Tan Jun Bang - Project Portfolio

PROJECT: Deliveria

Introduction

This document serves as a project portfolio for Deliveria. It summarises the contributes that I have made and the features I have implemented.

Project Overview

Deliveria is a **desktop application** that allows a **delivery manager** to **manage and assign delivery tasks** efficiently. While it consists of a *Graphical User Interface* (GUI) that is user-friendly, Deliveria is **optimized for those who prefer** to work with a *Command Line Interface* (CLI) which allows fast management of the delivery tasks in an organisation.

This is what Deliveria looks like:

[Deliveria GUI label] | *Deliveria_GUI_label.png*

Figure 1. The graphical user interface (GUI) of Deliveria

Summary of contributions

This section shows the enchancements that I have contributed to Deliveria.

- **Major enhancements**
    - **Major enhancement 1**: added **the enhancement to generate delivery task summary in a PDF Document.** (#208)
        - What it does: It generates delivery tasks' information of each driver for the day in PDF.
        - Justification: This feature will save time for delivery mangers to instruct the drivers. It acts as a instruction manual that contains essential information needed for the drivers to execute their tasks.
        - Highlights: This enhancement organises the drivers', customers' and tasks' information in a standardised table format that is easy to understand and refer. An in-depth understanding of iText7 is required to implement this enhancement.
        - Credits: A third-party library, iText7 is heavily utilized to generate the PDF document.
    - **Major enhancement 2**: added **the enhancement to generate delivery orders in a PDF Document.** (#239)
        - What it does: Generates a delivery order layout that encompasses the goods', customers' and company’s information in PDF format.

        - Justification: This feature will save time to create delivery orders which is necessary for every delivery tasks. It contains all the essential information needed to be used as a proof between the delivery company and the receiver, that goods delivered are as per order and accepted in good condition.
        - Highlights: This enhancement allows integration of company's information in the delivery order as a header. Updating of company's information can be achieved through a single command. An in-depth understanding of iText7 is required to implement this enhancement.
        - Credits: A third-party library, iText7 is heavily utilized to generate the PDF document.

- **Minor enhancements**
    - Refactor storage and introduced a `CentralManager` to encapsulates all the data needed to be saved and loaded, which makes it easier to access the data. (#118)
    - Added storage for `Task` and `Driver`. (#91, #113)
    - Added Create, Read, Update, Delete (CRUD) commands for Delivery Tasks. (#91, #100, #120, #121)
- **Code contributed**: [RepoSense]
- **Other contributions**:
    - Project management:
        - Integrated Coveralls for checking test coverage and Codacy for checking code quality: (#218)
        - Managed the User stories in the Github Project Dashboard (StoryBoard)
        - Managed Milestones on Github (Milestones)
    - Enhancements:
        - Introduced `IdManager` to keep track of the unique ID used by each entities. (Task, Customer & Driver) ((#113)
        - Added test cases to `Task` and `TaskList`. #125
        - Added test cases to `SavePdfCommand`, `DoneCommand` and `EditTaskCommand`. (#230)
    - Documentation:
        - Added implementation and diagrams of `AddTaskCommand` and `SavePdfCommand` in developer guide. (#107, #220)
        - Added and updated the diagrams for storage component in the developer guide. (#107, #124)
        - Added add task, delete task, edit task commands usage instructions in user guide. (#107, #124)
    - Community:
        - Reviewed most of my teammates' pull requests (with non-trivial review comments): (#234, #139, #98, #80)
    - Tools:

    - Integrated iText7 library to the project (#208)

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.

Generates Task Summary or Delivery Orders for a specific date in PDF document: `savepdf`

Document Type: `summary`
The `Task Summary` is a summary of delivery tasks that is assigned to each driver for the specific date. Its purpose is for user reference and archive. Refer to PDF Task Summary generated by `savepdf` command. for sample.

Document Type: `order`
The `Delivery Order` is a document that contain goods' information and requires customers' confirmation. Its purpose is to act as a proof between the delivery company and receiver that goods are delivered as per order and accepted in good condition. Refer to PDF Delivery Order generated by `savepdf` command. for sample.

