Eng Xuan En - Project Portfolio

PROJECT: Delino

Overview

Delino is a desktop delivery application for couriers to manage delivery tasks. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 10 kLoC

Summary of contributions

Given below are the summary of the contributions I have made for this project.

- **Major enhancement**:
  - Added `import` command (Pull request #215)
    - What it does: Allows the user to add multiple orders and return orders via a CSV file.
    - Justification: This is an essential update, which allows the user to save time from adding multiple orders and return order one by one.
    - Highlights: This enhancement depends heavily on `InsertCommandParser` and `ReturnCommandParser` to process the data retrieve from the data. Considerations have to be made to differentiate the data for `Order` and `ReturnOrder`, as they have different attributes in their models.
  - Add return order book for storage (Pull request #186)
    - What it does: Stores return orders.
    - Justification: The storage of return orders is important, as the return orders are required to be separate from orders and store as a separate json file in the hard disk from the orders.
    - Highlights: Considerations are to be made for how to store and retrieve the return order data from the json file which is stored in hard disk.
  - Modify the `clear` command (Pull requests #180, #218)
    - What it does: Gives the user three options to clear:
      - Only order list
      - Only return order list
      - Both order list and return order list
    - Justification: This is an essential update, which prevent the user from having the trouble clearing the orders or return orders one by one.
    - Highlights: Considerations are to be made for the clearing either one of the list or both

order list and return order list, as clearing the wrong list, the data in that list would be lost permanently.

- **Minor enhancement**:
  - Add `timestamp` field into the model(Pull requests #175, #198)
    - What it does: Allows orders and return orders to keep track of the delivery date and time.
    - Justification: This is an essential update, which allows user to input timestamp for the order and return order.
  - Add `comment` field into the model(Pull request #172)
    - What it does: Allows orders and return orders to store customer comment.
    - Justification: This is an essential update, which allows the courier to be able to take note about
      the customer's request, such as sending a message to notify the customer that the courier will be sending
      their parcel on that day.
- **Code contributed**: [Functional code & Test code]
- **Other contributions**:
  - Enhancements to existing GUI:
  - Modified the `Clear Window` to suit the needs for the modified `clear` command. (Pull request #180)
  - Replaced the address book icon to Delino icon. (Pull request 226)
  - Project management:
    - Co-team lead of the project, in charge of defining, assigning, and tracking project tasks. Furthermore, ensure project deliverables are done on time and in the right format.
  - Documentation:
    - Updated the product website's heading and navigation bar. (Pull request #119)
    - Edited various parts of the User Guide (Pull requests #173, #226)
    - Added use cases and updated the Storage and Implementation part of the Developer Guide for saving, import and clear features (Pull requests #186, #197, #218)
  - Community:
    - PRs reviewed (with non-trivial review comments): (Pull requests #126, #139, #174, #189)
    - Reported bugs and suggestions for other teams in the class: (Examples: PE dry run issue)
    - Contributed to forum discussion by asking help question: (Example: #104)
  - Tools:
    - Integrated a new Github plugin (AppVeyor) to the team repo (#37b0cd1)

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.

Features

Clearing all entries : `clear` [Done by - Eng Xuan En]

In this section, you will be able to find out how to use the `clear` command and the possible combinations of the `clear` command.

If you wish to delete huge amount of orders and return orders from Delino, the `clear` command will provide you the means to do so.

How to use the Clear command

Here is how you can clear the order list and return order list by following the steps below:

Step 1 : Type in the keyword `clear`.
Step 2 (Optional) : Provides the `-f` flag if you confirm that you would like to force clear both order list and return order list.
Step 3 (Optional) : Provides the `-o` or `-r` flag if you only want to clear either order list or return order list respectively.
Step 4 : Press `Enter` on your keyboard to send the instruction to Delino.

NOTE

- A space is required in between the command and flags. Refer to the examples below for more information.
- If **no** `-o` or `-r` flag is given, it will be treated as both. **Both** order list and return order list will be cleared.
- Flags can be in any order such that `clear -f -o` and `clear -o -f` indicate to force clear the order list.
- If an `-f` flag is given, no user confirmation will be requested.

What constitutes a valid Clear command

In this section, you will learn about the syntax for a valid `clear` command and the possible combinations of the command.

The syntax for a valid `clear` command can be seen below:

- `clear [FLAG]`

NOTE

- `clear` is the command word for the clear feature
- `FLAG` could be either `-f`, `-r` and `-o`; whereby `-f` flag indicate force clear and `-r` and `-o` indicates the return order list and order list respectively.

These are the possible combinations of the `clear` command:

Table 1. Possible Combinations of Clear command

Scenario	Command	Result
If you want to have a confirmation notice before clearing both lists.	`clear`	Pop-up will appear and ask the user for confirmation to clear both order list and return order list.
If you don't want to get prompt before clearing both lists.	`clear -f`	Both order list and return order list will be cleared immediately.
If you want to have a confirmation notice before clearing return order list.	`clear -r`	Pop-up will appear and ask the user for confirmation to clear the return order list.
If you don't want to get prompt before clearing return order list.	`clear -f -r`	Only return order list is cleared immediately.
If you want to have a confirmation notice before clearing order list.	`clear -o`	Pop-up will appear and ask the user for confirmation to clear the order list.
If you don't want to get prompt before clearing order list.	`clear -o -f`	Only order list is cleared immediately.

Importing a list of orders : `import` [Done by - Eng Xuan En]

In this section, you will be able to find out how to use the `import` command and the possible combination of the `import` command.

If you wish to add huge amount of orders and return orders to Delino, the `import` command will provide you the means to do so.

How to use Import command

Here is how you can import the orders and return orders into Delino by following the steps below:

Step 1 : Type in the keyword `import`.
Step 2 : Provide the CSV file name that you would like to import with the **.csv** extension behind.
Step 3 : Press `Enter` on your keyboard to send the instruction to Delino.

What constitutes a valid Import command

In this section, you will learn about the syntax of a valid `import` command, the format for both orders and return orders in the CSV file and the possible combination for the `import` command.

The syntax for a valid `import` command can be seen below:

- `import FILE_NAME`
  `import` is the command and `FILE_NAME` is the file name that is required to import with the **.csv** extension behind.

NOTE

- Only CSV file could be imported.
- Only **one** CSV file can be imported at one time.
- The `FILE_NAME` should include the extension. For example: `orders.csv`.
- The folder, **data**, which the CSV files are stored in, should be in the same directory as the JAR file.

