command is to revert the changes of the latest undo.

Format: `redo`

NOTE redo command can only be called after undo.

The **4 screenshots** below demonstrates the Undo/Redo command when the user executes `optimise budget command`.

- Before optimise

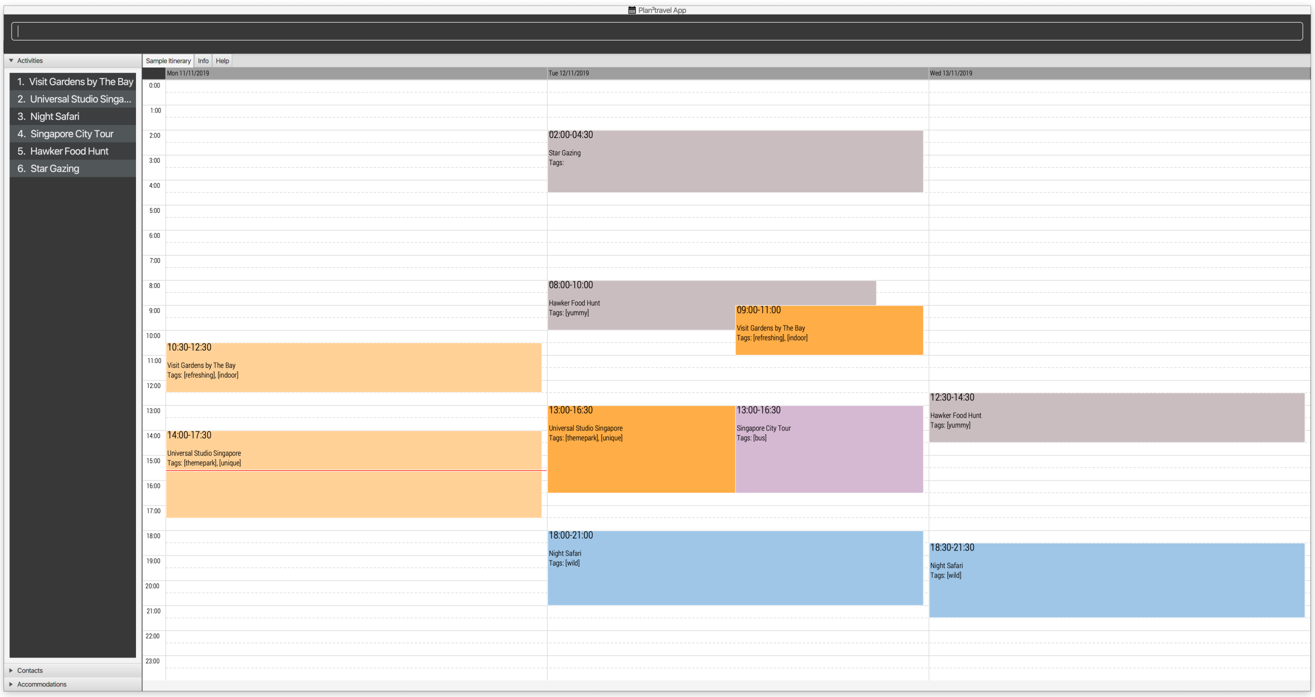

Plan²travel App
Activities
1. Visit Gardens by The Bay
2. Universal Studio Singa...
3. Night Safari
4. Singapore City Tour
5. Hawker Food Hunt
6. Star Gazing
Contacts
Accommodations
Sample Itinerary
Info
Help
Mon 11/11/2019
Tue 12/11/2019
Wed 13/11/2019
10:30-12:30 Visit Gardens by The Bay Tags: [refreshing], [indoor]
14:00-17:30 Universal Studio Singapore Tags: [themepark], [unique]
02:00-04:30 Star Gazing Tags:
08:00-10:00 Hawker Food Hunt Tags: [yummy]
09:00-11:00 Visit Gardens by The Bay Tags: [refreshing], [indoor]
13:00-16:30 Universal Studio Singapore Tags: [themepark], [unique]
13:00-16:30 Singapore City Tour Tags: [bus]
18:00-21:00 Night Safari Tags: [wild]
12:30-14:30 Hawker Food Hunt Tags: [yummy]
18:30-21:30 Night Safari Tags: [wild]