Format: `savepdf [pdf/DOCUMENT TYPE] [dt/DATE]`

- `DATE` format is dd/mm/yyy.
- `DATE` field is OPTIONAL. If date field is not declared, it will take the date of today.
- PDF document will be saved in a folder in the same directory as where you put the deliveria.jar.
    - PDF Task Summary will be saved in `DeliveryTasks` folder as `DeliveryTasks [DATE].pdf`.
    - PDF Delivery Order will be saved in "DeliveryOrders" folder as `DeliveryOrders [DATE].pdf`.
- Use the `update` command to update the company's information displayed in the Delivery Order PDF.

Examples:

- `savepdf pdf/order`
  Saves the delivery orders in PDF format for today.
- `savepdf pdf/summary dt/15/11/2019`
  Saves the task summary in PDF format for 15/11/2019.

Add a delivery task: `addT`

Adds a delivery task to the task manager.
Format: `addT [g/DESCRIPTION OF GOODS] [c/CUSTOMER ID] [dt/DATE OF DELIVERY]`

- DATE OF DELIVERY must be today onwards. Date format: d/M/yyyy.
- All fields are compulsory.

Examples:

- `addT g/100 frozon boxes of red grouper c/13 dt/10/12/2019`
- `addT g/1x washing machine c/10 dt/12/1/2020` ''' ==== Edits a delivery task : `editT`

Edits a existing delivery task in the task manager.
Format: `editT [TASK ID] [g/DESCRIPTION OF GOODS] [c/CUSTOMER ID] [dt/DATE OF DELIVERY]`

- Only indicate fields that you want to change.
- Edited DATE OF DELIVERY must be today onwards. Date format: d/M/yyyy.
- COMPLETED tasks cannot be edited.

Examples:

- `editT 3 g/50 frozen boxes of catfish`
  Edits the description of the task (Task ID: 3) to be `50 frozen boxes of catfish`.
- `editT 5 c/2 dt/10/12/2019`
  Edits the customer and date of delivery of the task (Task ID: 5) to be `Customer (Customer ID: 2)` and `10/12/2019` respectively. ''' ==== Deletes a delivery task / driver / customer: `del`

Deletes a task / driver / customer from its respective managers.
Format: `del [c/CUSTOMER ID]` | `del [t/DRIVER ID]` | `del [t/TASK ID]`

- Cannot delete a **DELIVERY TASK** that is on-going. Remove the driver from the task first before deleting.
- Cannot delete a **DRIVER** that is assigned to a on-going task. Remove the driver from the task first before deleting.

Examples:

- `del t/1`
  Deletes task (Task ID: 1) from the task manager.
- `del d/2`
  Deletes driver (Driver ID: 2) from the driver manager.

Appendix

[DeliveryTasks Pdf Layout] | *DeliveryTasks_Pdf_Layout.png*

Figure 2. PDF Task Summary generated by `savepdf` command.

Figure 3. PDF Delivery Order generated by `savepdf` command.

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.