WARNING

- CSV file should be saved only as CSV (Comma delimited), as shown in the figure below:

Figure 1. CSV File Save Type

The data in the `CSV file` should be written in the following format:

- Order data format:
  `ot/ORDER_TYPE`, `tid/TRANSACTION_ID`, `n/NAME`, `a/ADDRESS`, `p/PHONE_NUMBER`, `e/EMAIL`, `dts/DELIVERY_DATE_&_TIME`, `w/WAREHOUSE_LOCATION`, `cod/CASH_ON_DELIVERY`, `[c/COMMENTS_BY_CUSTOMER]`, `[type/TYPE_OF_ITEM]`

  Example:

	A	B	C	D	E	F	G	H	I	J	K
1	ot/order	tid/AX23456789	n/Amos Cheong	a/Blk 572 Hougang st 51 #11-37 S530572	p/90010019	e/amoscheong@gmail.com	dts/2020-05-10 1650	w/Marsiling	cod/$2.50	c/Leave it at the riser	type/glass

Figure 2. Order data format in CSV file

- Return order data format:
  `ot/ORDER_TYPE`,`tid/TRANSACTION_ID`, `n/NAME`, `a/ADDRESS`, `p/PHONE_NUMBER`, `e/EMAIL`, `rts/RETURN_DATE_&_TIME`, `w/WAREHOUSE_LOCATION`, `[c/COMMENTS_BY_CUSTOMER]`, `[type/TYPE_OF_ITEM]`

  Example:

	A	B	C	D	E	F	G	H	I	J
1	ot/return	tid/AX74546789	n/Sebastian Liew	a/Blk 664 Hougang st 51 #09-21 S530664	p/97764520	e/Sebastian@gmail.com	rts/2020-05-05 1200	w/Marsiling	c/Leave it at the riser	type/glass

Figure 3. Return order data format in CSV file

NOTE

- Commas `,` are required in between the different fields.
- Prefixes are required before any value for that field.
- `ORDER_TYPE` can only be either `order` or `return`.
- Only `COMMENTS_BY_CUSTOMER` and `TYPE_OF_ITEM` are optional.
- `ORDER_TYPE` denote the start of an delivery order or return order.
- If there are 2 or more `ORDER_TYPE` within a single CSV row, it will be treated as 2 or more orders.
- The last value of the same type will be stored if duplicate prefix type is found in a single order sentence.
- Duplicate and invalid order or return order will not be imported into Delino.
- You could download a sample CSV file in our release under assets to start with.

There is only one possible combination for the `import` Command:

Table 2. Possible combination of Import command

Scenario	Command	Result
If you want to import all of the orders and return orders at once via a CSV file.	`import orders.csv`	Import the contents of the CSV file, `orders.csv`, to Delino.

Command Summary

In this section, you can find out more about the commands supported by Delino (their respective format and example).

If you would like to know more about a specific command, you can view more information by clicking the provided link in the table below.

Table 3. Command Summary

Command	Format	Example
Insert	`insert tid/TRANSACTION_ID n/CUSTOMER_NAME a/ADDRESS p/PHONE_NUMBER e/EMAIL dts/DELIVERY_DATE_&_TIME w/WAREHOUSE_LOCATION cod/CASH_ON_DELIVERY [c/COMMENTS_BY_CUSTOMER] [type/TYPE_OF_ITEM]`	`insert tid/0123456789 n/Eng Xuan En a/Tampines St 84 Blk 877 S520877 #01-123 p/87654321 e/xuanen@example.com dts/2020-02-20 1300 w/Yishun industry cod/$4.50 c/please knock the door three times :D type/heavy`

Command	Format	Example

Clear	`clear [FLAG]`	`clear`
		`clear -f`
		`clear -r`
		`clear -f -r`
		`clear -o`
		`clear -o -f`

Delete	`delete FLAG INDEX`	`delete -o 2`
Delivered	`delivered FLAG INDEX`	`delivered -r 2` `delivered -o 1`

Edit	`edit FLAG INDEX ORDER_ATTRIBUTE_PREFIX/VALUE`	`edit -r 2 n/Xuan En`
		`edit -o 2 p/9999 4444`
		`edit -o 1 a/Blk 123 Pasir Ris Street 51 #12-21 S510123`
		`edit -r 3 n/Mr Tan p/0123 4567 a/Blk 141 Yishun st 71 #09-09 S760141`

Exit	`exit`	`exit`

Search	`search [FLAG] ORDER_ATTRIBUTE_PREFIX/KEYWORD [MORE_KEYWORDS]… [ORDER_ATTRIBUTE_PREFIX/KEYWORD MORE_KEYWORDS]…`	`search -r tid/ac1e345x7s`
		`search -r Jeremy Loh`
		`search -o tid/asj2od3943`
		`search -r p/92039999`
		`search -o p/92039999`
		`tid/asj2od3943 n/jeremy`

Help	`help`	`help`
Show	`show`	`show`
Import	`import FILE_NAME`	`import orders.csv`

List	`list [DONE_STATUS]`	`list`
		`list done`
		`list undone`

Return	`return tid/TRANSACTION_ID n/CUSTOMER_NAME a/ADDRESS p/PHONE_NUMBER e/EMAIL rts/RETURN_DATE_&_TIME w/WAREHOUSE_LOCATION c/COMMENTS_BY_CUSTOMER type/TYPE_OF_ITEM`	`return tid/ac17s2a n/BOBBY TAN a/123 Delta Road #03-333, Singapore 123456 p/91230456 rts/12-12-2020 1301 w/Jurong Warehouse c/NIL type/glass`
		`return tid/ac17s2a`
		`return tid/b1230512 n/Aaron Teo a/256 Alpha Road #03-222, Singapore 123567 p/91230456 e/aaron@example.com rts/12-12-2020 1400 w/Jurong Warehouse c/Leave it at the lobby type/metal`

Command	Format	Example

Nearby	`nearby [FLAG] POSTAL_SECTOR`
	OR
	`nearby [FLAG] AREA`
		`nearby east`
		`nearby -o 14`
		`nearby -r north`

Glossary

Command Prefix

Table 4. Command Prefix

Prefix	Order Attributes	Used in the following Command(s)
ot/	Order Type	Import
tid/	Transaction ID	Edit, Insert, Return, Search, Import
n/	Customer Name	Edit, Insert, Return, Search, Import
a/	Address	Edit, Insert, Return, Search, Import
p/	Phone Number	Edit, Insert, Return, Search, Import
e/	Email	Insert, Edit, Return, Search, Import
dts/	Delivery Date And Time	Edit, Insert, Return, Search, Import
rts/	Return Date and Time	Return, Search, Import
w/	Warehouse Location	Edit, Insert, Return, Search, Import
cod/	Cash On Delivery	Edit, Insert, Search, Import
c/	Comments by Customer	Edit, Insert, Return, Search, Import
type/	Type of Item	Edit, Insert, Return, Search, Import

Command Flags

Table 5. Possible Command Flags

Flag	Meaning	Used in the following Command(s)
-f	Force clear, no user confirmation will be requested	Clear
-o	Order flag, Operation on order list	Clear, Delete, Delivered, Edit, Nearby, Search
-r	Return Order flag, Operation on return order list	Clear, Delete, Delivered, Edit, Nearby, Search

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.

Design

Storage component

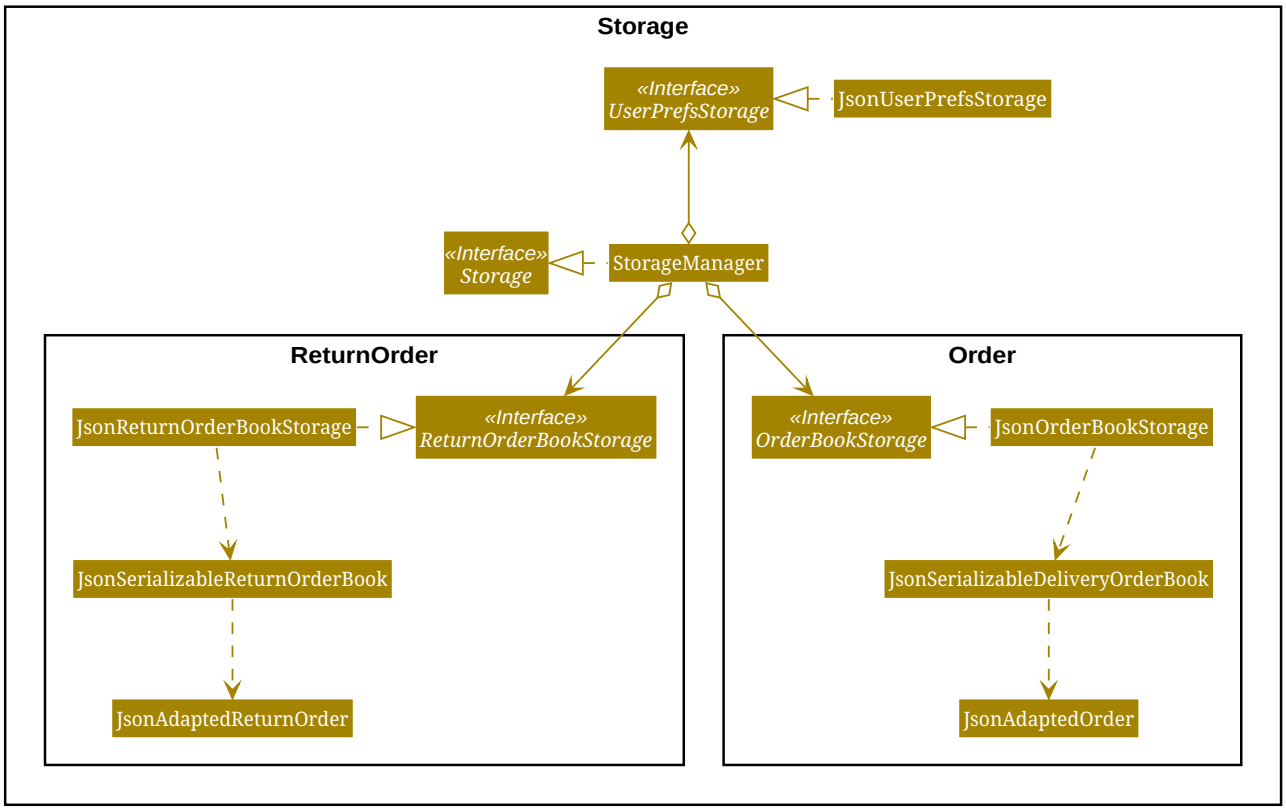

Figure 4. Structure of the Storage Component

API : `Storage.java`

The `Storage` component,

- can save `UserPref` objects in json format and read it back.
- can save both OrderBook and ReturnOrderBook data in json format and read it back.

Features

Clear feature

In this section, the functionality of the `clear` feature, the expected execution path, the structure of the `ClearCommand` class and the interactions between objects with the `ClearCommand` object will be discussed.

What is the Clear feature

The `clear` feature was implemented as a `ClearCommand` in the `logic` package.
The `clear` feature allows the user to remove the orders and return orders by input one command line.

Execution paths of Clear Command

The execution path of the `ClearCommand` is shown below:

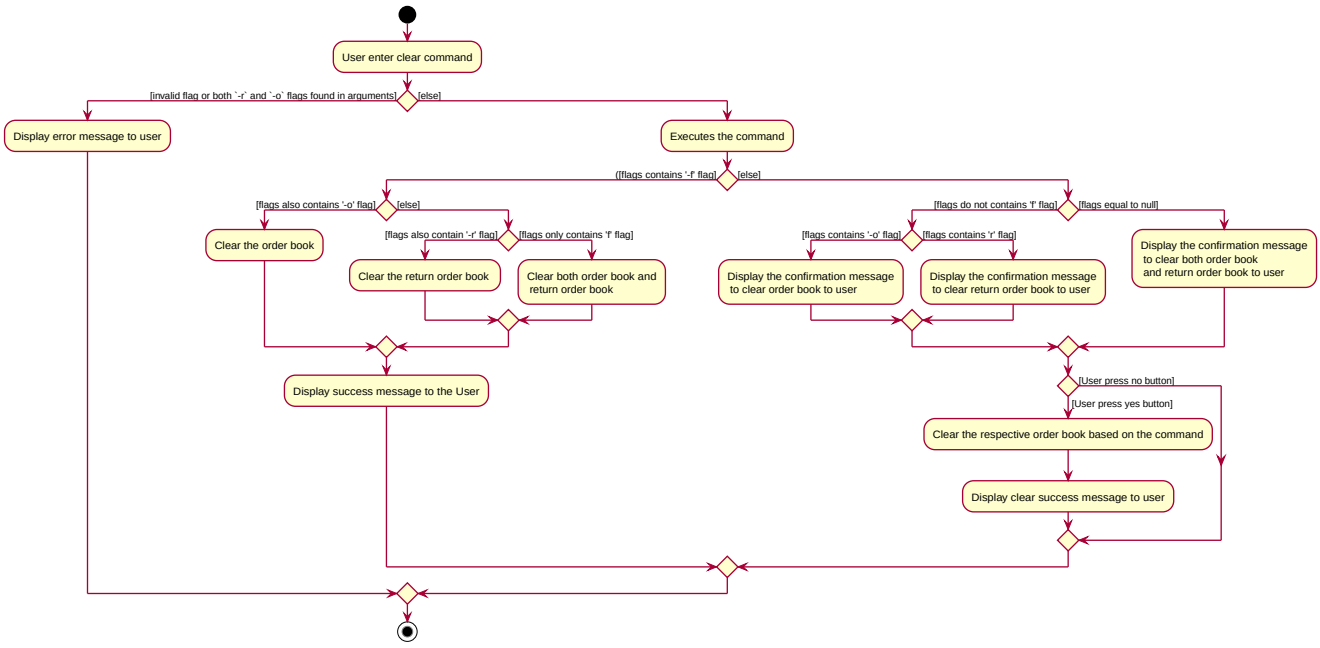

Figure 5. Clear Class Activity Diagram

After user enter the `clear` command, the `ClearCommandParser` will run the following two checks:

- Check if flag in the arguments do not belong to one of the three valid flags: `-f`, `-r` and `-o`
- Check if both `-r` and `-o` flags found in the arguments

If either one of the conditions occurs, exception will be thrown and the error message will be display to the user. Afterward, the new `ClearCommand` object will be executed.

During the execution of the `ClearCommand`:

- If `-f` flag is found in flags, the respective order book will be cleared and display a success message to the user.
- If there are no `-f` flag found in flags, a pop up will appeared with the confirmation message. User would be required to press either one of the following two buttons:
  - `Yes` button - The respective order book will be cleared and display successful clear message to the user.
  - `No` button - Pop up closed and end of activity.

Structure of Clear Command

The following diagrams shows the overview of the `ClearCommand` Class Diagram:

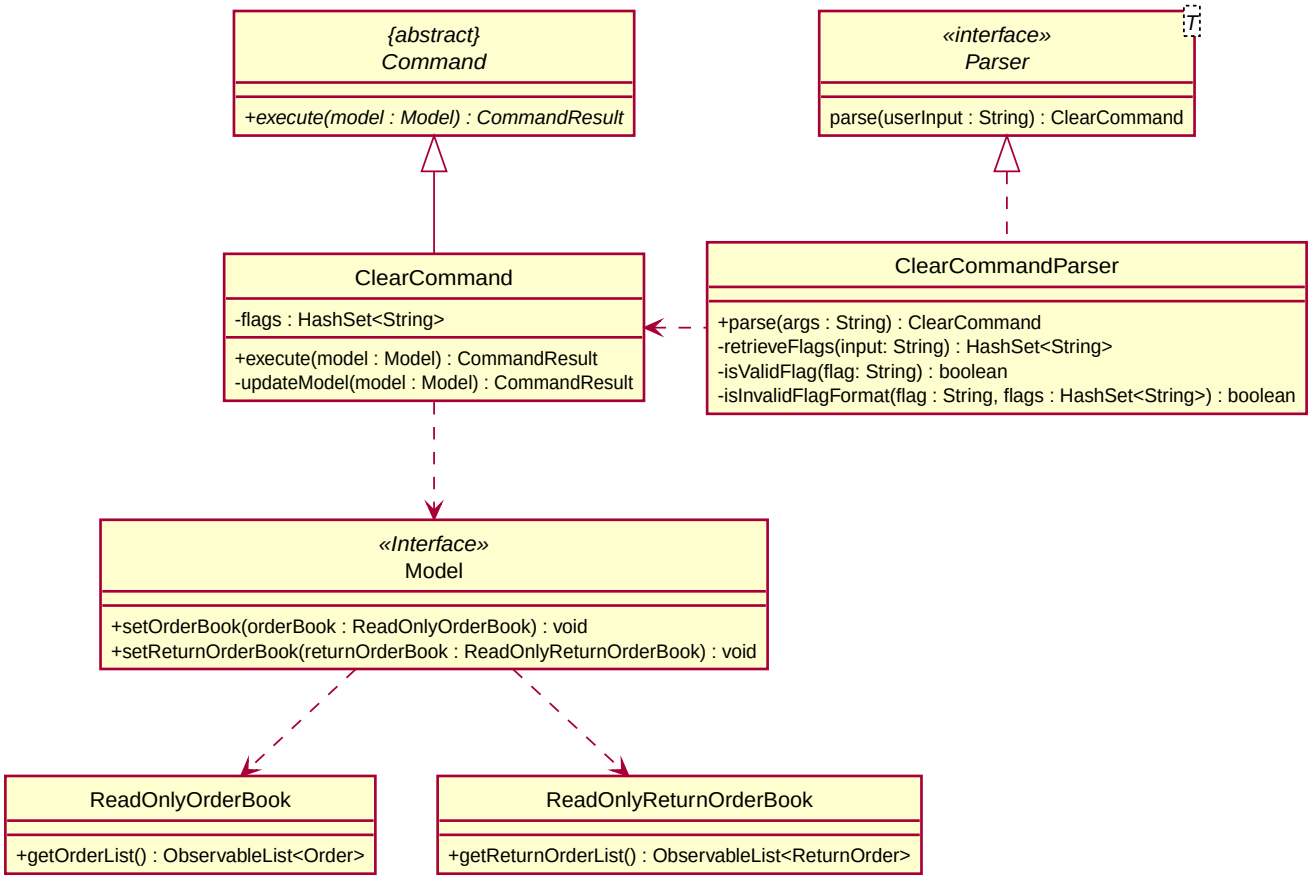

Figure 6. Clear Command Class Diagram

In the `ClearCommand` class, there are also some static messages for the different input command the user has key in:

1. `MESSAGE_USAGE`
   clear: Clear either both order book list and return order book list or one of them.
   Parameters: -o/-r/-f
   Example: clear -o -f
2. `MESSAGE_SUCCESS_ORDER_BOOK`
   Inform the user that order book list has been cleared successfully.

3. `MESSAGE_SUCCESS_RETURN_BOOK`
   Inform the user that return order book list has been cleared successfully.
4. `MESSAGE_SUCCESS_BOTH_BOOK`
   Inform the user that both order book lists have been cleared successfully.
5. `MESSAGE_ENQUIRY_ORDER_BOOK`
   Confirmation message to the user if the user want to clear order book list.
6. `MESSAGE_ENQUIRY_RETURN_BOOK`
   Confirmation message to the user if the user want to clear return order book list.
7. `MESSAGE_ENQUIRY_BOTH_BOOK`
   Confirmation message to the user if the user want to clear both order book lists.

Interactions between objects when Clear Command is executed

In this section, the interactions between objects when `ClearCommand` is executed will be display in the Clear Command Sequence Diagram below:

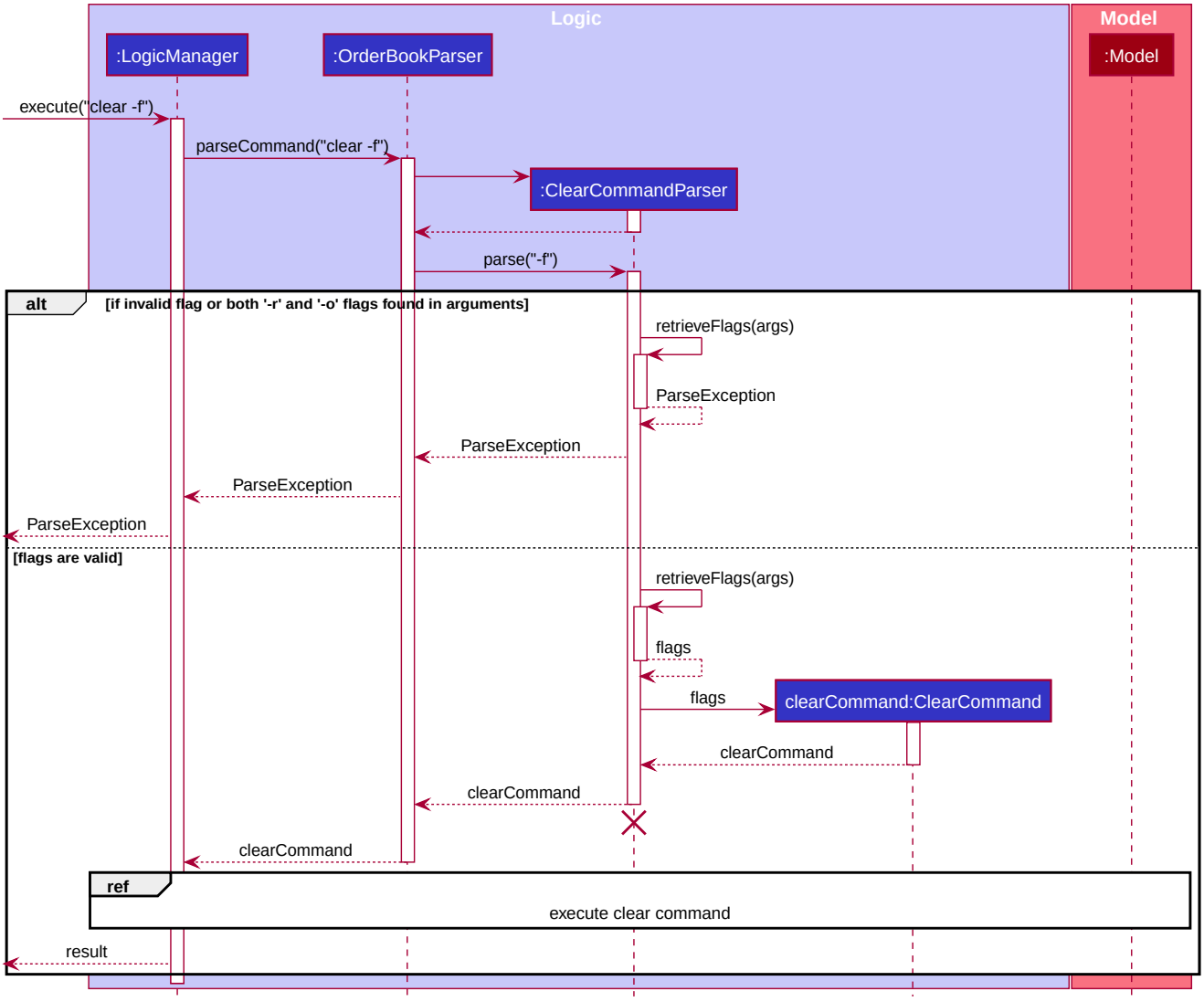

Figure 7. Clear Command Sequence Diagram

The above sequence diagram illustrate how `Clear` Command being processed when the user input `clear -f` to force clear both order book and return order book.

After the user input. the arguments passed to the `Clear` Command will be parsed by the `ClearCommandParser` class
If the given arguments are valid, a new `Clear` Command object will be returned.

In the `ClearCommandParser`, there will be two validation checks:
1. Ensure the flag is one of the three flags: `-f`, `-o` and `-r`
2. Ensure the arguments do not have both `-o` and `-r` flags.

After the two validation checks, the flag will be added into HashSet, **flags** which will then passed to the new `ClearCommand` object created by `ClearCommandParser` and it is being returned to the `LogicManager`. The `LogicManager` will start to run the execute the `clear` Command, which will be shown in details in below diagram:

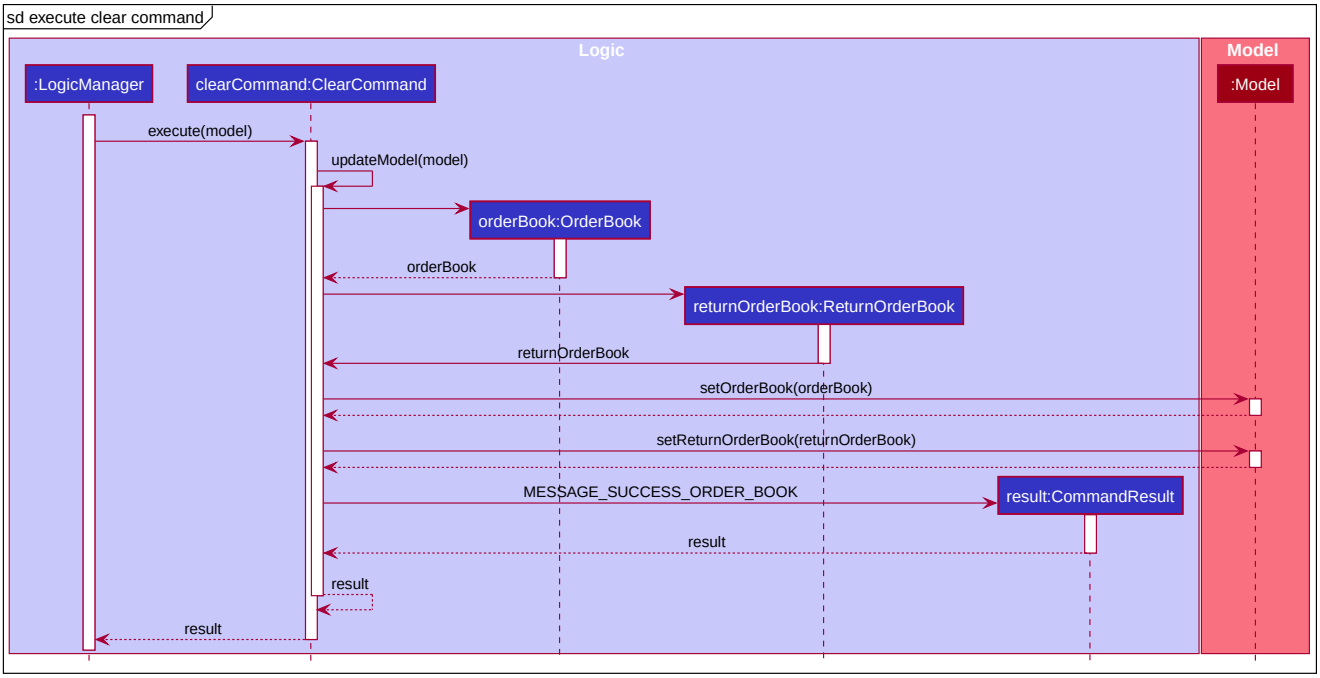

Figure 8. Execution of Clear Command Sequence Diagram

After `LogicManager` call the `ClearCommand#execute(model)`, the `clear` Command will update the model by pass a new `OrderBook` object and a new `ReturnOrderBook` object to `Model`. The `Model` will then update its own orderBook and returnOrderBook. In addition, the `clear` Command will pass back a new `CommandResult` object with the success message in it to the `LogicManager` at the end of the execution.

Import feature

In this section, the functionality of the `import` feature, the expected execution path, the structure of the `ImportCommand` class and the interactions between objects with the `ImportCommand` object will be discussed.

What is the Import feature

The `import` feature was implemented as the `ImportCommand` in the `logic` package.
The `import` feature allows users to save the trouble of adding the delivery orders and the return orders one by one when they have large amount of delivery orders or return orders to add into

Delino.