Figure 1. Day 2 has lots of overlapping activities

- After optimise 2

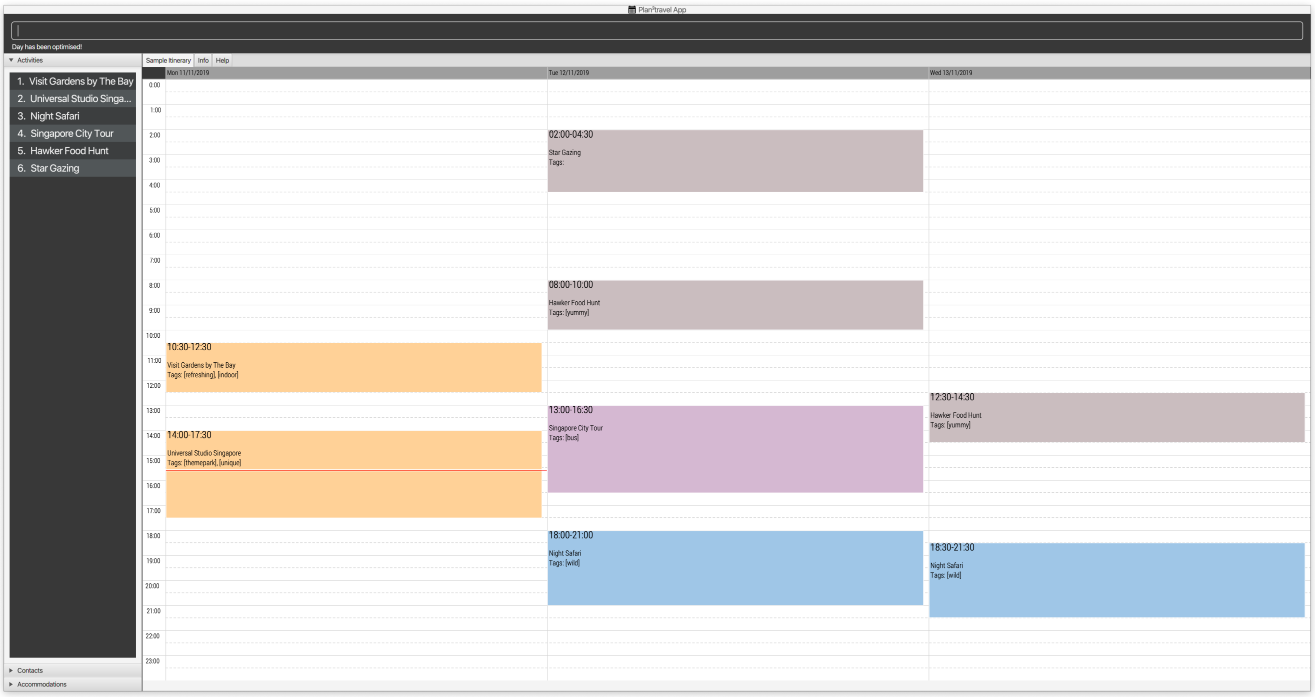
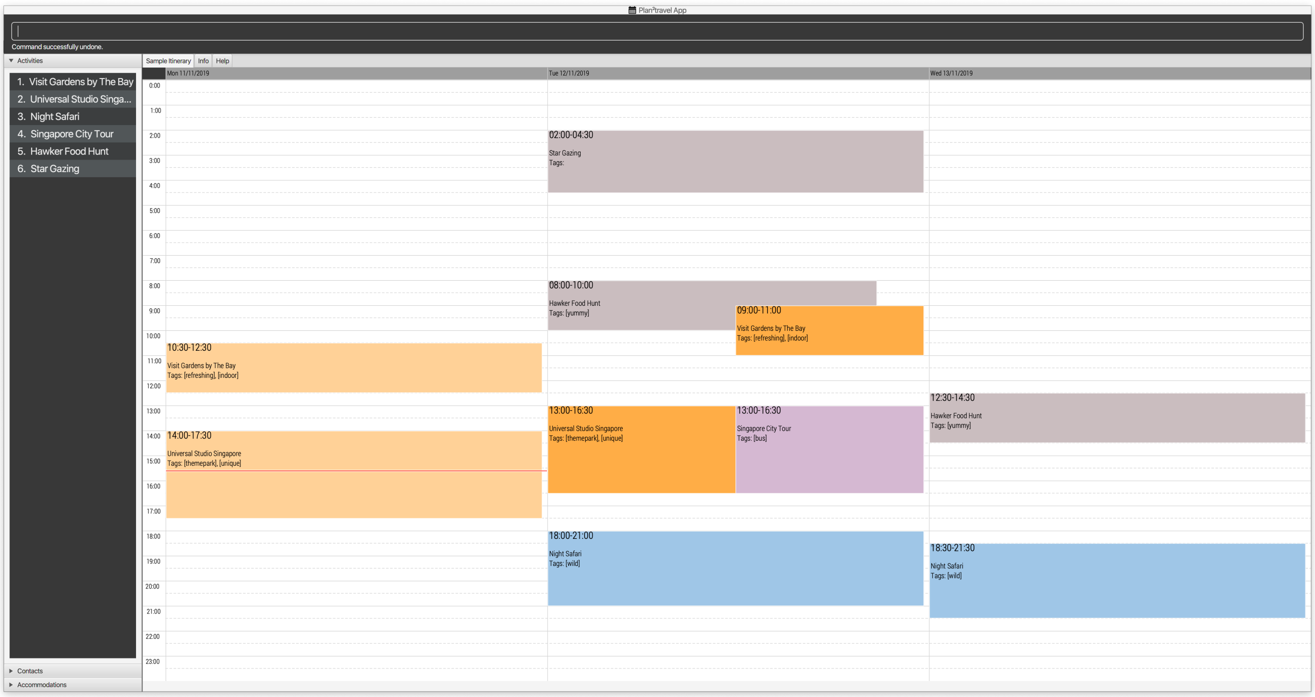

Day has been optimised!
Plan²travel App
Activities
1. Visit Gardens by The Bay
2. Universal Studio Singa...
3. Night Safari
4. Singapore City Tour
5. Hawker Food Hunt
6. Star Gazing
Contacts
Accommodations
Sample Itinerary
Info
Help
Mon 11/11/2019
Tue 12/11/2019
Wed 13/11/2019
10:30-12:30 Visit Gardens by The Bay Tags: [refreshing], [indoor]
14:00-17:30 Universal Studio Singapore Tags: [themepark], [unique]
02:00-04:30 Star Gazing Tags:
08:00-10:00 Hawker Food Hunt Tags: [yummy]
13:00-16:30 Singapore City Tour Tags: [bus]
18:00-21:00 Night Safari Tags: [wild]
12:30-14:30 Hawker Food Hunt Tags: [yummy]
18:30-21:30 Night Safari Tags: [wild]

Figure 2. Day 2 has been optimised, no overlaps.

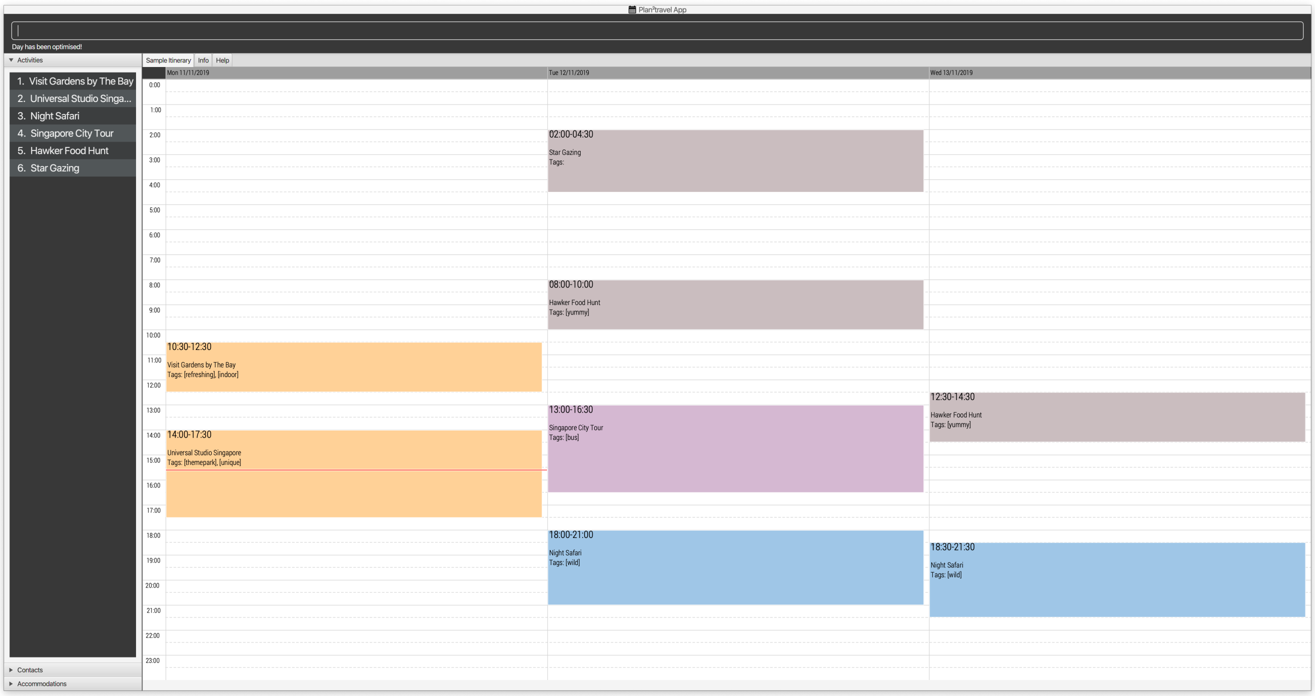
- After undo


Plan²travel App
Activities
1. Visit Gardens by The Bay
2. Universal Studio Singa...
3. Night Safari
4. Singapore City Tour
5. Hawker Food Hunt
6. Star Gazing
Contacts
Accommodations
Sample Itinerary
Info
Help
Mon 11/11/2019
Tue 12/11/2019
Wed 13/11/2019
02:00-04:30 Star Gazing Tags:
08:00-10:00 Hawker Food Hunt Tags: [yummy]
09:00-11:00 Visit Gardens by The Bay Tags: [refreshing], [indoor]
10:30-12:30 Visit Gardens by The Bay Tags: [refreshing], [indoor]
12:30-14:30 Hawker Food Hunt Tags: [yummy]
13:00-16:30 Universal Studio Singapore Tags: [themepark], [unique]
13:00-16:30 Singapore City Tour Tags: [bus]
14:00-17:30 Universal Studio Singapore Tags: [themepark], [unique]
18:00-21:00 Night Safari Tags: [wild]
18:30-21:30 Night Safari Tags: [wild]

Command successfully undone.

Figure 3. Optimise command successfully undone

- After redo


Plan²travel App
Activities
1. Visit Gardens by The Bay
2. Universal Studio Singa...
3. Night Safari
4. Singapore City Tour
5. Hawker Food Hunt
6. Star Gazing
Contacts
Accommodations
Sample Itinerary
Info
Help
Mon 11/11/2019
Tue 12/11/2019
Wed 13/11/2019
02:00-04:30 Star Gazing Tags:
08:00-10:00 Hawker Food Hunt Tags: [yummy]
10:30-12:30 Visit Gardens by The Bay Tags: [refreshing], [indoor]
12:30-14:30 Hawker Food Hunt Tags: [yummy]
13:00-16:30 Singapore City Tour Tags: [bus]
14:00-17:30 Universal Studio Singapore Tags: [themepark], [unique]
18:00-21:00 Night Safari Tags: [wild]
18:30-21:30 Night Safari Tags: [wild]

Day has been optimised!