Storage component

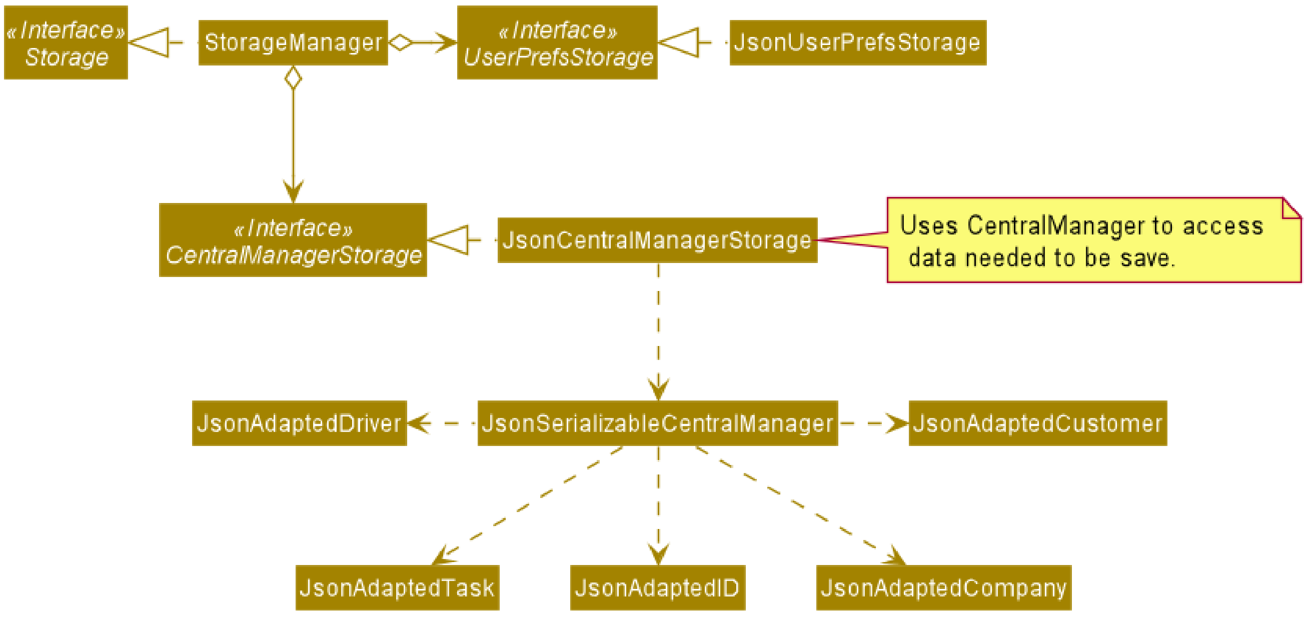
«Interface» Storage
StorageManager
«Interface» UserPrefsStorage
JsonUserPrefsStorage
«Interface» CentralManagerStorage
JsonCentralManagerStorage
Uses CentralManager to access data needed to be save.
JsonAdaptedDriver
JsonSerializableCentralManager
JsonAdaptedCustomer
JsonAdaptedTask
JsonAdaptedID
JsonAdaptedCompany

Figure 4. Structure of the Storage Component

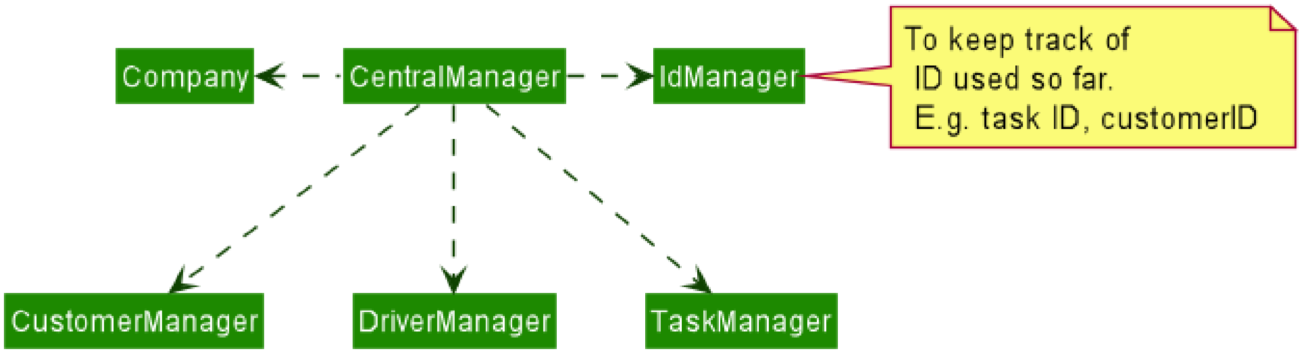

Figure 5. Structure of the CentralManager

API : `Storage.java`

The `Storage` component,

- can save `UserPref` objects in json format and read it back.
- uses `CentralManager` to consolidate all the data that needs to be saved. (e.g. Task Manager’s data)
- can save the `CentralManager` data in json format and read it back.