Execution paths of Import Command

The execution path of the `ImportCommand` is shown below:

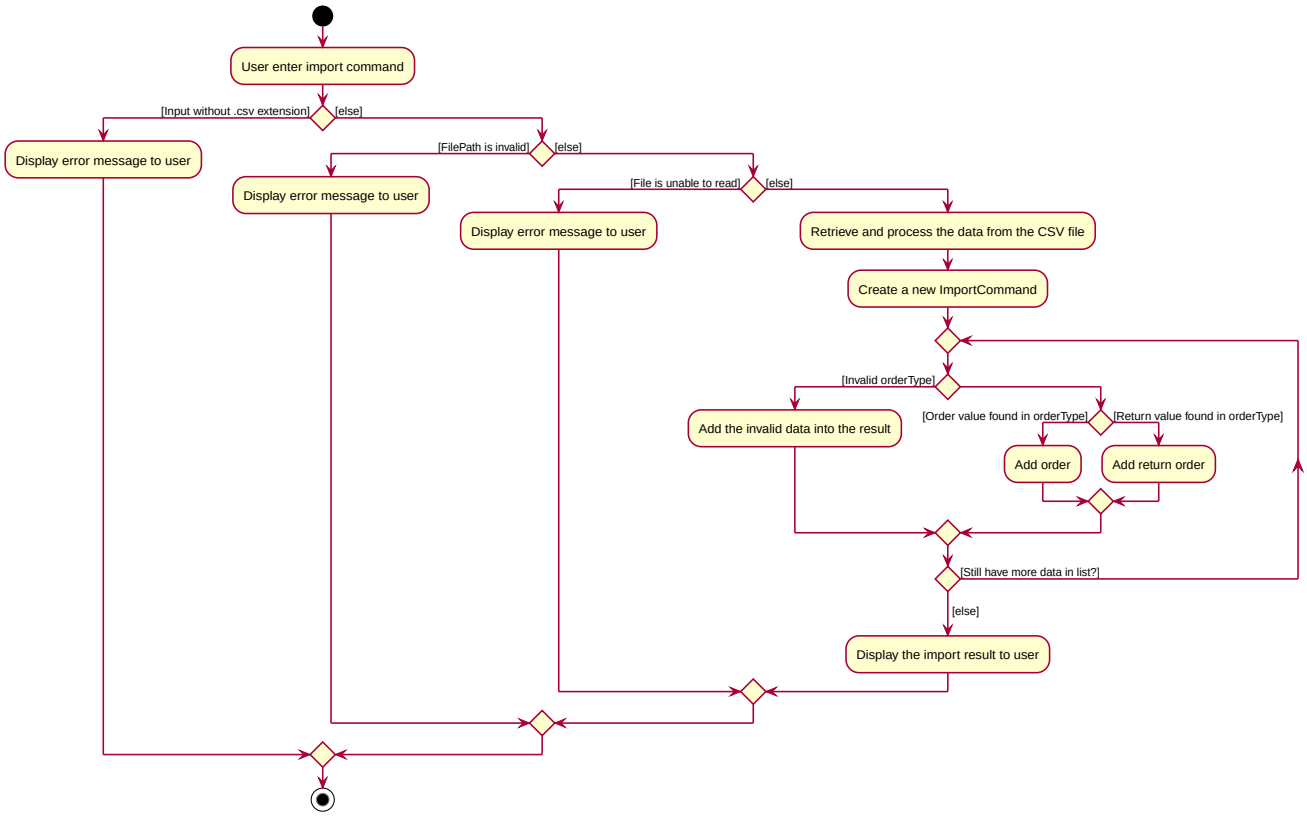

Figure 9. Import Command Activity Diagram

After the user enter the `import` command, there are three validation check for the file based on the input argument, `FILE_NAME`:

- Check if the input argument has the .csv file extension at the back:
  - If `Yes`, continue with the next validation check.
  - If `No`, display error message to the user.
- Check if the filePath is valid:
  - If `Yes`, continue with the next validation check.
  - If `No`, display error message to the user.
- Check if the file able to read:
  - If `Yes`, retrieve the data from the CSV file and process the data.
  - If `No`, display the error message to the user.

Afterward, a new `ImportCommand` will be created and executed. For every data inside the list, either order or return order will be added into the order book and return order book respectively based on the `orderType` value. If the `orderType` is invalid, add the data into the result, which will be displayed to the user after processing.

Structure of Import Command

The following diagram shows the overview structure of the `ImportCommand` Class Diagram:

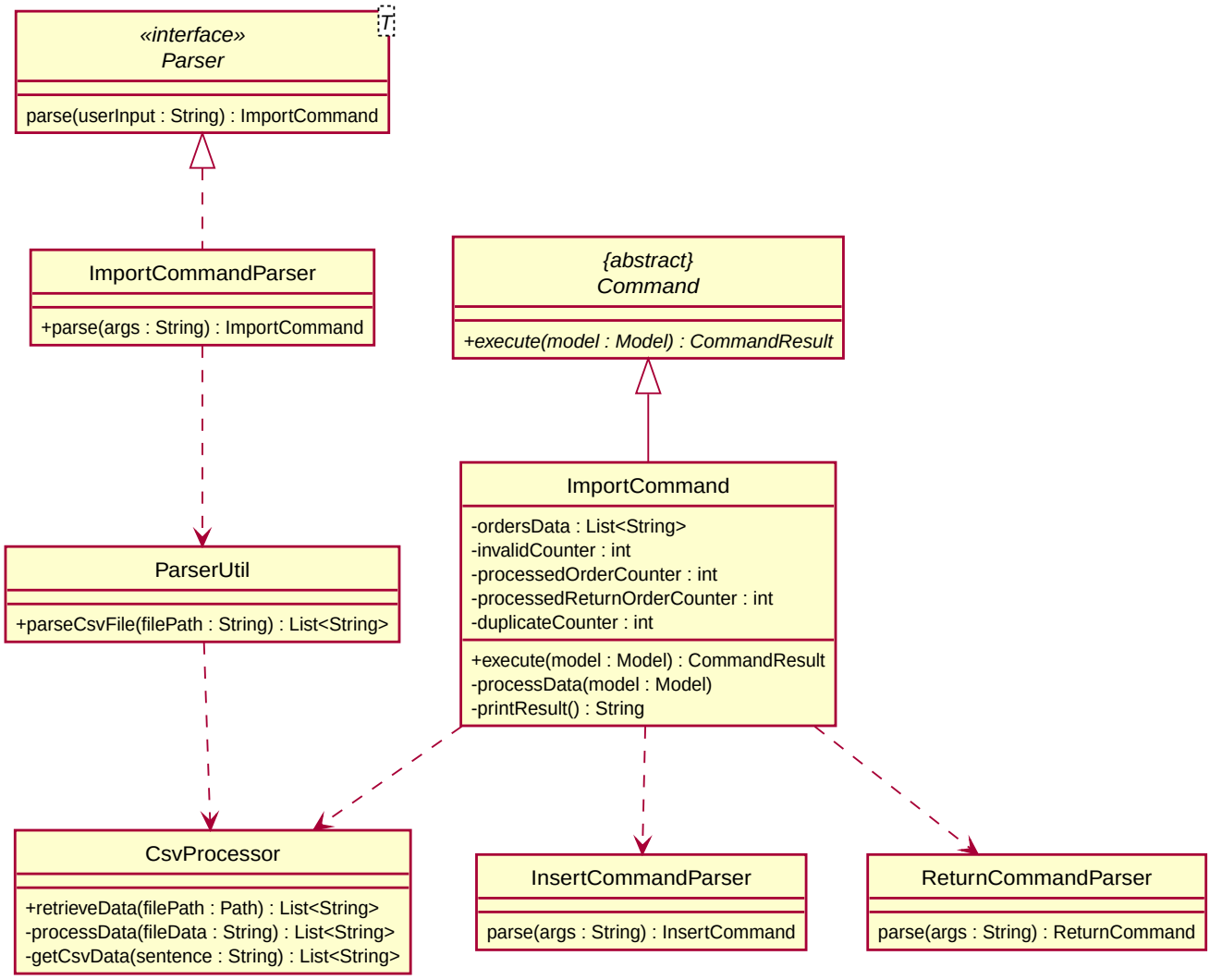

Figure 10. Import Command Class Diagram

In the `ImportCommand` Class, there are also a few static message to display to the user for the various scenarios occurred during the importing of data from the CSV file:

1. `MESSAGE_USAGE`
   import: Import the data in .csv file into Delino
   Parameters: fileName.csv\n
   Example: import orders.csv
2. `INVALID_MESSAGE`
   Invalid order type encountered.
3. `DUPLICATE_ORDER_MESSAGE`
   Duplicate order encountered.
4. `DUPLICATE_RETURN_MESSAGE`
   Duplicate return order encountered.
5. `MESSAGE_INVALID_CSV_FILEPATH`
   The csv file is not found in the data folder.

6. `PROCESS_FAILED_MESSAGE`
   Failed to process the data.
   This could be due to invalid order type encountered or invalid data input for the attributes in order and return order.

Interactions between objects when Import Command is executed

In this section, the interactions between the objects when `ImportCommand` is executed will be shown in the Import Command Sequence Diagram below:

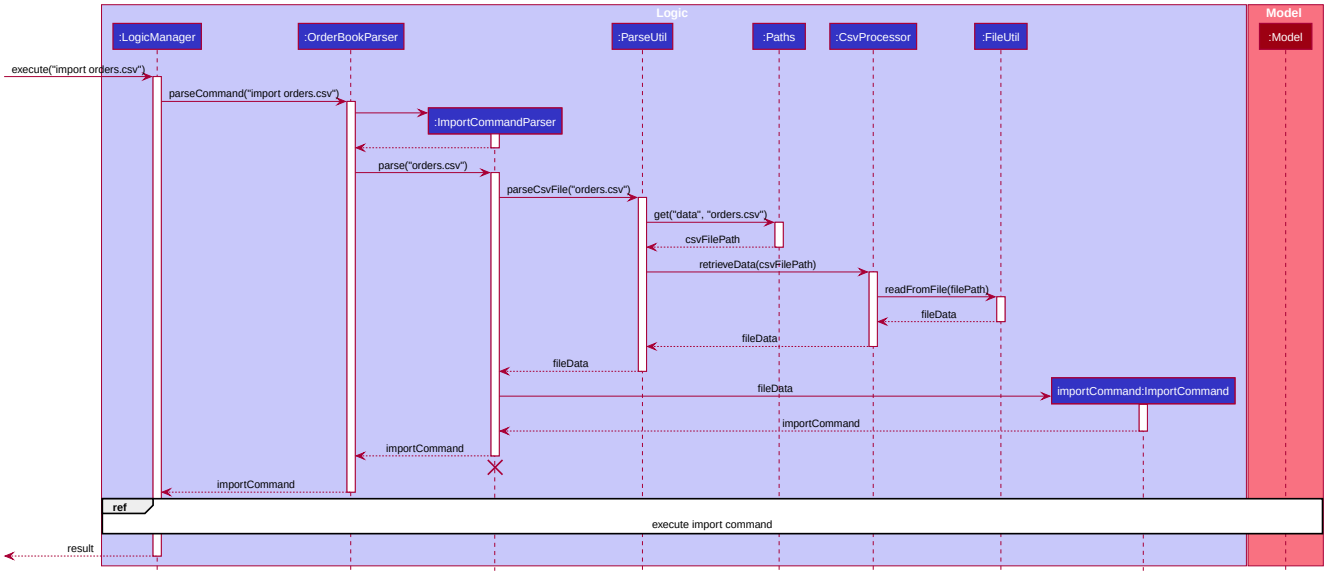

Figure 11. Import Command Sequence Diagram

The arguments passed to the `import` Command will be parsed by the `ImportCommandParser` class.
Then, the `ImportCommandParser` will called the `ParseUtil#parseCsvFile()` to get the filePath based on the input the user provides. Afterward, `CsvProcessor` will be called to retrieve the data from the csv file and return the processed `fileData` back to ParseUtil. The `fileData` will be further pass to `ImportCommandParser` and to the constructor of `ImportCommand`.

NOTE	`CsvProcessor` is a helper class that helps to retrieve the data from the csv file and process the data before giving to `ImportCommand`.

Afterward, the `ImportCommand` object is being returned to the `LogicManager` and the `LogicManager` will start to run the execute the `ImportCommand`, which will be shown at the diagram below.

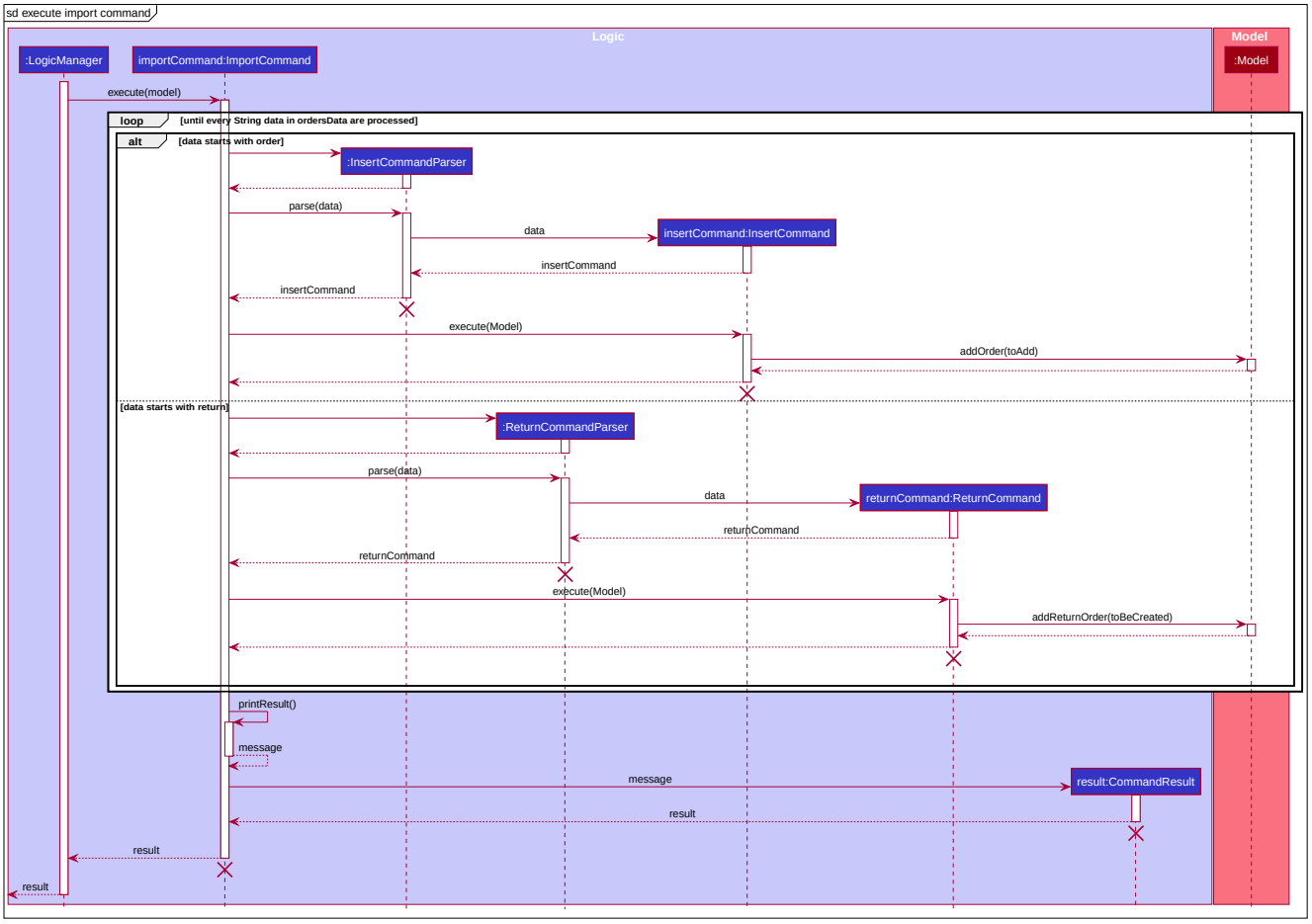

Figure 12. Execution of Import Command Sequence Diagram

The `ImportCommand#execute()` will first check if the data given starts with **order** or **return** and pass to the `InsertCommandParser` or `ReturnCommandParser` respectively.

Afterwards, `InsertCommandParser` or `ReturnCommandParser` will return a new `InsertCommand` or `ReturnCommand` respectively if it successfully parse the data. The `ImportCommand` will then call the `InsertCommand#execute()` or `ReturnCommand#execute()` depend whether it is delivery order or return order. This will cause a delivery order or return order being added into the `Model`.

The `ImportCommand` will call its own `printResult()` function and return a String message to the `CommandResult` object which is then pass back to the `LogicManager`.

Use Cases

Use case: UC02 - Clear all orders

MSS

1. User wants to clear all orders.
2. User requests to clear all orders.
3. Delino clear all existing orders.
4. Delino displays order cleared message.

Use case ends.

Extensions

2a. Delino detects invalid syntax from user input.

2a1. Delino shows an error message.

Use case ends.

2b. Delino detects no flag `-f`.

2b1. Delino trigger pop-up message.

- 2b2a. User select `yes` button.
  - 2b2a1. Return to step 2.
- 2b2b. User select `no` button.
  - 2b2b1. Use case ends.

2c. Delino detects no orders.

2c1. Delino shows no order to be cleared message.

Use case ends.

Use case: UC09 - Importing order details

MSS

1. User wants to import a specific orders from an external file.
2. User requests to import orders from an external file.
3. Delino checks for file existence.
4. Delino imports all orders from the external file.
5. Delino displays all orders imported.

Use case ends.

Extensions

2a. Delino detects invalid syntax from user input.

2a1. Delino shows an error message.

Use case ends.

3a. Delino detects invalid file path.

3a1. Delino shows the invalid file path error message

Use case ends.

4a. Delino is unable to open the file.

4a1. Delino shows permission denied error message.

Use case ends.

Glossary

Table 6. Command Prefix

Prefix	Meaning	Used in the following Command(s)
ot/	Order Type	Import
tid/	Transaction ID	Edit, Import, Insert, Return, Search
n/	Customer Name	Edit, Import, Insert, Return, Search
a/	Address	Edit, Import, Insert, Return, Search
p/	Phone Number	Edit, Import, Insert, Return, Search
e/	Email	Import, Insert, Edit, Return, Search
dts/	Delivery Date And Time	Edit, Import, Insert, Return, Search
rts/	Return Date and Time	Import, Return, Search
w/	Warehouse Location	Edit, Import, Insert, Return, Search
cod/	Cash On Delivery	Edit, Insert, Search
c/	Comments by Customer	Edit, Import, Insert, Return, Search
type/	Type of Item	Edit, Import, Insert, Return, Search

Table 7. Possible Command Flags

Flag	Meaning	Used in the following Command(s)
-f	Force clear, no user confirmation will be requested	Clear
-o	Order flag, Operation on order list	Clear, Delete, Delivered, Edit, Nearby, Search

Flag	Meaning	Used in the following Command(s)
-r	Return Order flag, Operation on return order list	Clear, Delete, Delivered, Edit, Nearby, Search

Appendix G: Instructions for Manual Testing

Clear orders

1. Clear all orders while all orders are listed
    a. Clear command format: `clear [FLAG]`
    b. Test case: `clear`
    Expected: Confirmation message will display in status message.
        i. If **Yes** button is pressed, the both order and return order lists will be cleared. Notify the user that both order lists have been cleared in the status message.
        ii. If **No** button is pressed, no order list is cleared.
    c. Test case: `clear -f`
    Expected: Both order list and return order list will be cleared. Notify the user that both order lists have been cleared in the status message.
    d. Test case: `clear -f -r`
    Expected: Only return order list will be cleared. Notify the user that return order list has been cleared in the status message.
    e. Test case: `clear -r -r -f`
    Expected: Only return order list will be cleared. Notify the user that return order list has been cleared in the status message.
    f. Test case: `clear -r -o`
    Expected: Invalid command input, as both `-r` and `-o` cannot be in a single command.
    Error details shown in the response message. A help message displayed for the user to type the correct command. Status bar remains unchanged
    g. Test case: `clear -r-f`
    Expected: Invalid command input, as space is required in between flags.
    Error details shown in the response message. A help message displayed for the user to type the correct command. Status bar remains unchanged === Import orders
2. Import a new list of orders from a .csv file given by the company
    a. Import command format: `import NAME_OF_FILE.csv`
    b. Prerequisites : The import file must be a `.csv file` and the `csv file` should be inside `data` folder which is the same directory as the JAR file. Otherwise, it will cause the app to raise an exception and print the error message. Should not import a file that is non-existent
    c. Test case: `import customers_20_02_2020.csv`
    Expected: In the response box, a message will appear to indicate that the import is successful. At the same time, the contents of the .csv file will be shown to the user in the

form of a list of orders