He Yingxu - Project Portfolio

PROJECT: iTrackPro

Overview

iTrack Pro is for the grocery shop owners to keep track of the products, customers, and transactions of the shop and have a better understanding of the business by viewing the performance of products and behaviors of customers. It is also able to provide an analysis of the entire business to help the owner manage the shop.

Summary of contributions

- **Major enhancement**:
  1. Added **the ability to manage transcations**, e.g., **addt**, **editt**, **findt**, **listt**, **undot**, **cleart**.
     - What it does: Adding transaction will record a transaction and update the quantity and sales of the associated product. Editting a transaction will edit any attributes of a trancation as well as the quantity/sales attribute of the associate product, if necessary. Finding transaction filters transactions by attributes values, e.g. date time, customer name. Undoing transaction removes a transaction from the system and update the quantity and sales attribute of its product if affected. Listing transaciton shows all the transactions to the screen. Clearing transactions simply wipe out all the transactions in the system.
     - Justification: This feature allows the user to manage the inventory by recording transactinos. By adding a transaction, the system will reduce the quantity of certain products and increase the sales of them. By finding or listing a transaction, the user can view the specific transctions in certain date or purchased by certain person, or just all the transactions. By undoing or editing transactions, users have the ability to correct any mistakes he/she made. Lastly, users might want to clear all transactions only in rare cases, e.g. entering a new financial year or opening a new store.
     - Highlights: Adding/editting/undoing transactions will also update the quantity/sales attributes of related products. E.g. editing a transaction's product will re-store the original product to its previous status and update the quantity & sales of the new product. A future date time in the future is not allowed.
     - Credits: These operations are inspired from the operations of persons in AB3 system.
  2. Added the **ability to plot the quantity sold of a product in a certain time range**.
     - What it does: The command fetches all transactions related with the specified product in a certain time period, then converts the data to daily time series format and plot it in a bar chart.
     - Justification: This feature allows users to understand the popularity of a product visually

in a trend format, such that they could make decisions of whether to stock up or down the product.

- Highlights: If a time range is not specified, the command will consider the last 7 days by default. An end date before start date or date in the future is not allowed.

3. Added the histogram of products to the statistics panel.
    - What is does: The two histograms are passivly displayed on the statistics panel, e.g. they are displayed upon the application is launched. It fetches the quantity and sales of all the products and group them by the amount of quantity/sales. The data is then plot besides the top selling product lists in a line chart format.
    - Justification: It allows the user to understand whether the business is healthy, E.g. whether most of the products have sufficient inventory, is there any super-sales-creator or all the products share the same sales evenly.
    - Highlights: The graph will be updated everytime a user takes any action, e.g. adding a new product.

- **Minor enhancement**:
    - Update the UI from three side-by-side lists to tab panes with transactions representated by rows in a table (Pull request #186).
    - Split the original shared quantity class to TransactionQuantity and ProductQuantity with different minimal values (Pull request #192).
- **Code contributed**: tp-dashboard
    - [model: transaction, model/util: quantity]
    - [logic/command: transaction, logic/command/product: PlotSalesCommand, logic/parser: transaction, logic/parser/product: PlotSalesCommandParser]
    - [storage: tranasction]
    - [ui: transaction, ui: PlotWindow]
    - [test/model: transaction, test/testutil: transaction, test/logic/commands: transaction, test/logic/parser: transaction,]
- **Other contributions**:
    - Project management:
        - Set up the issue labels and milestones with my team mates.
        - Managed releases `v1.3.1` on GitHub
        - Merge PRs created by my team mates.
        - Suggested the team to assign people to each issue and properly tag them with to milestones and PRs as required by the instruction.
    - Enhancements to existing features:
        - Refacotered the process of triggering notification windows from directly generating from the `AddTransactionCommand` to passing signals to command result and making the `MainWindow` to handle the plot. (Commit 803fa01 of Pull request #192)
        - Fixed the problem of re-listing all products everytime adding a transaction (Pull request

#182)

- Fixed the problem of allowing users to input future time. (Pull request #197)
- Refactored the process of checking duplicate input attributes (this commit from Pull request #209)

◦ Documentation:

- Updated the readme from the template of AB3 to customized iTrackPro page (Pull request #)
- Updated the Dev Guide about managing transaction and plot sales command. (Pull request #109, #212)
- Updated some explainatiions in User Guide (#212)

◦ Community:

- Contributed to forum discussions (examples: #58)
- Reviewed the PRs and documentations from other teams in the tutorial together with my team mates(examples: #104, #42, 3)

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.