Task Feature (E.g. Add Delivery Task)

Implementation

The **Add Delivery Task** feature adds a new task into a task list.
It uses the `AddTaskCommand`, which extends `Command`, to add a `Task` into the `TaskManager`. `AddTaskCommandParser` is also utilised to parse and validate the user inputs before sending it to `AddTaskCommand` to execute. 'AddTaskCommand' requires the following fields: `Task`, `customerId`. The attributes of Task is as follows:

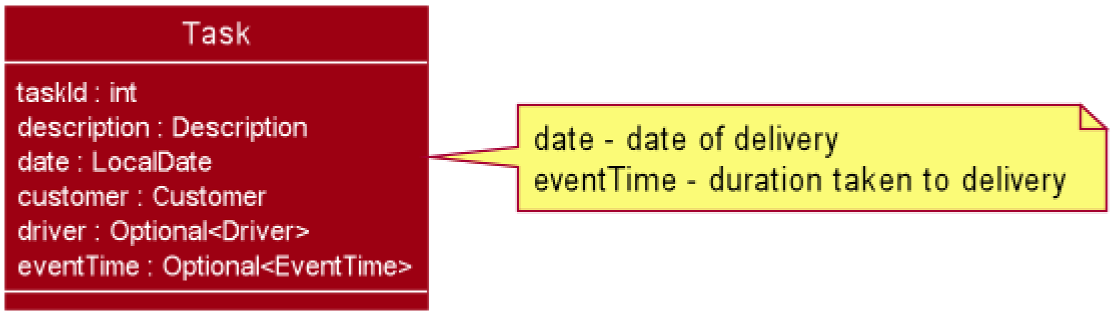
Task
taskId : int
description : Description
date : LocalDate
customer : Customer
driver : Optional<Driver>
eventTime : Optional<EventTime>
date - date of delivery
eventTime - duration taken to delivery

Figure 6. Class Diagram of Task class.

As seen in the above class diagram, `driver` and `eventTime` are optional fields that are not mandatory when adding a task. They will be assigned subsequently using `assign` command. (Refer to Assign feature) The mandatory fields for users are: 'description', 'date' and 'Customer'. After the validation is completed, `AddTaskCommand` will fetch `Customer` using the `customerId` through the `CustomerManager`. A unique id will also be allocated to the task for differentiation.

The following sequence diagrams show how the add task operation works:

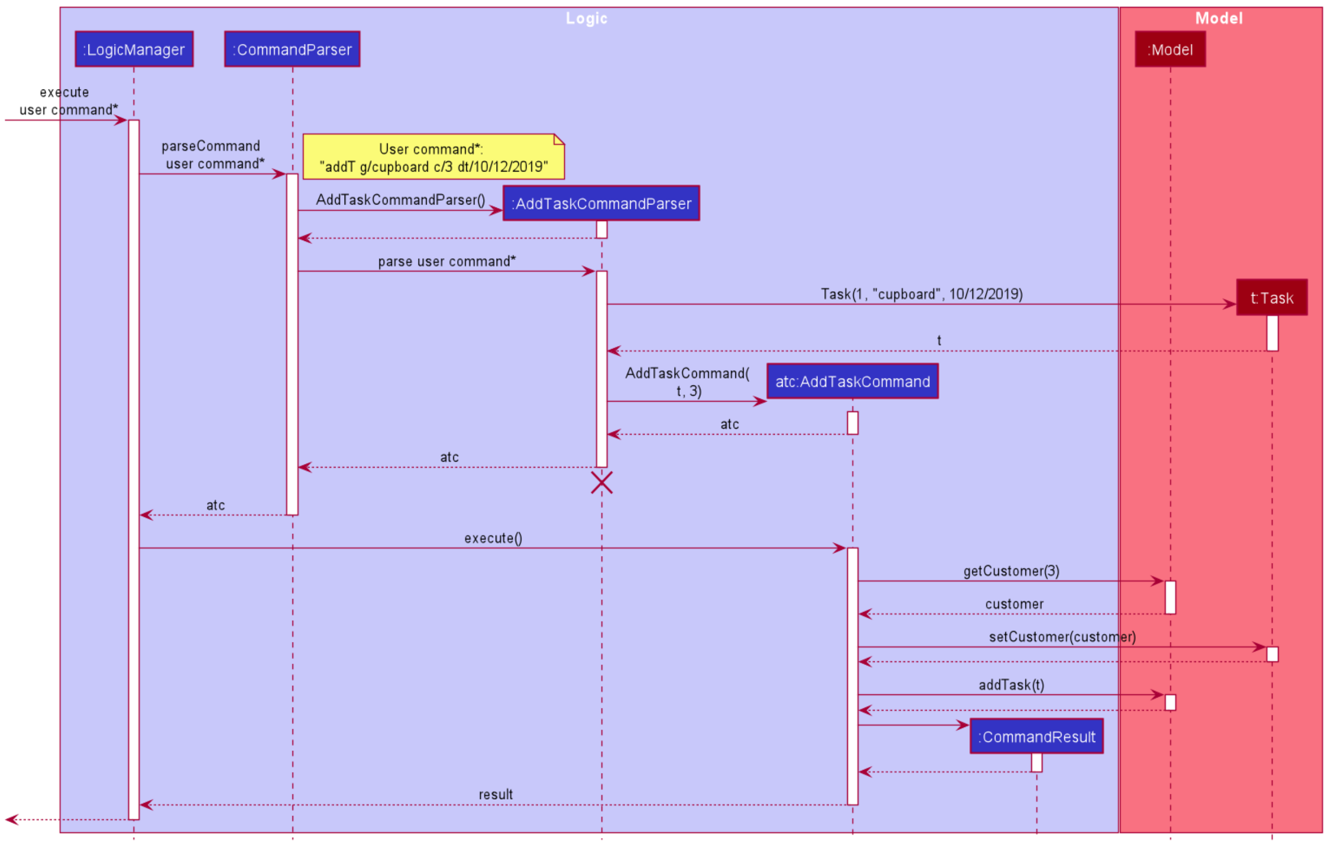

Figure 7. Sequence Diagram of adding a task.

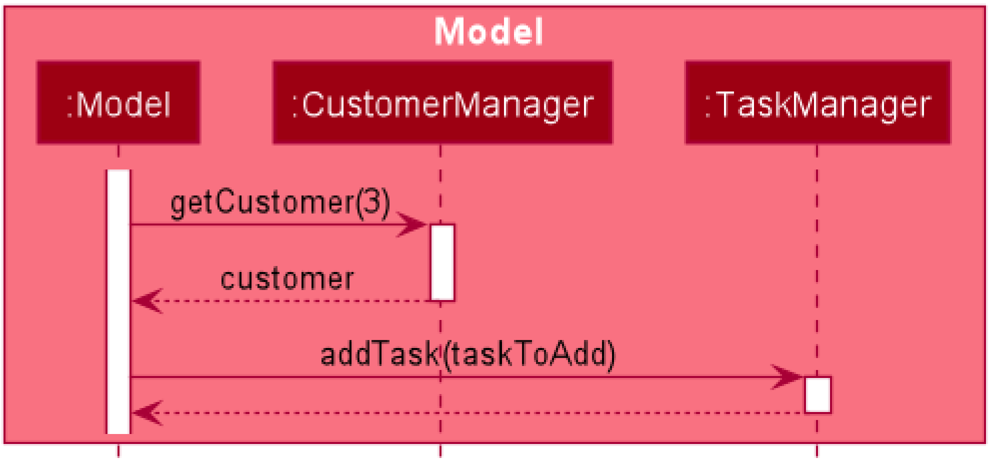

Figure 8. Sequence Diagram of Model interaction with the CustomerManager and TaskManager for adding a task.

NOTE The flow of how the task is being accessed and managed as shown above is the same for other task related command such as edit task command (`editT`) and delete task command (`del`).

Design Considerations

Aspect: Coupling of Task and other entities (Driver and Customer)

- **Alternative 1 (current choice):** Task class contains Driver and Customer classes as attributes.
    - Pros: Centralised Task class that encapsulates all the information, which makes it easy to

manage task.

  - Cons: Task will have to depend on Driver and Customer. Decreases testability.

- **Alternative 2:** Driver and Customer classes have Task class as attribute.
  - Pros: Easy to access tasks through the respective classes. (Driver and Customer classes)
  - Cons: Having 2 classes depend on Task class. Decreases testability.

