Karan Dev Sapra - Project Portfolio for *TravelPal*

About Our Project

This project portfolio page documents my contribution to the team project *TravelPal* which I undertook along with 4 other computer science students. This application , which is an alteration of another pre-existing basic command line interace desktop addressbook application , is an all-in-one holiday planner that enables travellers to meticulously plan their travels including features like expenditure and inventory managing.

Due to our project requirements, TravelPal is primarily for users who prefer to use a Command Line Interface (CLI), rather than the more traditional way of using a mouse. The display of our application, however, encompasses a Graphical User Interface (GUI) for easy readability and navigability.

Moreover, our project specification defines our target user to be students who are travel enthusiasts.

Summary of contributions

This section details my contribution to the TravelPal application, as well as a summary of my code implementation and the value adddition it has to the project in terms of functionalitiy and usefullness.

Major enhancements: Implementation of the *Trip Inventory Manager* feature

- **What it does**: The *Trip Inventory Manager* allows users to create a list of inventory items they need on their trip. Users can choose to manually add a new inventory item or add an inventory item inside of an event in the trip itinerary - which will automatically be reflected on the Trip Inventory List. If the item has already been purhcashed or packed by the user, he is able to mark it using a checkbox.
- **Justification**: As it is very common for people to forget things before a trip, this feature prevents them from being forgetful. Moreoever, since the inventory items you store in an event are seen in the Trip Inventory Manager - whenever a user is planning his trip's events - at the same time he can take a note of the items he would need to bring for the event. Furthermore, since the target users are travel enthusiasits, they are more prone to travel their trips more meitulously - and hence this feature would be a good use case for them.
- **Highlights**: This modification requires understanding and interaction of other features such as the event feature since every event stores its own inventory List. Therefore, I had to be mindful to design all my classes with OOP (Object Oriented Programming) principles such as the SOLID principles (mnemonic acronym for five important OOP design principles) to prevent bugs.
- **Credits**: I had read the Developer Guide and code of the address book application to modify and

implement my features.

Second enhancements: Preventing Duplicate Inventory Items.

One of the most important contributions of mine was in preventing the Travel Inventory Manager from showing any duplicate inventory items.

- **What it does**: If the Travel Inventory Manager already contains an inventory with a specific name, the Travel Inventory Manager should not show any duplicates (of the same name).
- **Justification**: Having multiplate duplicate inventory items would make the *Travel Inventory Manager* difficult to read and would make packing difficult. Therefore, this feature was essential to implement to maintain readibility and organization.
- **Highlights**: As each event stores an inventory list, the same name could appear in different events (since the events are not supposed to cross check amongst each other). Therefore, I had to spend a lot of time thinking of a workable solution to only delete the inventory item name from the Travel Inventory Manager if no more events had an inventory item of that name in them. I finally thought of a solution which involves keeping count of the number of Events that have a particular inventory item name, for all names in the *Travel Inventory Manager*.

Other Contributions

- Code contributed: [RepoSense code][Github pull requests]
- Idea generation:
  - Contributing in developing some of the main idea of *TravelPal* (expenses, currency, inventory, events etc.) during the project's inception. (see travelpal idea generation google docs)
- Enhancements to existing features:
  - Wrote test cases for *Inventory Manager*
  - Updated the UI to make it more interactive and user friendly #171
- Project management:
  - Reviewed Pull Requests: [My reviews]
  - Updated ContactUs page #16
  - Created first draft and initial layout and plan of the User Guide #17

Contributions to the User Guide

This section displays the section of User Guide relating to the usage of the *Trip Inventory Manager* which was created and edited by me.

Trip Inventory Manager

Introduction

TravelPal's *Trip Inventory Manager* is an integrated inventory planning and management system. It displays all the inventory items required for your trip in an intuitive and simple manner. Thus, making packing for your next trip a super easy task!

There are two types of inventory items:

1. Event-specific inventory items (auto-generated from event)
2. Non-event-specific inventory items

This section of the user guide explains how to view and manage your inventory items using *Trip Inventory Manager*.

User Interface Overview

Shown below is the *Trip Inventory Manager*.

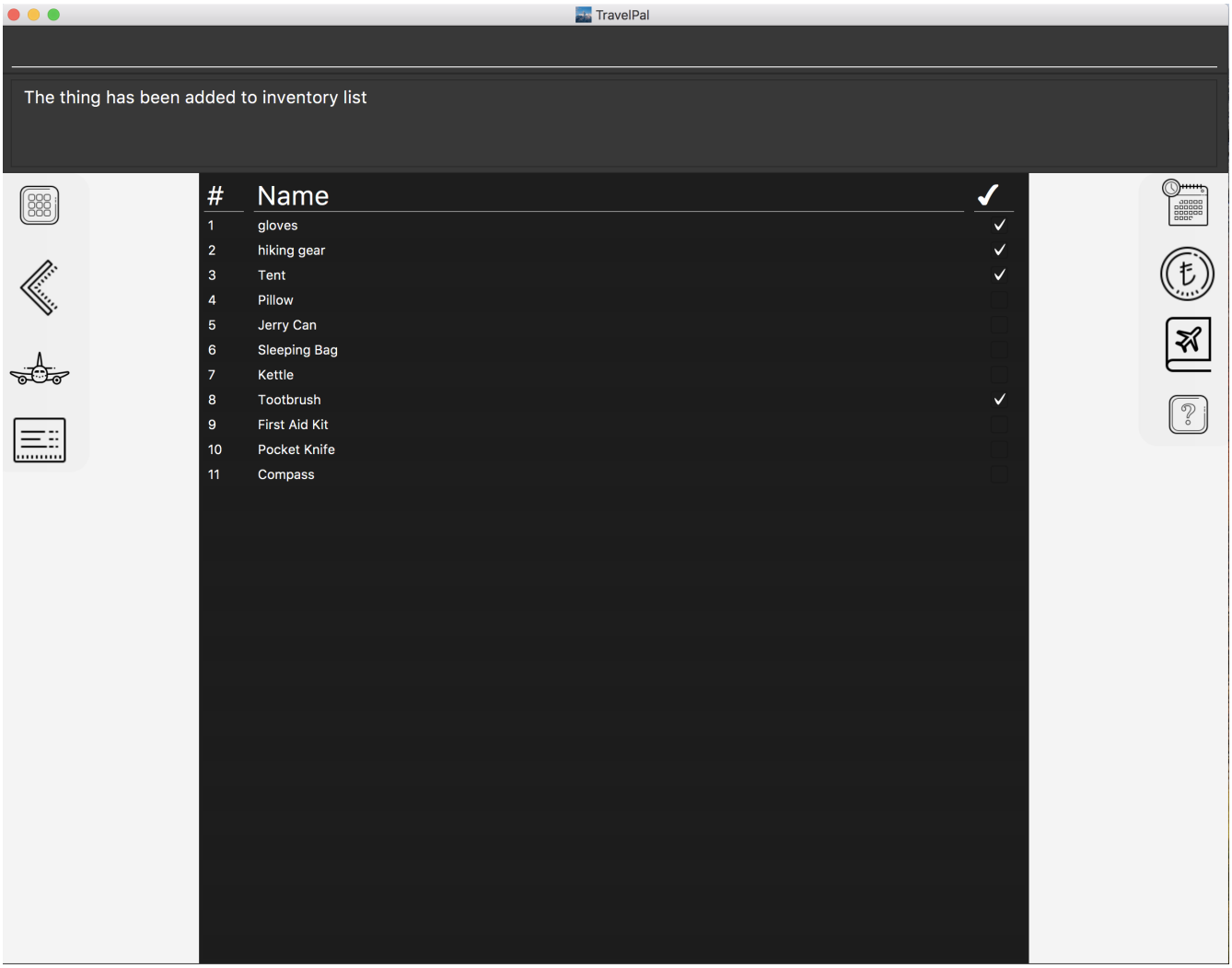

Figure 1. Overview of Trip Inventory Manager user interface

The *Trip Inventory Manger* displays all your event-specific inventory items and non-even-specific inventory items in one table. To provide the best reading experience, *Trip Inventory Manger* does not display duplicate names.

Commands

On the *Trip Inventory Manager* page, the following command are available:

- `add <name>`: Adds a non-event-specfic inventory item with the name, <name>.

> **NOTE** <name> must not already exist in the *Trip Inventory Manger*

- `delete <index of inventory item>`: deletes the inventory item

> **NOTE** The inventory item to delete must exclusively be a non-event-specific inventory item. Otherwise, to delete it, you will have to delete the inventory item from all events it is inside.

- `check <index of inventory item>`: marks the inventory item as packed
- `uncheck <index of inventory item>`: marks the inventory item as not yet packed

> **NOTE** See the user guide for usage scenarios which are not mentioned in this project portfolio page.

Contributions to the Developer Guide

This section of the Developer Guide titled *The Inventory List* (which details the its' Models, User Interface and Logic) was created and edited by me.

[Inventory] The Inventory List

In the application, there can be two types of Inventory Lists:

1. Overall Trip Inventory List

An overall trip inventory list is the inventory list of all the items the user needs for the trip.

Each trip only has **one** overall trip inventory list.

2. An Event inventory list

An event inventory list is the inventory list of all the items the user needs for the event.

Each trip can have **mutiple** event inventory lists.

Therefore, in a trip, all the events inventory lists are subsets of one overall trip inventory list.

Aspect : Model

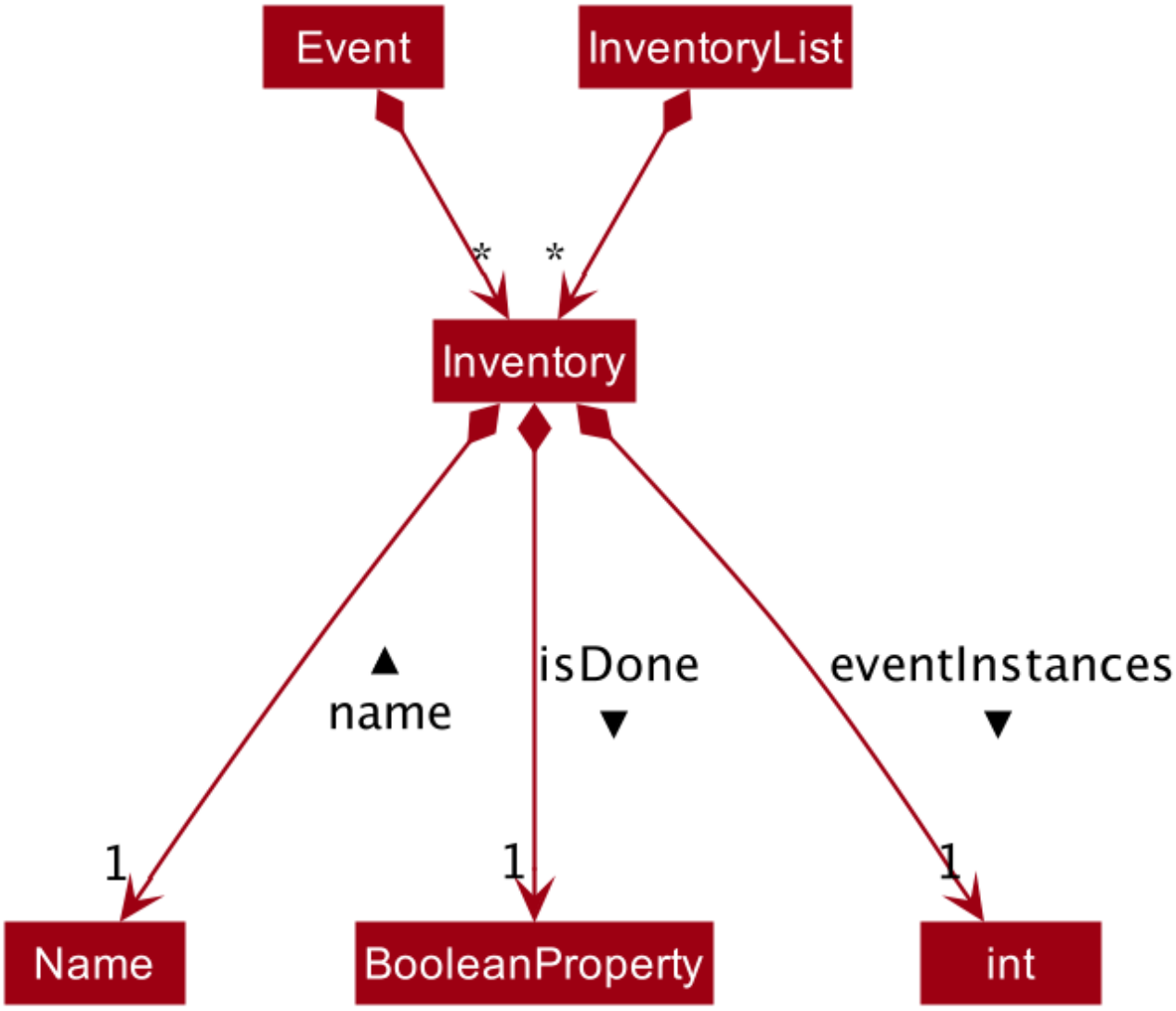

Figure 2. Class diagram of a 'InventoryList' and an `Event`, and its contained models

Inventory

The model for an inventory item stored in memory is stored in the `Inventory` class.