Plotting sales of a product: `plotsales`

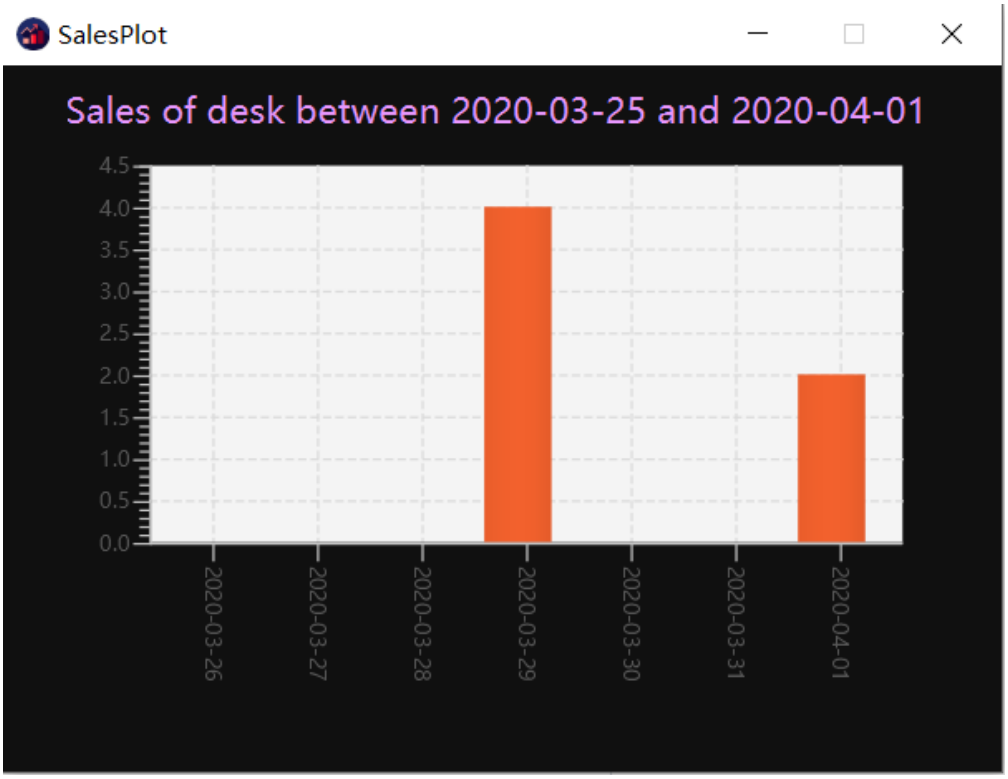
Plots a graph with the sales of the selected product in a given time period.
Format: `plotsales PRODUCT_INDEX [sd/START_DATE] [ed/END_DATE]`

- The start date and end date must follow a format of `yyyy-mm-dd hh:mm`
- The start date must be before or equal to end date
- At least one product must be present

TIP The start date and end date attributes are optional. If omitted, the system will plot the last 7 days by default.

Examples:

- `plotsales 1 sd/2020-02-20 10:00 ed/2020-02-28 10:01`
  Plots a graph with the sales of the selected product between 20th Feb 10am and 28th Feb 10:01am in 2020.
- `plotsales 1`
  Plots a graph with the sales of the selected product in the past week.



Undo a transaction : `undot`

Undo the specified transaction from the system. It allows the user to remove a transaction in case he/she keyed inaccurate information.
Format: `undot INDEX`

- Undos the transaction at the specified `INDEX`.
- The index refers to the index number shown in the displayed transaction list.
- The index must be a positive integer 1, 2, 3, …

NOTE	Why undot instead of deletet? Deletet implies that transaction is only deleted but undot is more fitting as the product details will be modified too.
WARNING	Adds the quantity in the transaction back to the product and reduces the sales of the product by transaction amount.