Task Feature (E.g. Add Delivery Task)

Implementation

The **Add Delivery Task** feature adds a new task into a task list.
It uses the `AddTaskCommand`, which extends `Command`, to add a `Task` into the `TaskManager`. `AddTaskCommandParser` is also utilised to parse and validate the user inputs before sending it to `AddTaskCommand` to execute. 'AddTaskCommand' requires the following fields: `Task`, `customerId`. The attributes of Task is as follows:

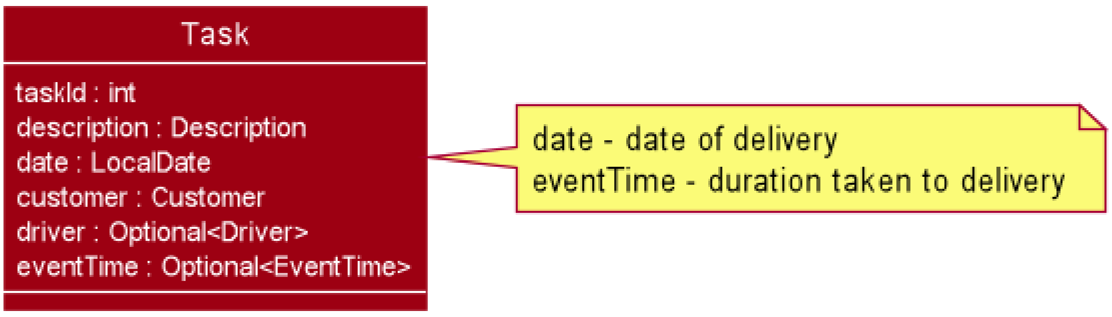
Task
taskId : int
description : Description
date : LocalDate
customer : Customer
driver : Optional<Driver>
eventTime : Optional<EventTime>
date - date of delivery
eventTime - duration taken to delivery

Figure 9. Class Diagram of Task class.

As seen in the above class diagram, `driver` and `eventTime` are optional fields that are not mandatory when adding a task. They will be assigned subsequently using `assign` command. (Refer to Assign feature) The mandatory fields for users are: 'description', 'date' and 'Customer'. After the validation is completed, `AddTaskCommand` will fetch `Customer` using the `customerId` through the `CustomerManager`. A unique id will also be allocated to the task for differentiation.

The following sequence diagrams show how the add task operation works:

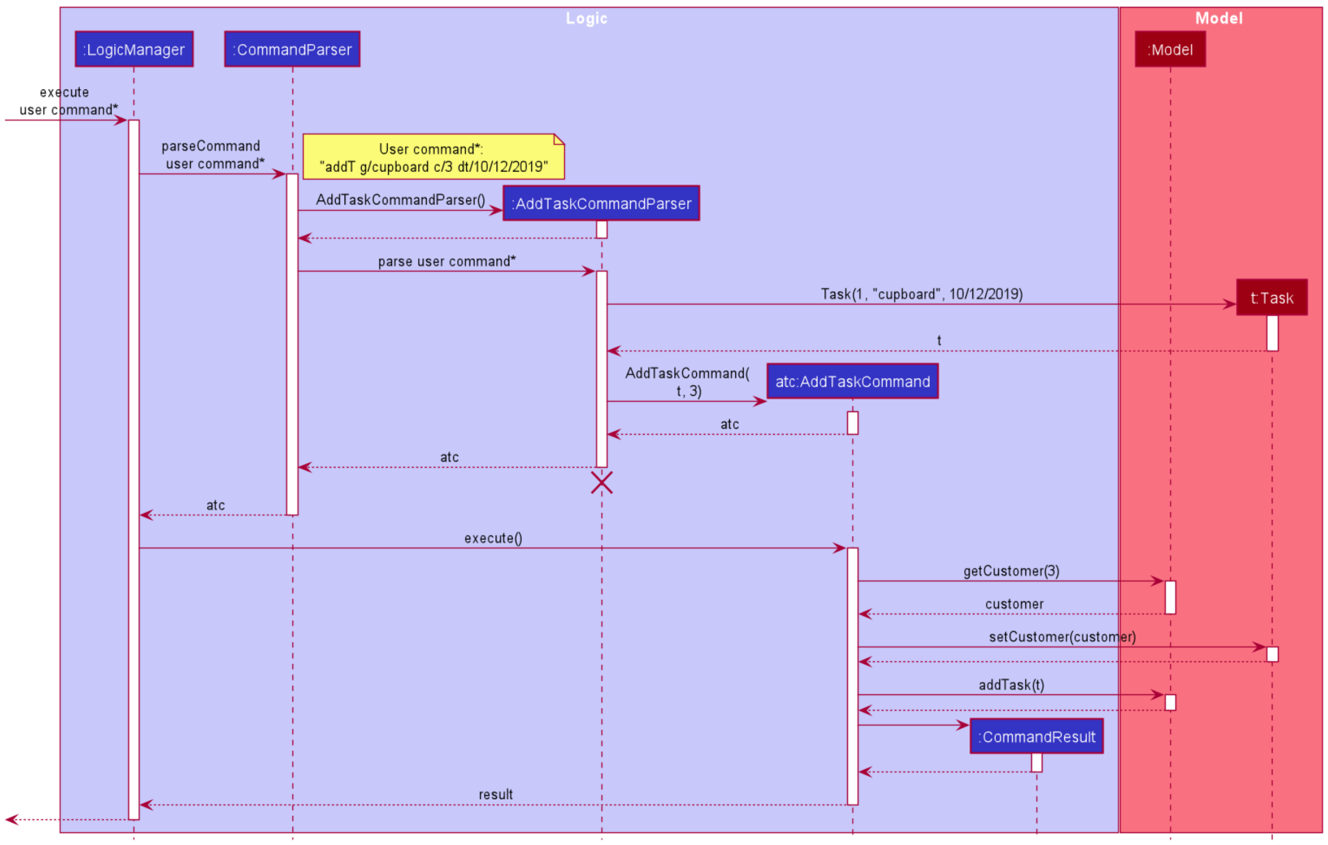

Figure 10. Sequence Diagram of adding a task.

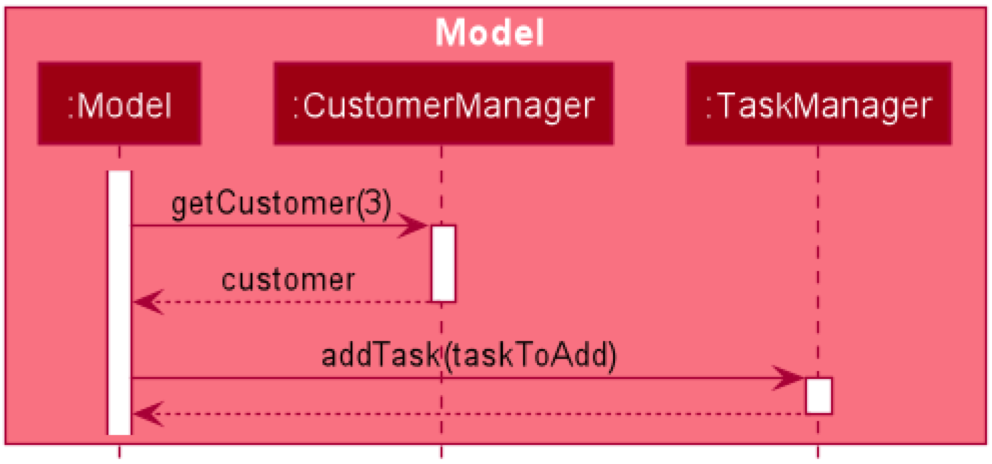

Figure 11. Sequence Diagram of Model interaction with the CustomerManager and TaskManager for adding a task.

NOTE	The flow of how the task is being accessed and managed as shown above is the same for other task related command such as edit task command (`editT`) and delete task command (`del`).

Design Considerations

Aspect: Coupling of Task and other entities (Driver and Customer)

- **Alternative 1 (current choice):** Task class contains Driver and Customer classes as attributes.
  - Pros: Centralised Task class that encapsulates all the information, which makes it easy to

manage task.
    - Cons: Task will have to depend on Driver and Customer. Decreases testability.
- **Alternative 2:** Driver and Customer classes have Task class as attribute.
    - Pros: Easy to access tasks through the respective classes. (Driver and Customer classes)
    - Cons: Having 2 classes depend on Task class. Decreases testability.