Figure 4. Optimise command successfully redone, no overlaps.

4. 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.

4.1. Undo/Redo feature

The `undo` command allows user to undo by one command (if the command is `undoable`). The redo command allows user to return to the original state before latest undo.

4.1.1. Implementation

The undo/redo feature utilizes various classes to operate, such as `CommandHistory` class and the classes within the `events package` in the logic component.

CommandHistory is a `static` class that contains the undoEventStack and redoEventStack of the application, each containing Events.

KEY IDEA:

Event — Every UndoableCommand can be wrapped into its own unique Event.
undo/redo — Every Event has an undo and redo method.
EventFactory — An Event object is generated by the EventFactory when executing the UndoableCommand.

List of UndoableCommand:

add activity/ accommodation/ contact/ days
delete activity/ accommodation/ contact/ day
edit activity/ accommodation/ contact
schedule
unschedule
autoschedule
optimise
clear

Step 1. The user executes an `UndoableCommand`.

Step 2. The UndoableCommand is executed, generating an `Event` in the process.

Step 3. `EventFactory` will parse the `UndoableCommand` to generate an Event.
(eg. DeleteActivityCommand will result in DeleteActivityEvent generated)

NOTE EventFactory is a static class that will parse an UndoableCommand and generate the corresponding Event.

Step 4. Event is added to `undoEventStack` stored in CommandHistory. The `redoEventStack` in CommandHistory is also `cleared` upon generating a new Event.

Step 5. The UndoableCommand has been executed, returning a `CommandResult` to be shown.

Step 6. To undo the previous UndoableCommand, the user executes `undo` command. An `UndoCommand` is generated.

Step 7. `UndoCommand` is `not` an `UndoableCommand`. Executing the UndoCommand gets the Event from the top of undoEventStack and calls the undo method of `Event`.

NOTE	Both UndoCommand and RedoCommand are not UndoableCommands, no Events are generated.

Step 8. The `Event` is popped from the undoEventStack and pushed to redoEventStack in CommandHistory. A CommandResult is returned and the Event is `undone`.

Step 9. To redo the command that has been undone, the user executes `redo` command. A `RedoCommand` is generated and it is `not` an `UndoableCommand`. This execution is similar to steps `6` and `7`, except Event is popped from redo stack instead and pushed to undo stack.

The **following two sequence diagrams** shows how the user's input is handled for `'delete activity 1'`.

The first diagram below shows the execution of `'delete activity 1'` command. It shows how the `Event` is generated and how the `undoEventStack and redoEventStack is updated`.

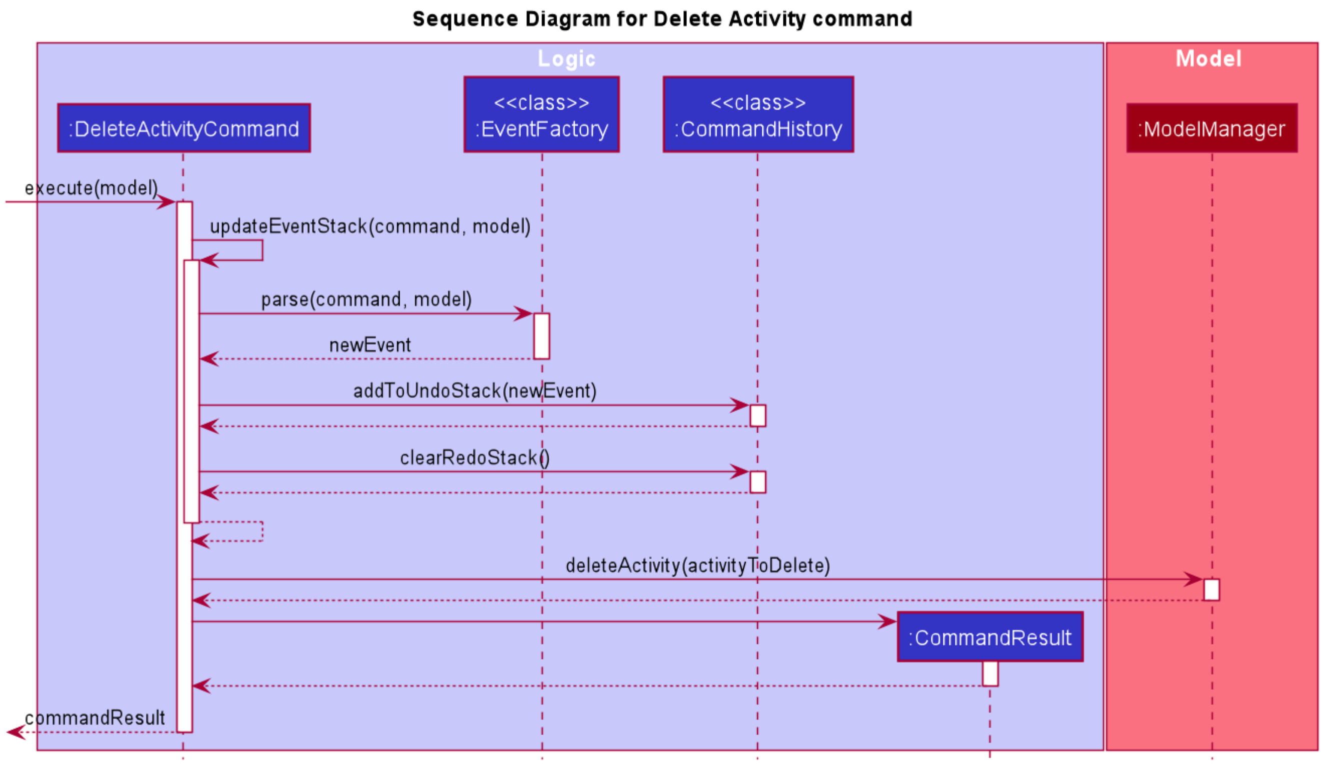


Figure 5. Executing delete activity

NOTE	When DeleteActivityCommand is executed, it generates the `activityToDelete` by extracting the `Activity` to be deleted from the Model's list of activities based on the index specified in user's command input. It then calls the Model's `deleteActivity` method.

The second diagram below shows the execution of `undo` command. Executing `UndoCommand` calls undo of the DeleteActivityEvent, which returns an AddActivityCommand with the Activity (initially deleted) to be added back at the correct index. Both Activity and Index were stored in DeleteActivityEvent.

This new AddActivityCommand is executed, and the DeleteActivityCommand is successfully undone.

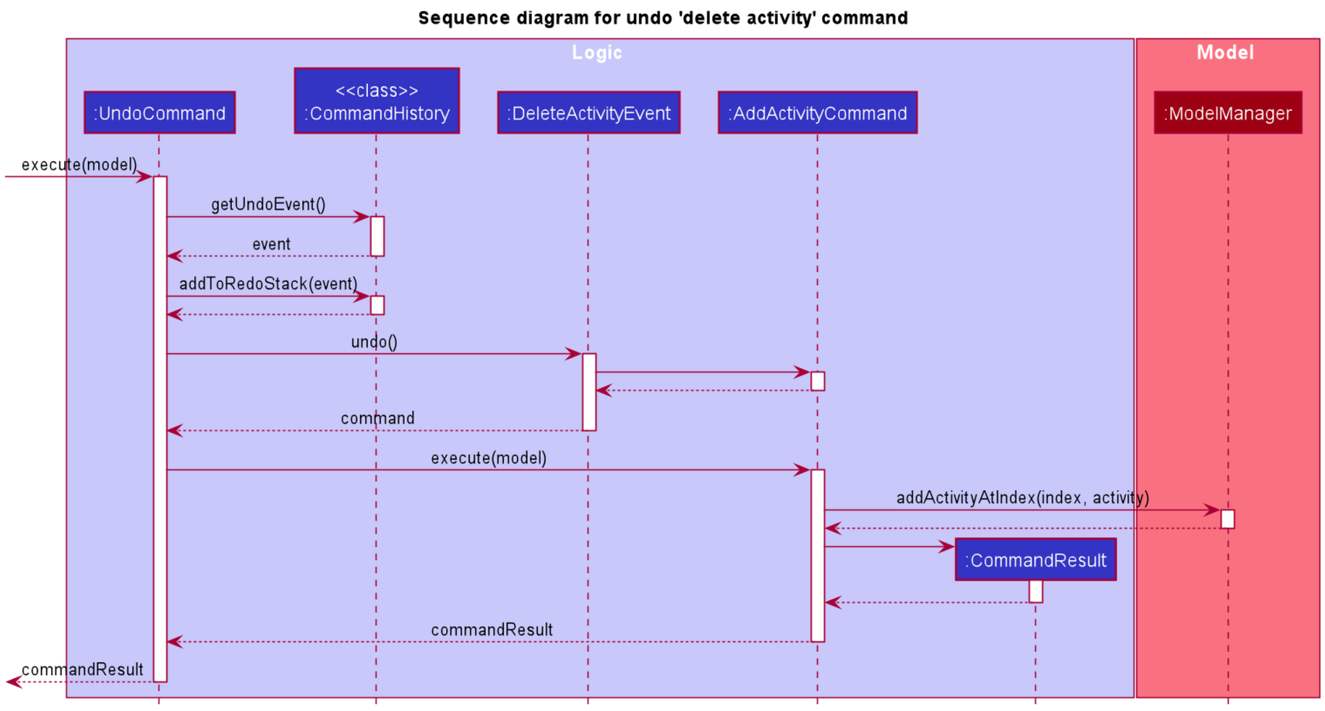
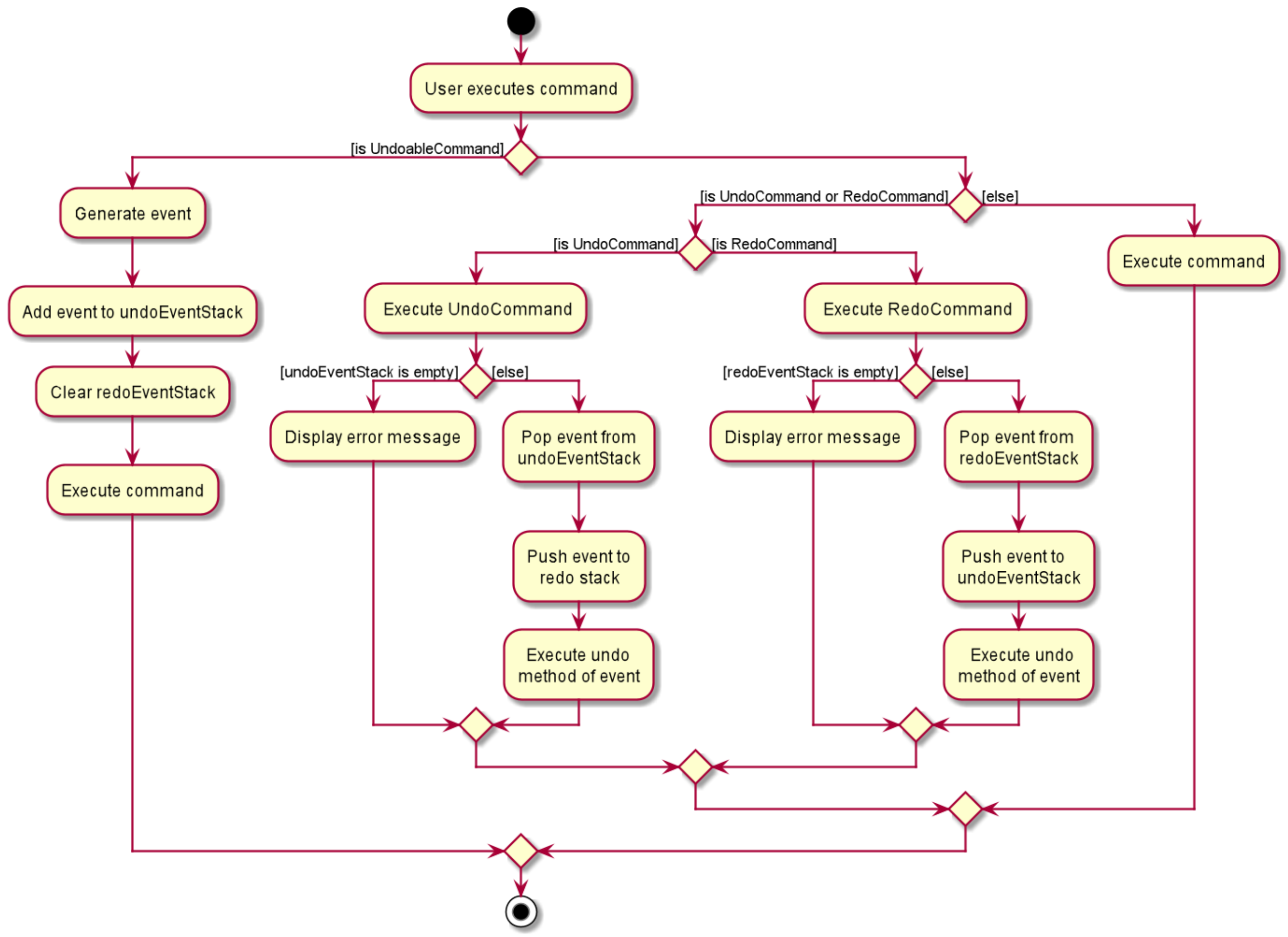


Figure 6. Executing undo for 'delete activity'

NOTE Each Event stores the necessary data required by the reverse Command to undo the effects of the initial UndoableCommand.
(eg. AddActivityEvent stores the Activity added, as DeleteActivityCommand requires the Activity to undo the initial AddActivityCommand□s changes)

The following **activity diagram** summarizes what happens when a user executes an UndoableCommand, an UndoCommand, a RedoCommand, or any other Commands.



NOTE If the user does not execute an UndoableCommand, UndoCommand or RedoCommand, the stack of Events in CommandHistory will **not be affected**. (eg. view, list commands)

4.1.2. Design Considerations

Aspect: How undo & redo executes

- **Option 1 :** Wrap every UndoableCommand in an Event class, which has undo and redo methods.
  - Pros:
    - Uses less memory by storing Event objects rather than storing every state of the Model.
    - Convenient for future extensions for new Commands added. Just need to ensure for every UndoableCommand, there must be a Command that is able to undo its changes.
    - Command classes obey Single Responsibility Principle, they do not need to know how to undo or redo itself, as it is abstracted to their corresponding Event classes.
  - Cons:
    - Every UndoableCommand requires another Command to undo its changes. Might be difficult to manage if more UndoableCommands are added.
- **Option 2 :** Saves the entire Model data (comprising of accommodations, activities, contacts and days).

- Pros:
  - Easy to implement.
- Cons:
  - May have performance issues in terms of memory usage. Expensive to store the various objects at every state.

Aspect: Data structure to support the undo/redo commands

- **Option 1 (current choice):** Use of static class CommandHistory to store a stack of Events to undo, and a stack of Events to redo.
  - Pros:
    - Easy to implement and understand. Every Event object is generated through EventFactory and stored in CommandHistory.
  - Cons:
    - Static class is used instead of Singleton implementation. No single instance of CommandHistory is created, cannot be passed as a parameter to other methods and treated as a normal object, hence might pose a difficulty during extensions.
- **Option 2:** Create a HistoryManager class to store a single list of Model objects for undo/redo
  - Pros:
    - Straightforward and easy implementation, storing a deep copy of Model whenever an UndoableCommand is executed.
  - Cons:
    - Need to keep track of the Model object obtained from the list to set during Undo/Redo. Difficult to manage the pointer in the list.