Examples:

- `listt`
  `undot 2`
  Undo the 2nd transaction in the displayed list.
- `findt dt/2020-01-03 16:00`
  `undot 1`
  Undo the 1st transaction in the results of the find 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.

Add/edit/undo/list/find transactions

The user input is handled by the `MainWindow` class in Ui first, then passed to the `LogicManager` and parsed into `AddTransactionCommand`, `EditTransactionCommand`, etc. Depending on the nature of each command, new transaction or updated transaction will be added to a new index or existing index of the `UniqueTransactionList`, hosted by the `InventorySystem` class. For the `deleteTransactionCommand`, a transaction will be dropped from the `internalList`. Since the `quantity` and `sales` attribute will affect the same attributes of a product, the affiliated `product` will also be edited. In the end, the `filteredTransactionList` of the system will be updated so that the user can view the change accordingly. For the list and find transction commands, the filteredTransactionList will be updated for the UI to interact with users. One command is implemented for each operations in the logic module:

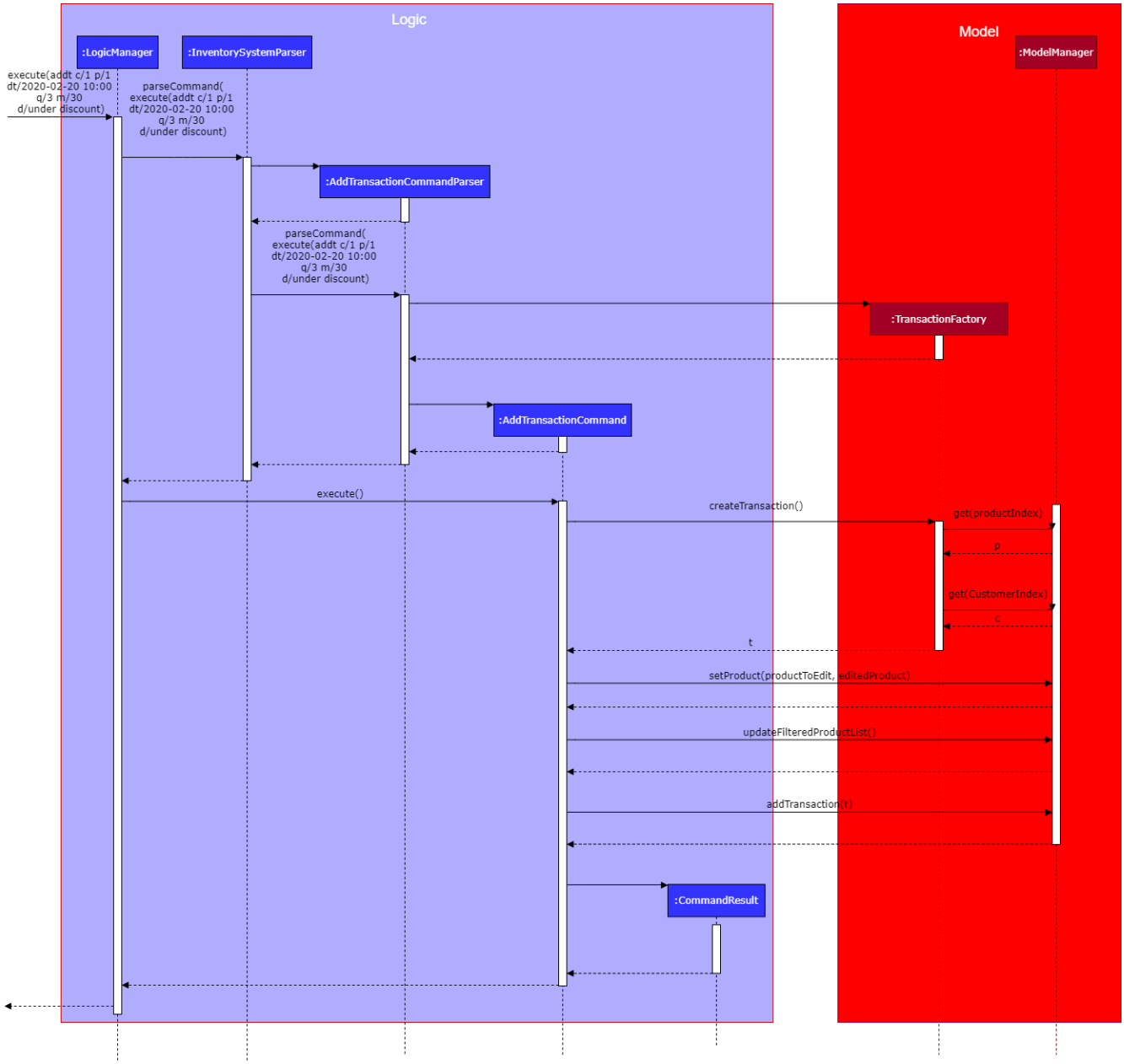
- `AddTransactionCommand` — Adds a transaction into the system and update the the quantity and sales attribute of the corresponding product.
- `EditTransactionCommand` — Edit details of a transaction. If `quantity` is changed, edit the affected product as well.
- `UndoTransactionCommand` — Undo a transaction from the system and edit the affiliated product.
- `ListTransactionCommand` — List all the transaction in the system.
- `FindTransactionCommand` — Find certain transactions by keywords.

