Oon Zhi Xiang - Project Portfolio

Hi! I am Zhi Xiang, a Year 2 Computer Science student at National University of Singapore (NUS). Below is a summary of my contributions for this project.

PROJECT: Seller Manager Lite

Overview

For our module CS2103T: Software Engineering, my team and I were tasked to enhance a desktop address book application called AddressBook.

We eventually turned it into a product called Seller Manager Lite (SML), which aims to help small-scale handphone resellers manage their sales and inventory with ease. Users interact with it using a Command Line Interface (CLI) and it has a Graphical User Interface (GUI) created with JavaFX. It is written in Java, and has more than 15,000 lines of code.

My role was being in charge of the Logic component, implementing the commands and parsers for Customer, Phone and Order data. Additionally, I was tasked to implement an undo/redo, find (search) and copy commands.

Summary of contributions

- **Major enhancement**: I 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.
  - 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.
  - Credits: yamgent - I adapted some of his code to implement the UndoRedoStack and fixed the bugs that came with his approach.
- **Minor enhancement**: I added the ability to create/read/update/delete customer, phone and order.
- **Minor enhancement**: I added find command(s) that allows the user to search for customer, phone or order that matches certain keywords. This required enhancing the existing find command.
- **Minor enhancement**: I added copy command(s) that allows the user to copy the data that they want onto their clipboard. This would make it convenient for them to copy data out of the application.
- **Code contributed**:

- contributed a total of 11k+ LOC:
    - ~5k functional: functional code
    - 6k+ test: test code
- Functional
    - Undo/Redo: #161
    - Find: #124, #165
    - CRUD: #80, #102, #119, #124, #183
- Test
    - Command Tests: #168
    - Model Manager: #106
- **Other contributions**:
    - Project management:
        - Managed releases `v1.3.1` and `v1.3.2` (2 releases) on GitHub
    - Enhancements to existing features:
        - Created an Archived Order Tab Panel (backend) as we realise that there was too much clutter in the order tab. #148
        - Added a history command to allow forgetful users to remember what they typed in. #154
        - Helped to implement Model Manager #75
    - Documentation:
        - Restructured User Guide to make it more reader-friendly: #168, #258
    - Community:
        - PRs reviewed (with non-trivial review comments): #73, #76, #137, #145
        - Reported bugs and suggestions for other teams in the class: ExerHealth #217, ExerHealth #224
    - Tools:
        - Set-up Netlify
        - Set-up Codacy #123
        - Added reposense config.json file #120

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.

Customer Commands

Commands that work on customers in SML.

Switch to Customer Tab Panel : `switch-c`

Switches to Customer Tab Panel.

Format: `switch-c`

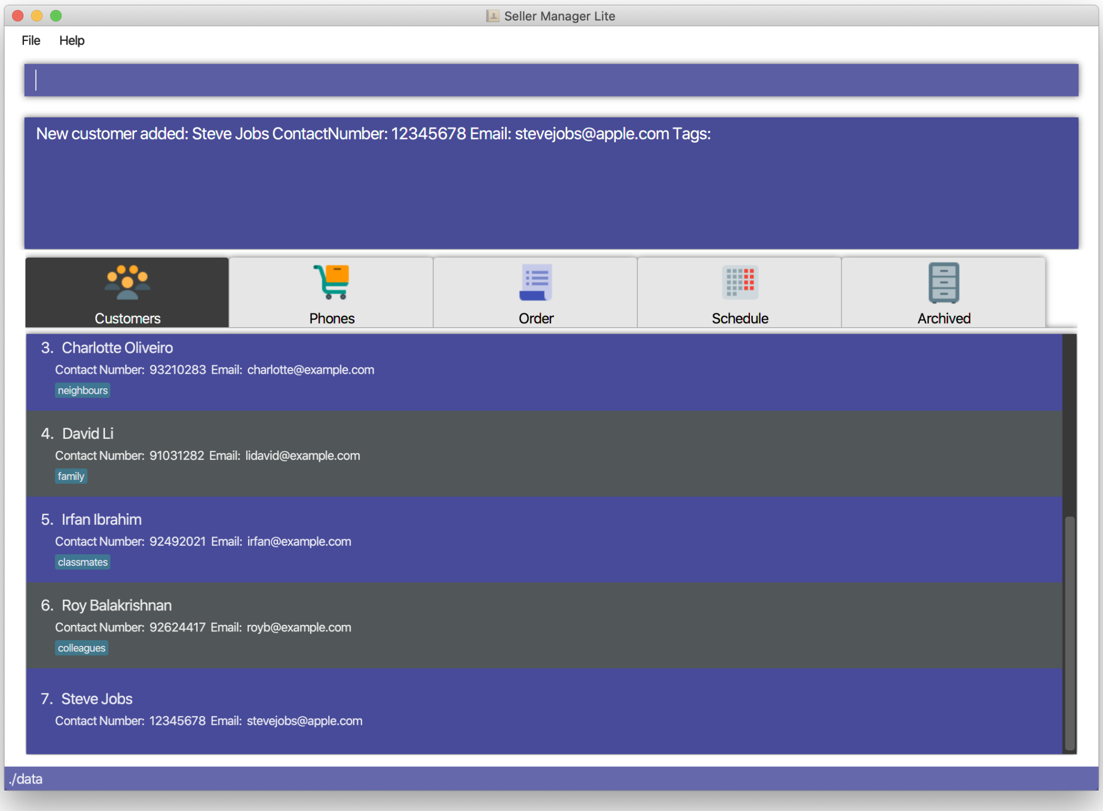
Add a customer : `add-c`

Adds a Customer to the CustomerBook.
This can be done in any Tab Panel.

Format: `add-c n/NAME c/CONTACT_NUMBER e/EMAIL [t/TAG]…`

TIP	Contact numbers should be 8-digits long.
TIP	A customer can have any number of tags, including 0.
TIP	Customers can share the same name.
TIP	Customers cannot share the same contact number or email.

Examples: * Adds a single customer . `add-c n/Steve Jobs c/12345678 e/stevejobs@apple.com`



New customer added: Steve Jobs ContactNumber: 12345678 Email: stevejobs@apple.com Tags:

3. Charlotte Oliveiro
Contact Number: 93210283 Email: charlotte@example.com
neighbours

4. David Li
Contact Number: 91031282 Email: lidavid@example.com
family

5. Irfan Ibrahim
Contact Number: 92492021 Email: irfan@example.com
classmates

6. Roy Balakrishnan
Contact Number: 92624417 Email: royb@example.com
colleagues

7. Steve Jobs
Contact Number: 12345678 Email: stevejobs@apple.com

Delete a customer : `delete-c`

Deletes a customer in SML. Note that deleting a customer will also delete the orders and schedules associated with the customer.

Format: `delete-c INDEX`

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

Examples:

- Delete the 2nd customer.
  1. `list-c`
  2. `delete-c 2`
- Delete the 1st customer after performing a find customer command.
  1. `find-c alice`
  2. `delete-c 1`

Find a customer : `find-c`

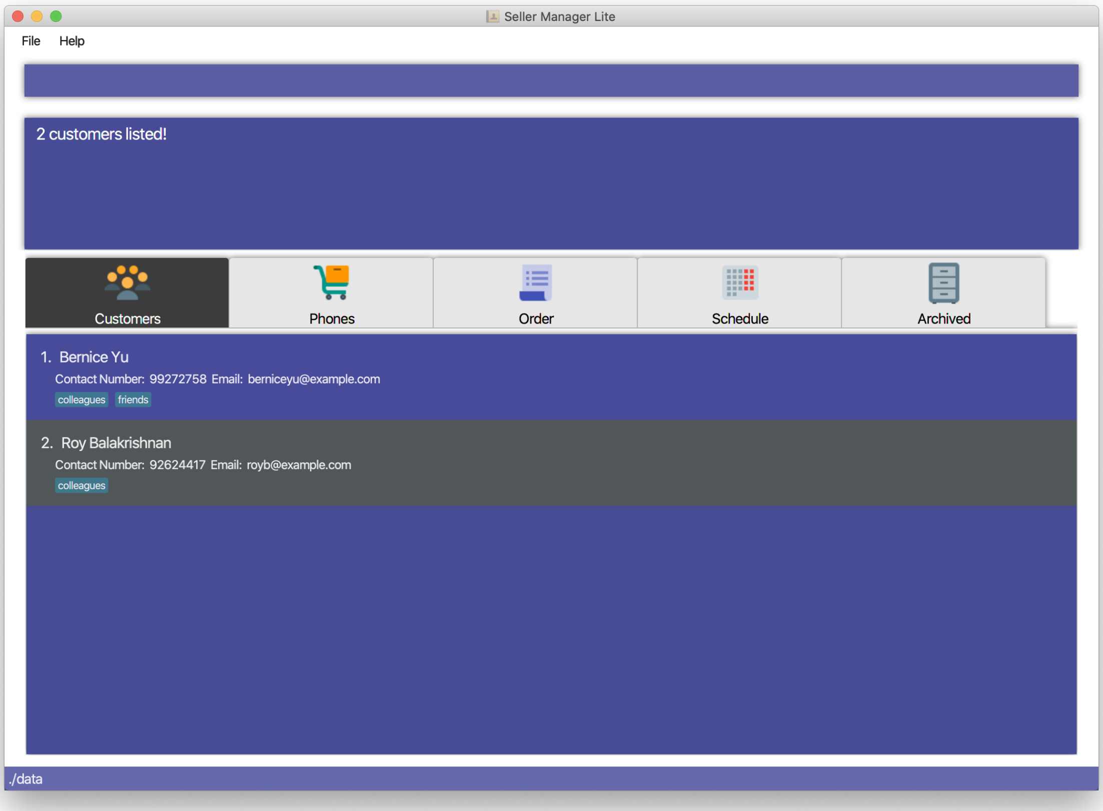
Finds customers whose fields contain any of the given keywords.

Format: `find KEYWORD [MORE_KEYWORDS]…`

- The search is case insensitive. e.g `hans` will match `Hans`
- The search matches anywhere for name, contact number, email and tags.
- The search looks for partial matches e.g. `ha` will match `hans`. However, `hns` will not match `hans`.
- Entries matching ALL fields will be returned (i.e. `AND` search).
  - e.g. `find-c aaa bbb` will match a customer with name `Aaah` and with tag `bbb`;

Examples:

- Find customers with keyword `colleague`
  1. `find-c colleague`

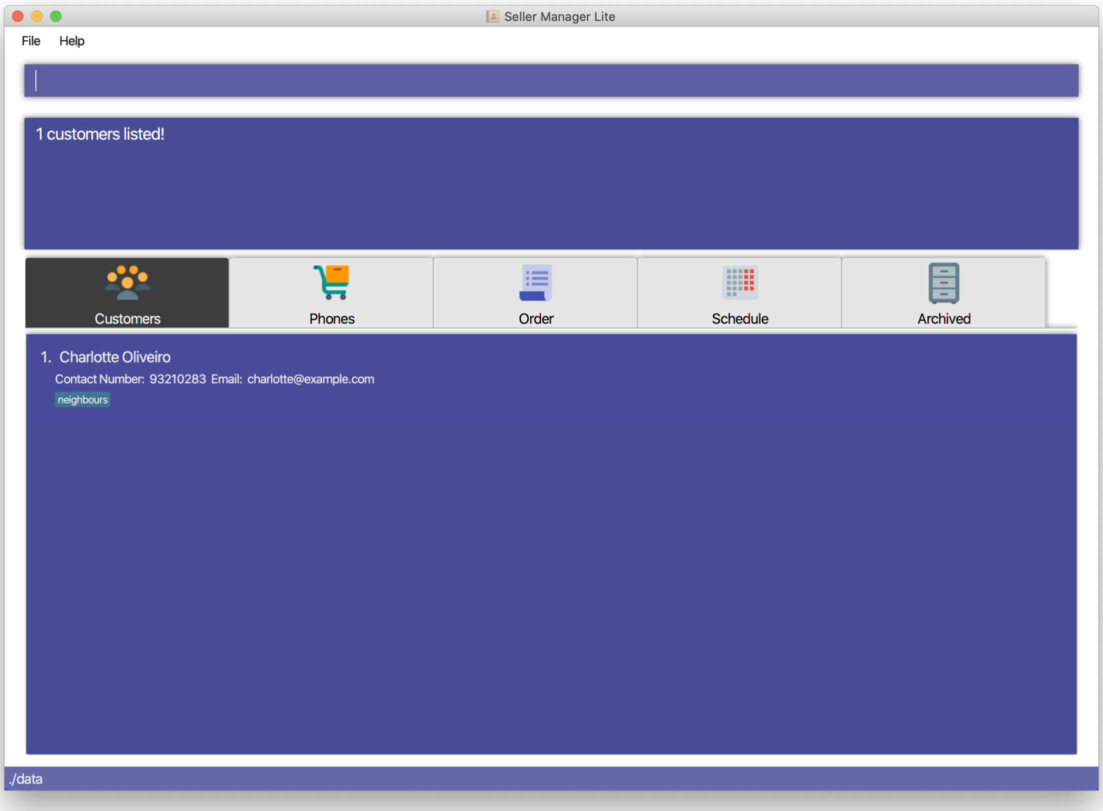


2 customers listed!

1. Bernice Yu
Contact Number: 99272758 Email: berniceyu@example.com
colleagues friends

2. Roy Balakrishnan
Contact Number: 92624417 Email: royb@example.com
colleagues

- Find customers with keywords `charlotte oliveiro`
    1. `find-c charlotte oliveiro`



1 customers listed!

1. Charlotte Oliveiro
Contact Number: 93210283 Email: charlotte@example.com
neighbours

Contributions to the Developer Guide

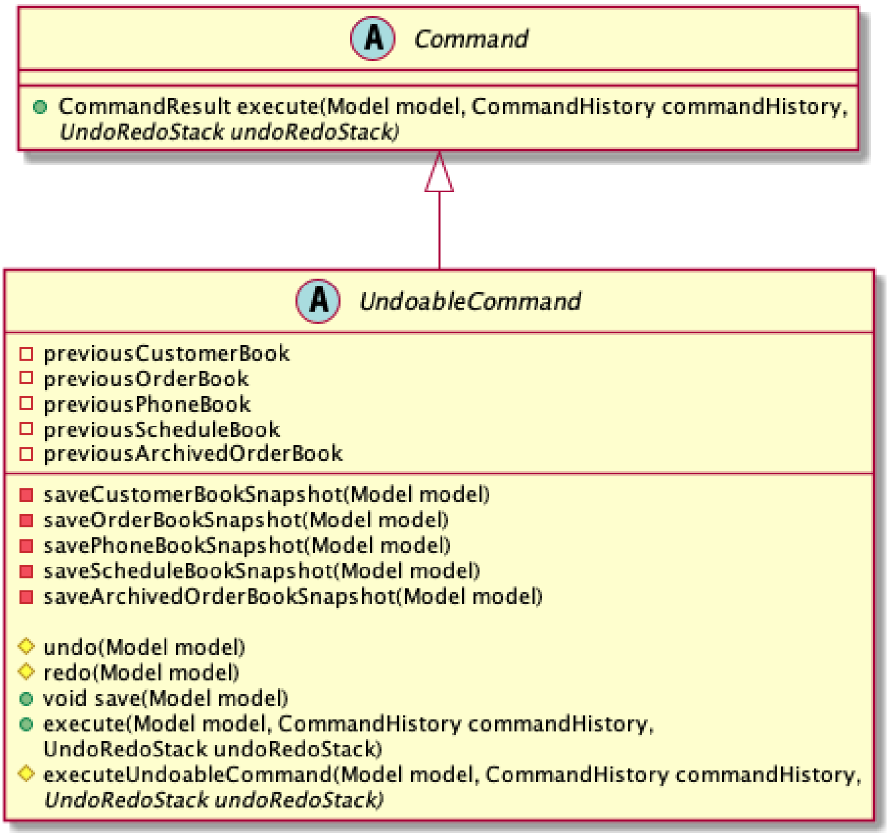
Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

Undo/Redo feature

Current Implementation

The undo/redo mechanism is facilitated by `UndoRedoStack`.

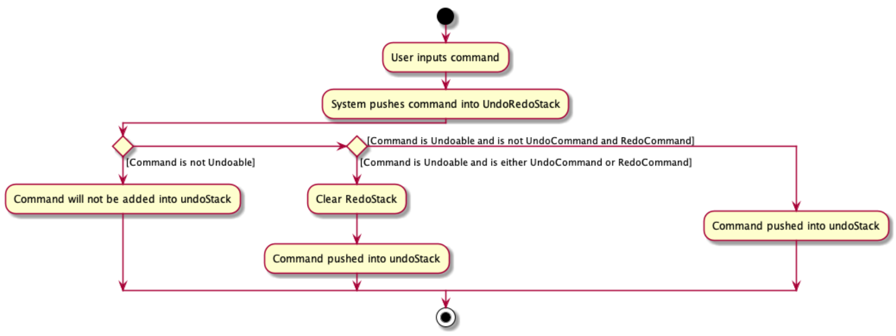
`UndoRedoStack` contains 2 stacks, `undoStack` and `redoStack`. `undoStack` and `redoStack` contains commands that are of type `UndoableCommand`. `UndoableCommand` extends Command and has the following attributes and methods.



A Command
CommandResult execute(Model model, CommandHistory commandHistory, UndoRedoStack undoRedoStack)
A UndoableCommand
previousCustomerBook
previousOrderBook
previousPhoneBook
previousScheduleBook
previousArchivedOrderBook
saveCustomerBookSnapshot(Model model)
saveOrderBookSnapshot(Model model)
savePhoneBookSnapshot(Model model)
saveScheduleBookSnapshot(Model model)
saveArchivedOrderBookSnapshot(Model model)
undo(Model model)
redo(Model model)
void save(Model model)
execute(Model model, CommandHistory commandHistory, UndoRedoStack undoRedoStack)
executeUndoableCommand(Model model, CommandHistory commandHistory, UndoRedoStack undoRedoStack)

When an UndoableCommand is being executed, the methods `saveCustomerBookSnapshot(Model model)`, `savePhoneBookSnapshot(Model model)`, `saveOrderBookSnapshot(Model model)`, `saveScheduleBookSnapshot(Model model)` and `saveArchivedOrderBookSnapshot(Model model)` will be called. This ensures that the states of all 5 books are being stored.

After a command is executed, `LogicManager` will add it into the `UndoRedoStack`. This will be explained in the activity diagram below.

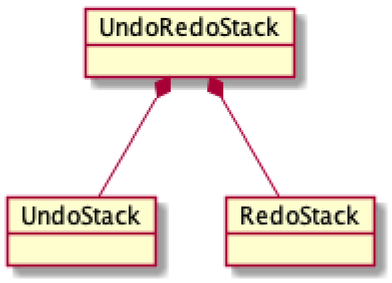

User inputs command
System pushes command into UndoRedoStack
[Command is Undoable and is not UndoCommand and RedoCommand]
[Command is not Undoable]
[Command is Undoable and is either UndoCommand or RedoCommand]
Command will not be added into undoStack
Clear RedoStack
Command pushed into undoStack
Command pushed into undoStack

Next, when `UndoCommand` is being performed, `UndoStack` will remove the first command in its stack and add it to `RedoStack`. It will then call `UndoableCommand#undo()` of the command that is removed. The `undo()` method will then set the model to the previous snapshots of CustomerBook, PhoneBook, OrderBook, ScheduleBook and ArchivedOrderBook. Afterwhich, it will save the original state of the model (e.g. before Undo took place) by calling `UndoableCommand#save(Model model)`.

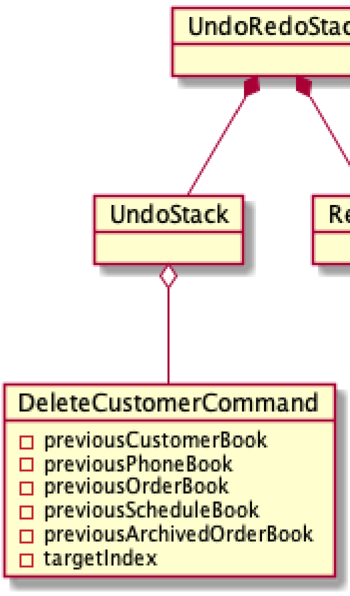
Likewise, when `RedoCommand` is being performed, `RedoStack` will remove the first command in its stack and add it to `UndoStack`. It will then call `UndoableCommand#redo()` of the command that is removed. The `redo()` method will then set the model to the previous snapshots of CustomerBook, PhoneBook, OrderBook, ScheduleBook and ArchivedOrderBook.

Given below is an example usage scenario and how the undo/redo mechanism behaves at each step.

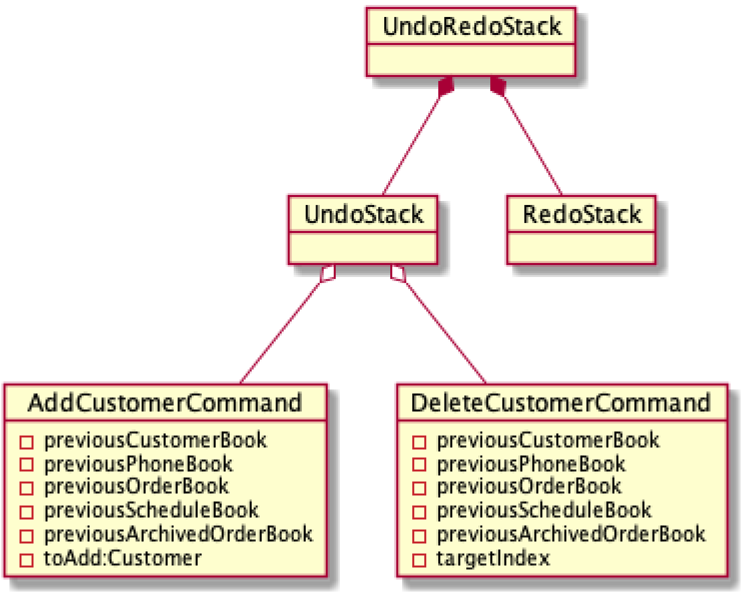
Step 1. The user launches the application for the first time. The `UndoRedoStack` will be initialized.



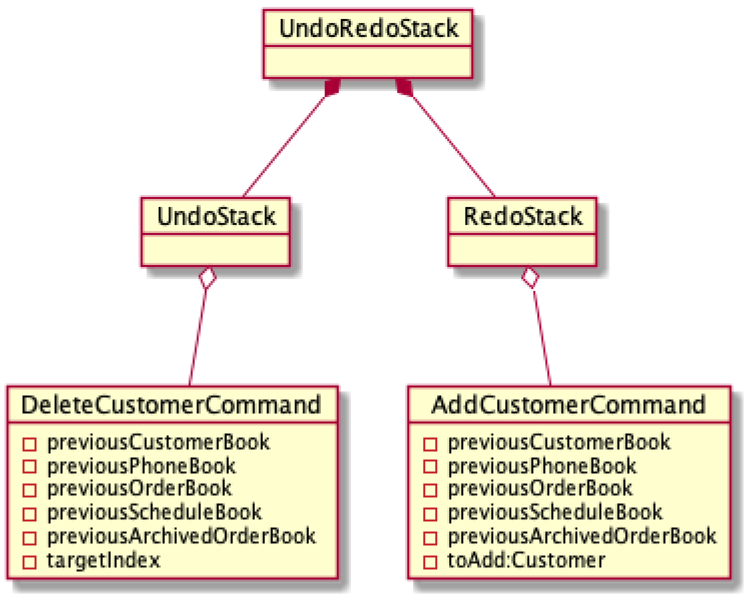
Step 2. The user executes `delete-c 5` command to delete the 5th customer. The `delete-c 5` command will be pushed into the `UndoRedoStack`.



Step 3. The user executes `add-c n/David …` to add a new customer. The `add-c` command will save all states of CustomerBook, PhoneBook, OrderBook, ScheduleBook and ArchivedOrderBook.

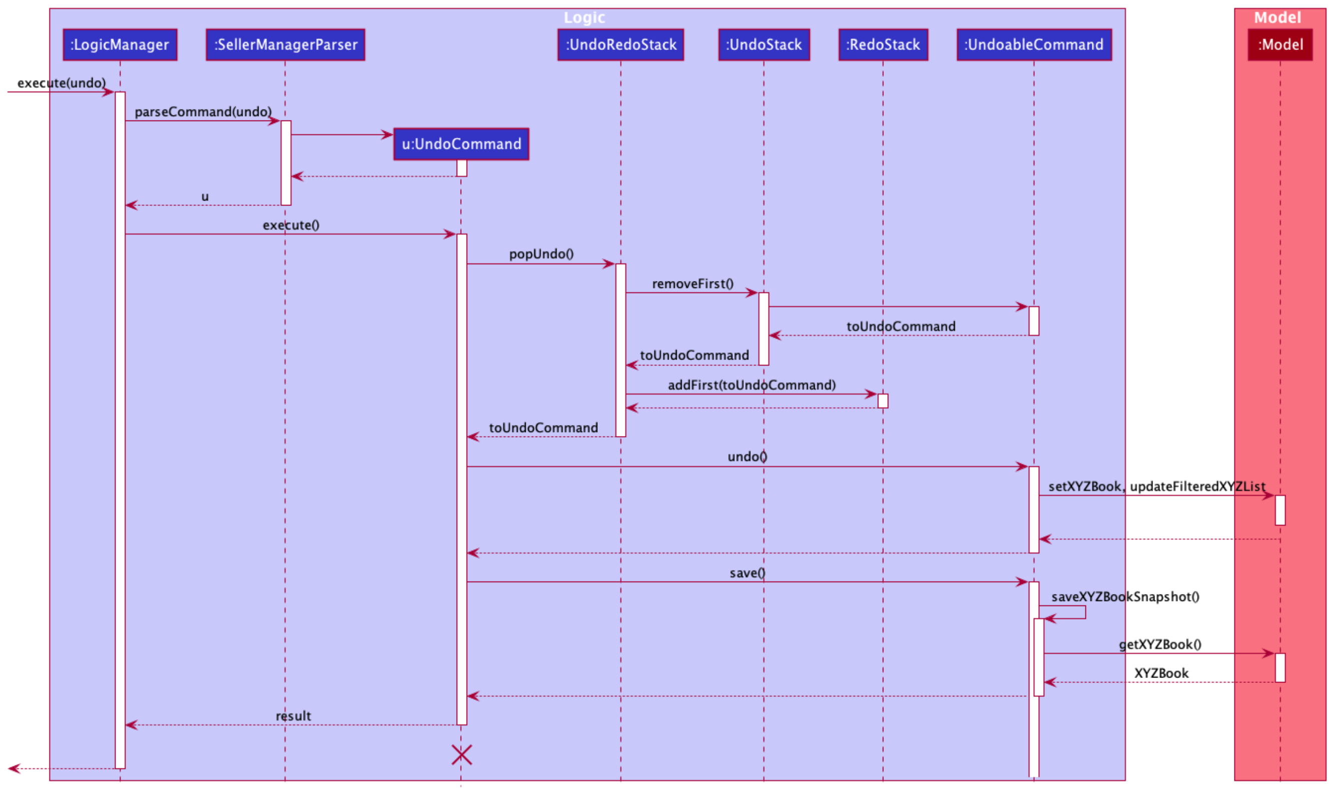


Step 4. The user now decides that adding the customer was a mistake, and decides to undo that action by executing the `undo` command.



NOTE	UndoCommand will check whether if there is any command to be undone by calling the `UndoRedoStack#canUndo()` method.

The following sequence diagram shows how the undo operation works:

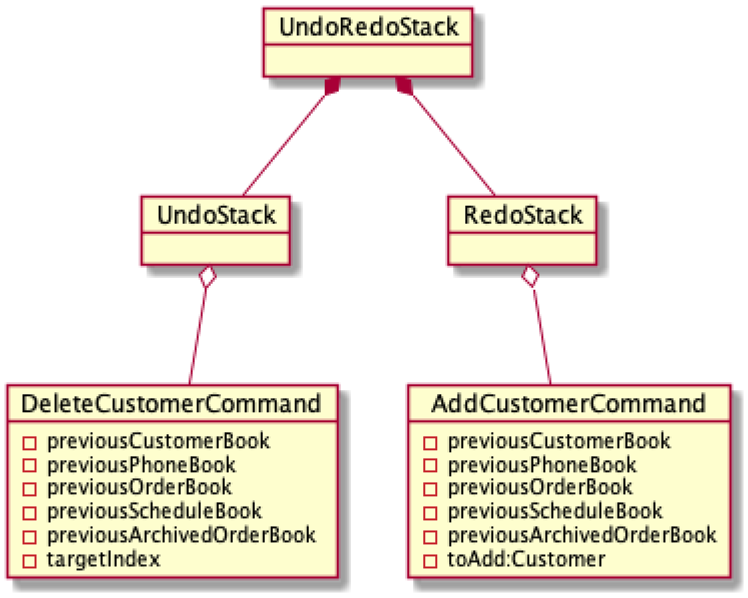


NOTE	The lifeline for `UndoCommand` should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram.

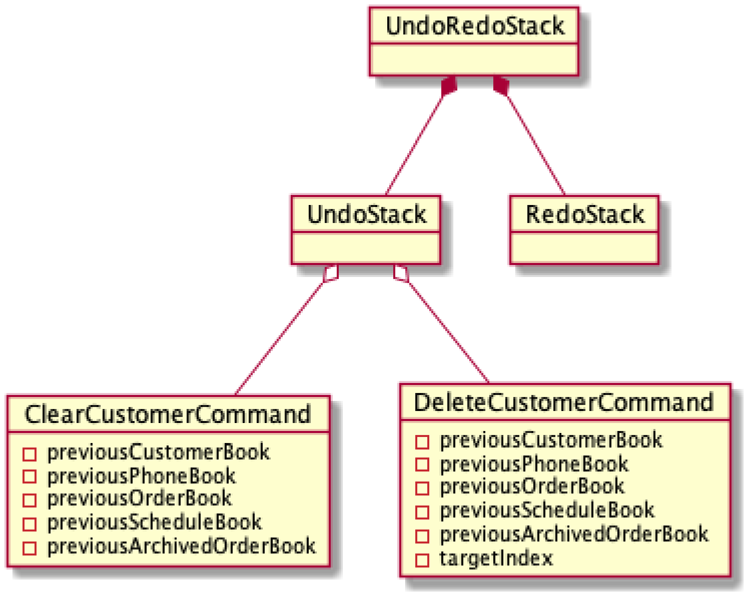
NOTE	XYZ here refers to Customer, Phone, Order, Schedule and ArchivedOrder.

The `redo` command does the opposite — it calls `UndoableCommand#redo()`.

Step 5. The user then decides to execute the command `list-c`. Commands that do not modify the books (Customer, Phone, Order, Schedule and Archived Order), such as `list-c`, do not extend UndoableCommand.



Step 6. The user executes `clear-c`.



Design Considerations

Aspect: How undo & redo executes

- **Alternative 1 (current choice):** Saves the entire mdodel.
  - Pros: Easy to implement.
  - Cons: May have performance issues in terms of memory usage.
- **Alternative 2:** Individual command knows how to undo/redo by itself.
  - Pros: Will use less memory (e.g. for `delete-c`, just save the customer being deleted).
  - Cons: We must ensure that the implementation of each individual command are correct.