It contains three attributes: the `name`, `isDone` and `eventOccurances`. The `name` attribute is implemented using a custom built `Name` class which ensures that the `name` is valid. `isDone` was implemented using a BooleanProperty and indicates whether the inventory item has been packed. While, `eventInstances` is an integer which represents the total number of `Event` models that have the same inventory item.

Event

`Inventory` models are contained within an `Event`, if the corresponding inventory items are part of an event. `Event` stores the `Inventory` models in an `ArrayList`. Therefore, each `Event` model contains an Event Inventory List (as defined above).

Since, currently, the application does allow the user to alter the 'isDone' toggle of an `Inventory` model contained in an event, the `isDone` attribute is never set once it is is initialised (as `false`).

Furthermore, as the `eventInstances` attribute is only relevant to the Overall Trip Inventory List, the

`eventInstances` attribute is set as `-1`.

InventoryList

All unique (by name) `Inventory` models in any `Event` model is contained within an `InventoryList`. It stores the inventory items in a JavaFX `ObservableList`. Therefore, the `InventoryList` contains the Overall Trip Inventory List (as defined above).

An `ObservableList` was chosen as the user interface requires the changes in the inventory list to be displayed instantly. Furthermore, `isDone` was implemented using a `BooleanProperty` instead of a simple `boolean` to instantly detect and display changes in any inventory item's `isDone` value.

Additionally, an `InventoryList` can also contain miscellaneous inventory items (not contained in any event).

Aspect : User Interface of Overall Trip Inventory List (with Model `InventoryList`)

The Overall Trip Inventory List (with Model `InventoryList`) is displayed in the `TableView<Inventory>` of the `InventoryPage`, as seen in the object diagram:

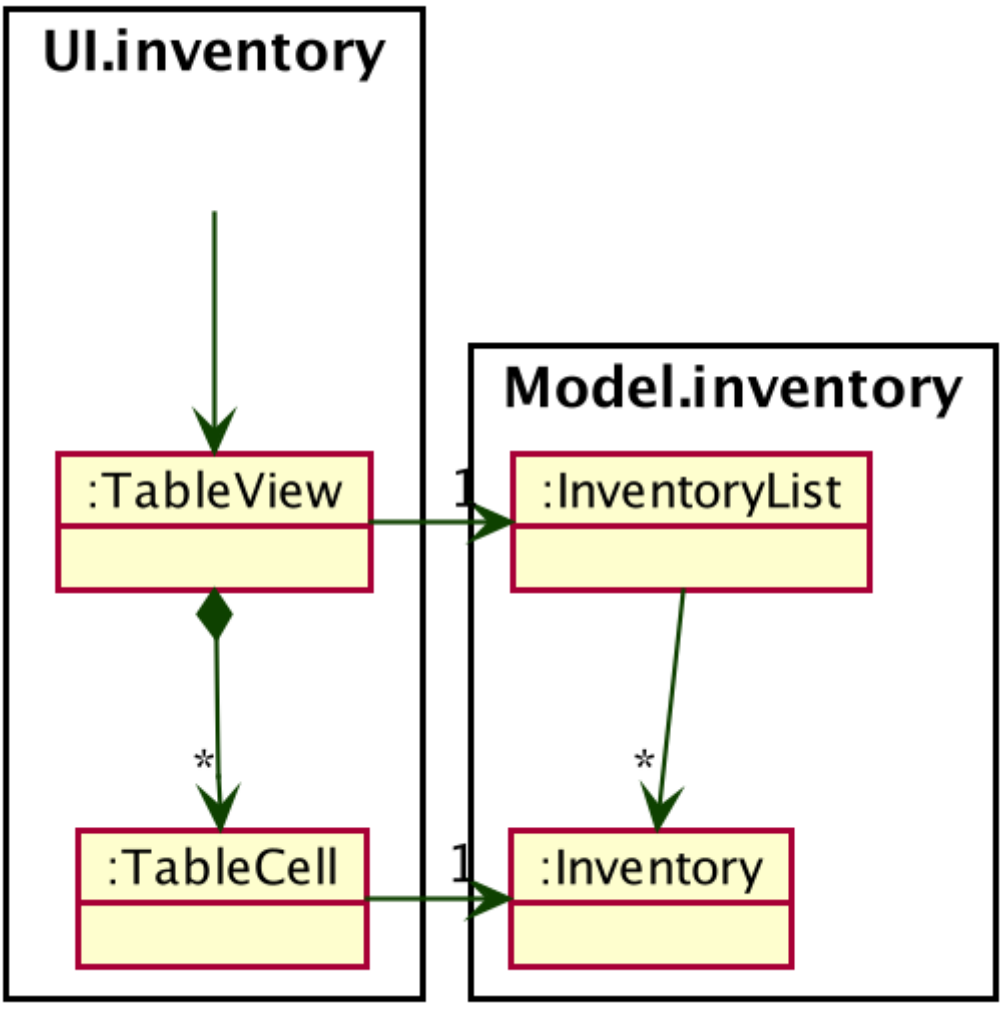

Figure 3. Object diagram of table view component, as contained by `InventoryPage` (not shown)

The TableView has 3 columns. The first column numbers the items. The second column displays the item's name. And the thrid column displays whether the item's `isDone` property is true or false (through a checkbox).

Both the first and the second column's `cellValueFactory` were set to `new PropertyValueFactory(STRING)` with String `STRING`. However, the third column was set to `CheckBoxTableCell.forTableColumn(BOOLEAN)` with Boolean `BOOLEAN`.

Aspect : User Interface of Event Inventory List (with Model `Event`)

The Event Inventory List (with Model `Event`) is displayed in the `ListView<Inventory>` of the `EventPage`, as seen in the object diagram:

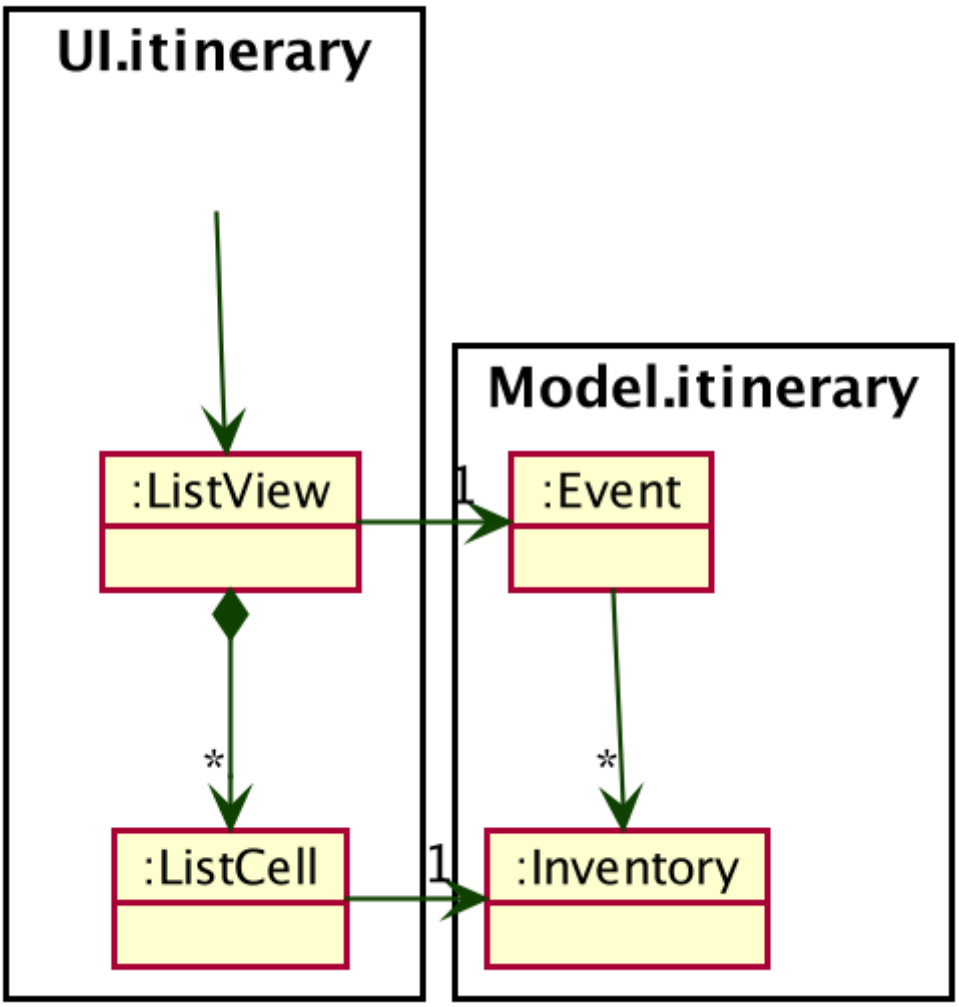

Figure 4. Object diagram of list view component, as contained by `EventPage` (not shown)

The `ListView` uses the CSS styling of `resources/view/Inventory/InventoryListViewTheme.css' by calling 'listView.getStylesheets().add(CSS_FILE_PATH)` with the File Path `CSS_FILE_PATH`)

Furthermore, the reason why `ListView` and `ListCell` are in package `Ui.itinerary` is because `EventPage` is in itinerary `Ui.itinerary`. Similarly, `Event` is in package `Model.itinerary`

Aspect : Logic of adding item to Overall Trip Inventory List (with Model `InventoryList`)

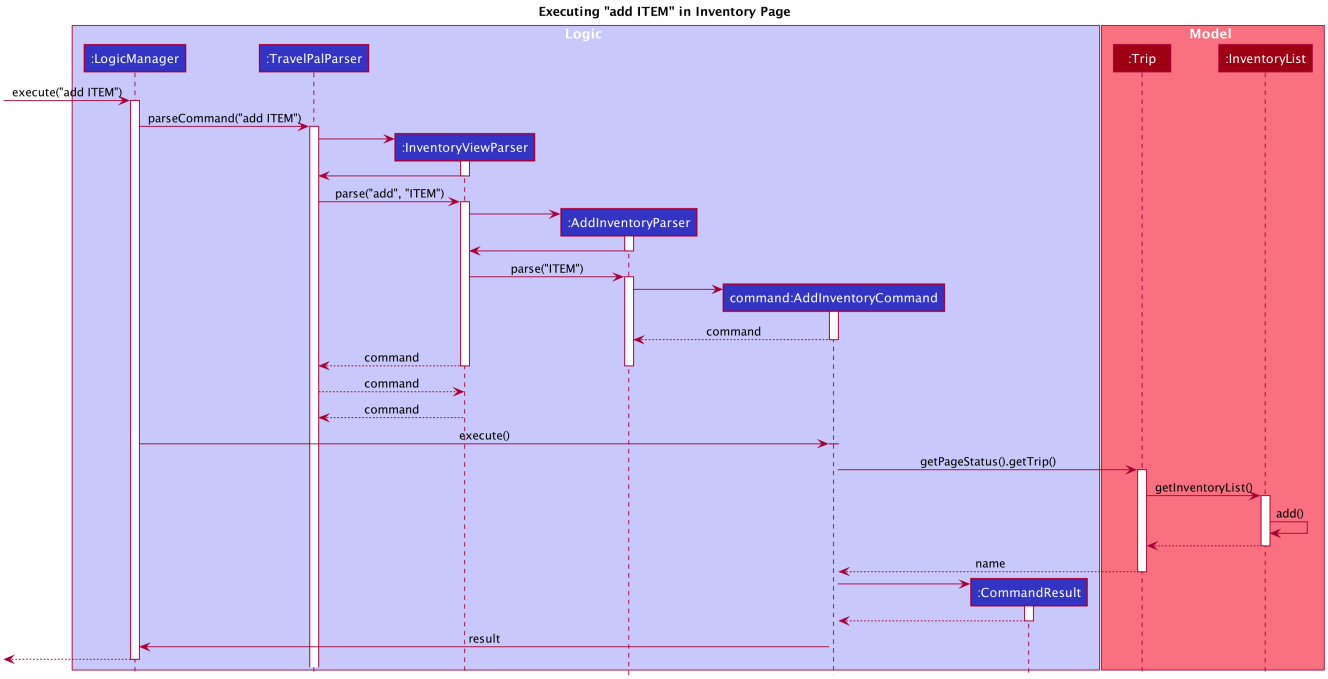

Figure 5. Sequence diagram for adding an item to the Overall Trip Inventory List

From the sequence diagram, it can be observed that only if `InventoryViewParser` and `AddInventoryParser` returns a Command, will the Command be executed. This ensures the AddCommand is only executed when intended.

Moreover, as seen from the sequence diagram, whenever the `AddInventoryCommand` is executed, it will always return the `InventoryView` that corresponds to the Trip that the user is correctly on - ensuring that there is only one Overall Trip Inventory List.

Furthermore, the `InventoryList` calls its own internal function of `add` to add an item to its list - which loops through the entire list to make sure that the item to be added does not already exist - otherwise it does not add it . Therefore, this also allows all the items in InventoryList to be unique.

Possible Improvement in Future Release

Currently, both the Event Inventory List and the Overall Trip Inventory List utilise a common `Inventory` class to store inventory items.

However, an inventory item in the Event Inventory List does not need to keep count of the `eventInstances` attribute. Although, the current solution of setting `eventInstances` to -1 allows both classes to share the `Inventory` class, it might not be scalable and there may lead to bugs in the future.

Therefore, in a future update, it would be rational to remove make the `Inventory` class an abstract class with just a `Name` attribute. And have 2 different classes extend it for the 2 different use cases.

Achievements and Learnings

Development *TravelPal* has been my first time working with and modifying a pre-existing brown-field software engineering projects. During the process, I have gained a lot practice in programming and in maintaining object oriented principles. This was also my first time using Git and working on a software engineering project as a team. I feel I have improved a lot through this experience, both personally and as a team player.