For each command, a parser is implemented to parse the input into arguments.

- `AddTransactionCommmandParser` — Parse the add transaction input and generates `AddTransactionCommand`.
- `EditTransactionCommandParser` — Parse the edit transaction input and generates `EditTransactionCommand`.
- `UndoTransactionCommandParser` --Parse the undo transaction input and generates `UndoTransactionCommand`.
- `ListTransactionCommandParser` --Parse the list transaction input and generates `ListTransactionCommand`.
- `FindTransactionCommandParser` --Parse the find transaction input and generates `FindTransactionCommand`.

The following sequence diagram shows how each operation works.

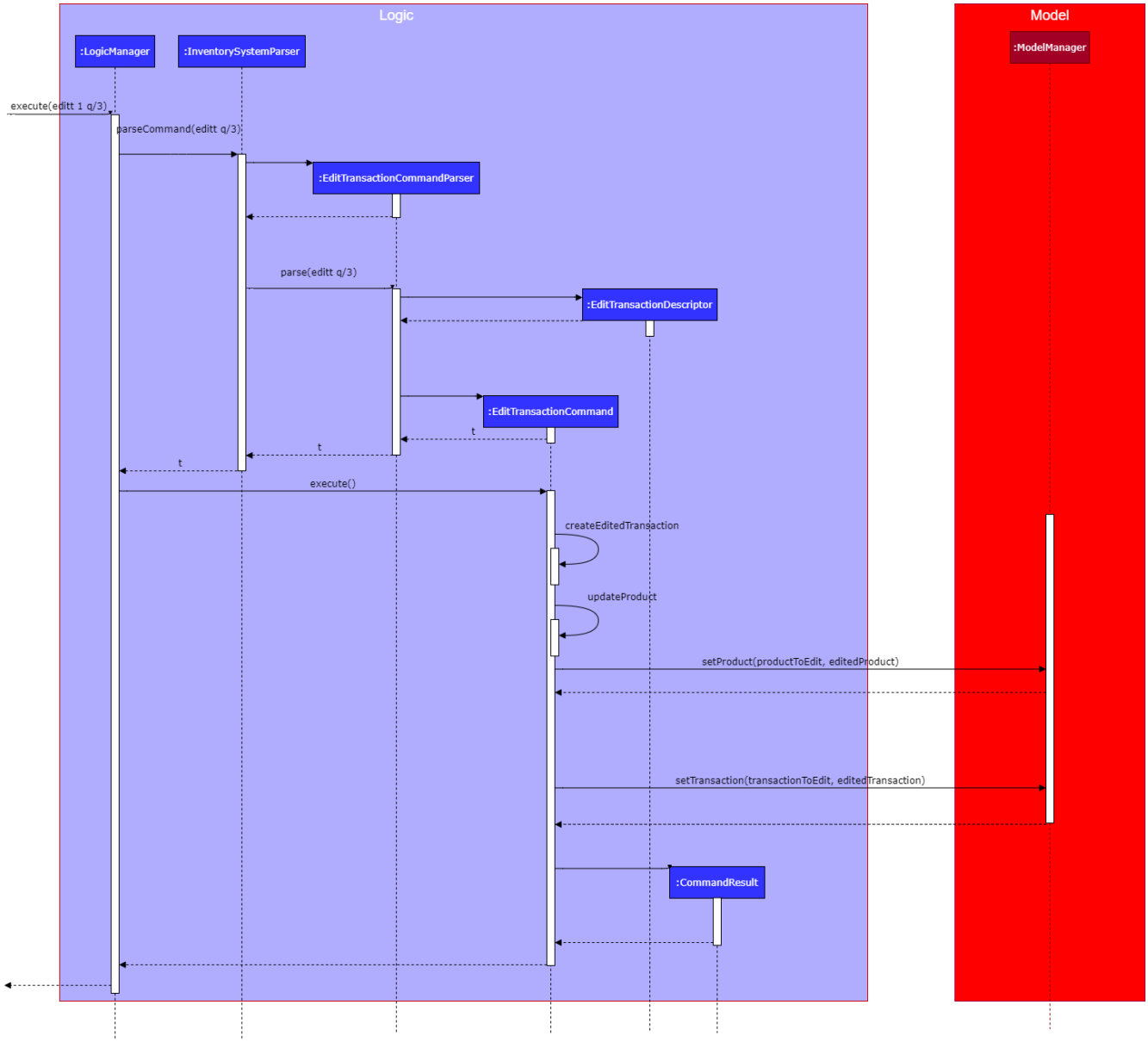
AddTransaction Operation:



NOTE The AddTransactionCommmandParser returns a transactionFacotry with `productIndex` and `customerIndex`, while a transaction is only generated in AddTransactionCommand.

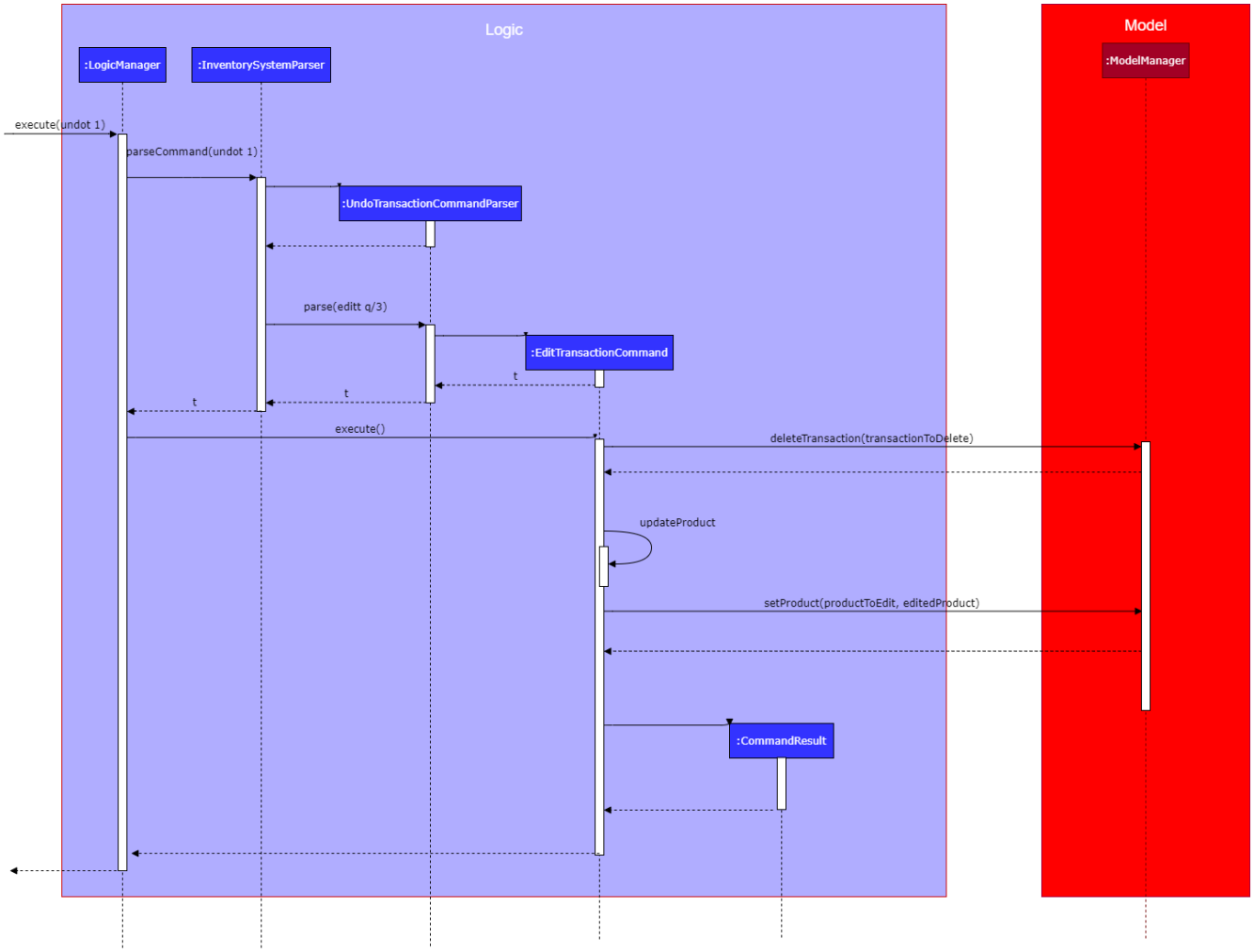
EditTransaction Operation:

Unlike the edit operation of customer and products, editting transaction will trigger another operation of editting its associated product with the new quantity and money. If its product is editted, its quantity will be added back to the original product's quantity, and the new quantity will be deducted from the quantity of the newly referenced product. The update on sales object will be done in the reverse way. In the end, the new transction and product will replace the old ones in the system.



UndoTransaction Operation:

Undoing a transaction will add back its quantity to teh associated product and remove its amount of money from the sales of that product.



ListTransaction/FindTransaction Operation:

The list operation for transaction is the same as that for products and customers.

Design Considerations

Aspect: How to store product & customer in transaction.

- **Alternative 1 (current choice):** Store an unique id and name of the product/ customer.
  - Pros: Do not need to update transaction while product is editted.
  - Cons: More complex when displaying the customer/product information in UI. Needs to query model whenver the system needs to calcualte statistics related with product and transactions.
- **Alternative 2 (previous choice):** Store the product/ customer instance as an attribute.
  - Pros: Easy to construct a transaction and display product/ customer name.
  - Cons: Easy to generate bugs while any of the instance is editted. Needs to update the product in transaction when a product is editted.

Aspect: How to change the quantity & sales attribute of product while editing transactions.

- **Alternative 1 (current choice):** If quantity/ product is changed, check validation first, re-store the quantity & sales of the original product, and then and update the quantity & sales of the new

product.

  - Pros: Straightforward logic, not likely to create bugs.
  - Cons: Validation checking would be very complex.

- **Alternative 2 (previous choice):** If quantity/ product is changed, re-store the quantity & sales of the original product, check validation (whether the product has that much inventory as required on transaction), and then and update the quantity & sales of the new product.
  - Pros: Easy to implement.
  - Cons: Likely to generate bugs when the new quantity exceeds inventory, i.e. the edit operation is not valid.

Plot the quantity sold of a product

The plot sales command is facilitated by `InventorySystemParser`. First, the InventorySystemParser class parses the user command. Then the `PlotSalesCommandParser` parses the user input arguments into the index of the product, the `start date`, and the `end date`. The generated `PlotSalesCommand` is executed by the LogicManager. The command execution generates a daily time sequence and calcualte the quantity sold on each day by querying all the related transactions. The time series data and the signal of displaying the sales graph is then encapsulated as a CommandResult object which is passed back to the Ui.

The following sequnce diagram shos how the plot sale operation works: