Li Zeyong - Project Portfolio

Project: ABC Business Contacts

ABC Business Contacts is a free desktop contact management application that helps the user to manage a large collection of contacts conveniently and keep track of appointments. The user can store contacts as well as other information and retrieve them efficiently with the help of **ABC**.
The user interacts with it using a Command Line Interface(CLI), and it has a Graphic User Interface(GUI) created with JavaFX. It is written in Java, and has about 6 kLoC.

Code contributed: [Functional code] [Test code]

Enhancement Added: `find` Command Reworked

External behavior

Start of Extract [from: User Guide]

Locating persons by keywords: `find`

Command Name: `find`
Shorthand Alias: `f`
Function : Displays a filtered list of persons whose specified fields contain any of the given keywords
Format: `find [n/KEYWORD…] [p/KEYWORD…] [e/KEYWORD…] [a/KEYWORD…] [t/KEYWORD…]`

NOTE	There must be at least one argument

- The search is case insensitive e.g `hans` will match `Hans`
- Only exact words will be matched e.g. `Han` will not match `Hans`
- Persons matching at least one search term in the specified field will be returned e.g. `find n/Hans Bo` will return `Hans Gruber`, `Bo Yang`
- Wildcard symbols `*` and `?` are allowed in the parameters where `*` matches any non-space string and `?` matches any non-space unit-length symbol
- The search is done on the most recent listing. Successive `find` commands make the list smaller

If you want to find a person named `John Watson`:

1. Type in
   `>> find n/john`
   (See Figure 4.4.1)

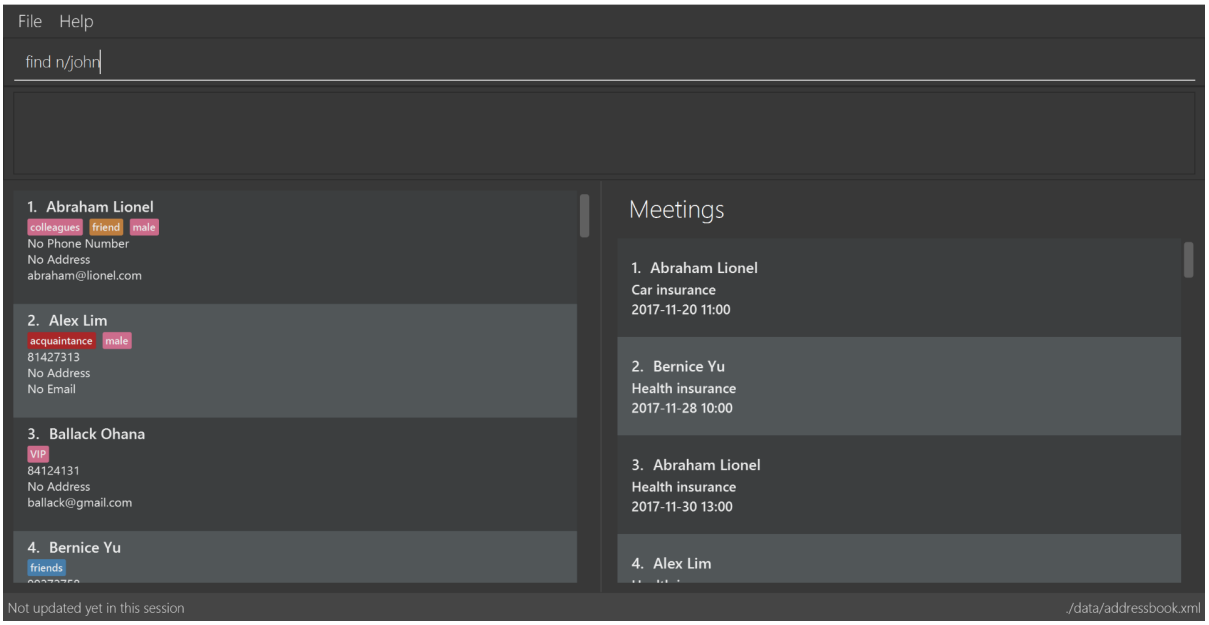


Figure 4.4.1

2. Press `Enter` and you should see a list of persons having the name `john`
(See Figure 4.4.2)

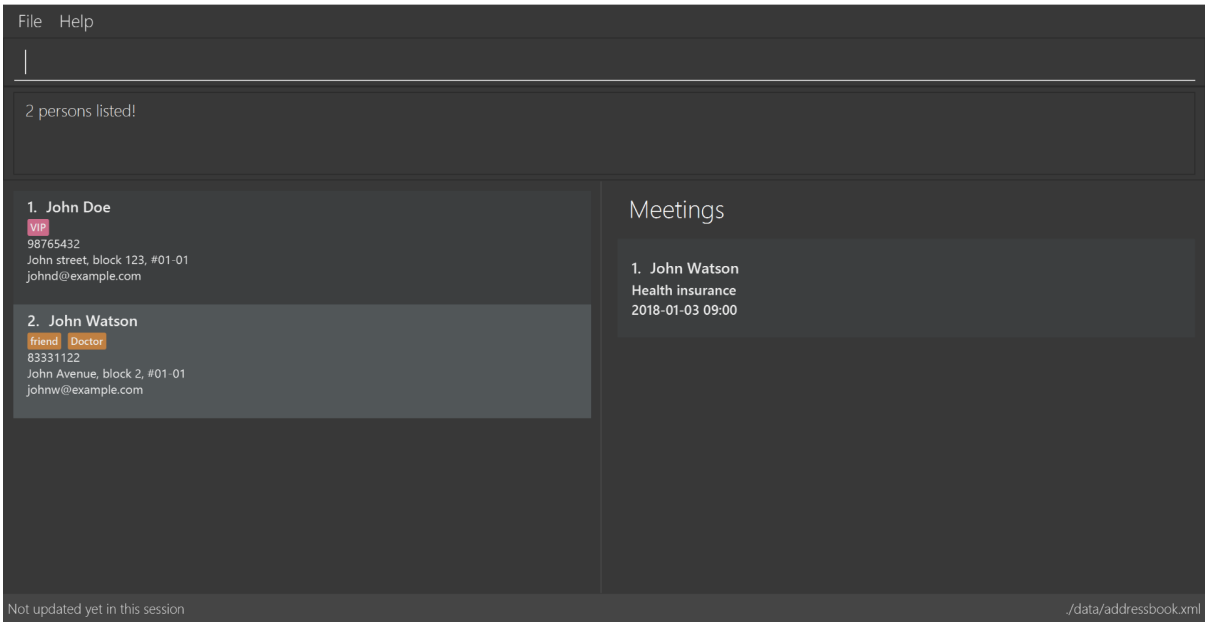


Figure 4.4.2

Here are some other ways you can use `find`:

- `>> find t/friends family p/88887777`
  Lists any person having tags `friends` or `family` or whose phone number is `88887777`.
- `>> f e/*@example.com`
  Lists any person whose email domain is `example.com`.
- `>> find n/steph?n`
  Lists persons whose name is `stephan` or `stephen`.

End of Extract

Key Feature 1: Works on all fields

Justification

The new `find` command now works on all fields, i.e. `Name`, `Phone Number`, `Email`, `Address` and `Tag`. For example, the user can find a contact by phone numbers.
This allows the user to locate specific contacts efficiently using all available information on top of `Name`.

Key Feature 2: Allows wildcard symbols

Justification

Wildcard symbols `*` and `?` are allowed in the parameters to match unknown symbols.
The use of wildcard symbols allows fuzzy search. It enables the user to search even if he forgets certain details.
The use of wildcard symbols also allows the user to search for a group of contacts sharing something similar. For example, the user can use `*@example.com` to search for contacts whose emails share the same domain.

Key Feature 3: Filters the displayed list

Justification

The `find` command filters the displayed list of contacts. Contacts not shown in the most recent listing would **NOT** show up in the result of a `find` command. Therefore, successive `find` commands would make the list smaller and smaller.
By filtering the displayed list, the user can narrow down the list to locate a specific contact without exact information or find a group of contacts sharing multiple similar properties.

Start of Extract [from: Developer Guide]

Filtering mechanism in find

Basic mechanism

The list of persons displayed is filtered by a [Predicate] when the method `updateFilteredPersonList(predicate)` from the `Model` interface is invoked.

The relevant methods in the Model interface are as follows:

```
public interface Model {

  ...

  /** Returns the predicate of the current filtered person list */
  Predicate<? super ReadOnlyPerson> getPersonListPredicate();

  /** Updates the filter of the filtered person list to filter by the given {@code
predicate}.*/
  void updateFilteredPersonList(Predicate<ReadOnlyPerson> predicate);

}
```

When `updateFilteredPersonList(predicate)` is invoked, every `Person` in **ABC** is evaluated against the `predicate`. A `Person` is added to the displayed list if `predicate.test(person)` is evaluated to be TRUE. Therefore, all `Person` instances that fulfill the conditions specified in `predicate` are displayed.

Filtering the displayed list

Note that all `Person` instances in the displayed list satisfy a Predicate `currentPredicate`. Given a new Predicate `newPredicate`, filtering the displayed list of contacts is equivalent to selecting `Person` instances that satisfy both `currentPredicate` and `newPredicate`. From Figure 3.4.2.1, it can also be viewed as the intersection of two lists of `Person` objects, each satisfying one of the two predicates respectively.

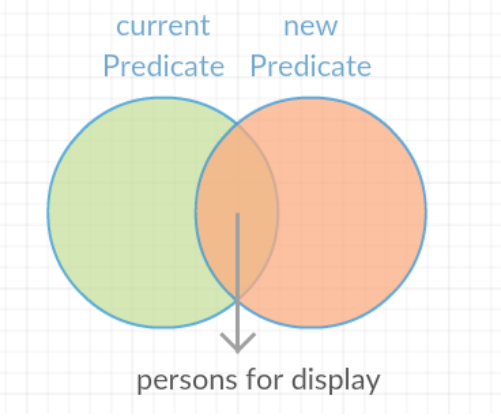


Figure 3.4.2.1 : Venn Diagram for Filtering

Implementation

The actual implementation of filtering the displayed list involves three steps.

1. Invoke `getPersonListPredicate()` provided in the Model interface to get the `currentPredicate`.
2. Use [Predicate.and()] to generated the logical AND of the two predicates.
3. Update the list using the predicate generated in step 2.

For more details, refer to the sequence diagram(Figure 3.4.3.1) below.

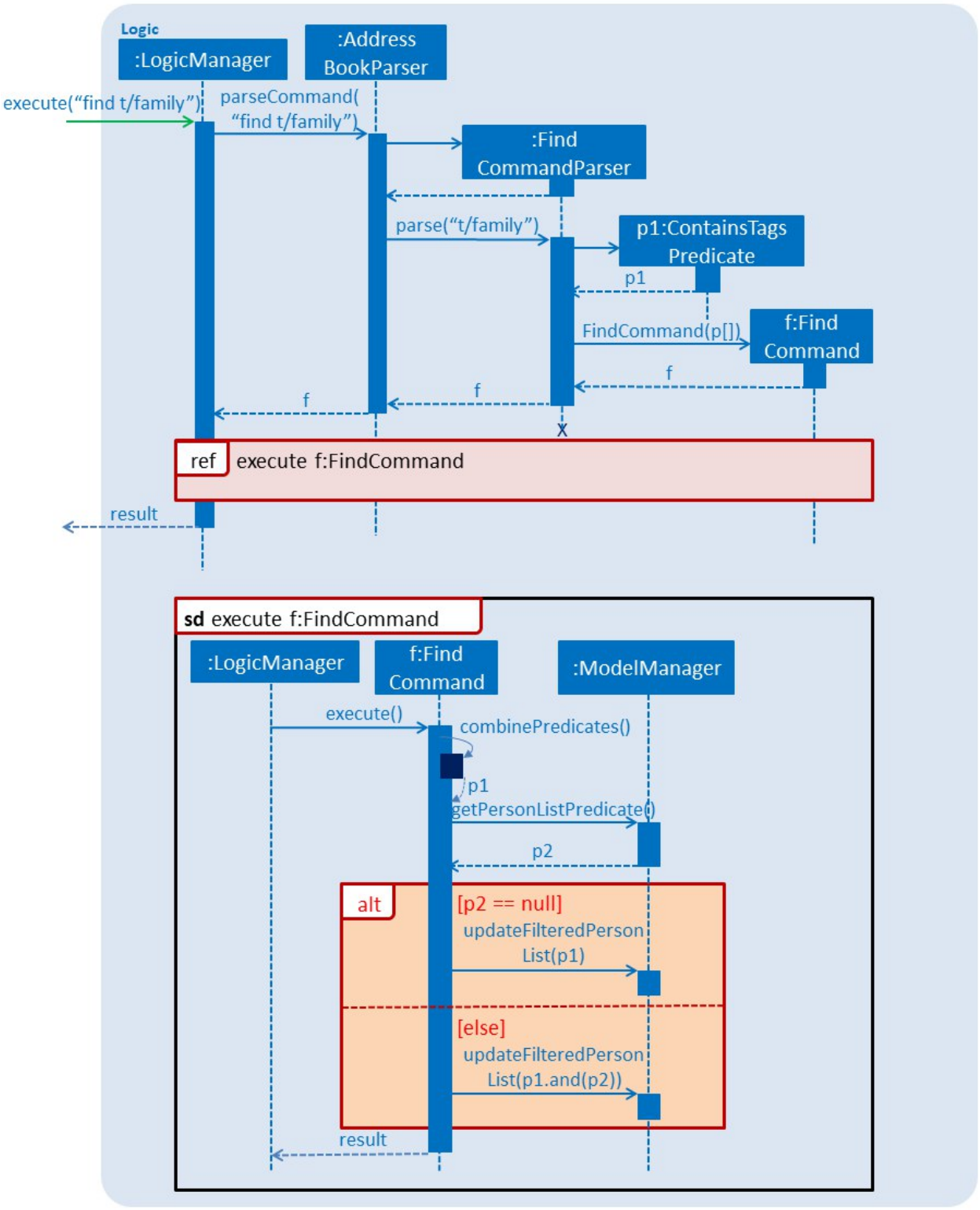


Figure 3.4.3.1 : Sequence Diagram for Find

Design consideration

The design for filtering the displayed list applies the [Open/Close Principle].

- By providing a new extension of `getPersonListPredicate()` in the `Model` interface, the new feature is enabled.
- By making use of the logical AND of two predicates, the list can be filtered without modification of the fundamental filtering mechanism.

End of Extract

Enhancement Added: Auto-Completion Suggestion

External behavior

Start of Extract [from: User Guide]

- Suggestions will pop up for partial words keyed in. Press kbd:[TAB] to auto-complete using the first suggestion or press `Up` and `Down` arrow keys and `Enter` to choose the suggestion.

End of Extract

Suggestions are also generated based on the context of what the user has keyed in. If the user is typing a command word, a list of valid command words would be generated. If the user is keying in tags, a list of valid tags from the stored contact data would be generated.

Justification

Auto-Completion allows users to type less. Moreover, since all suggestions are generated from the stored data, it can reduce the chance of mistyping. For example, if the user wants to find a contact by a very long name, the name would show up as a suggestion when the user type in the first few letters. On the other hand, if the user needs to type the name out in full, there is a higher chance for him to make mistakes.

Enhancement Added: `resize` Command

External behavior

Start of Extract [from: User Guide]

Resizing the main window : `resize`

Command Name: `resize`
Shorthand Alias: `rs`
Function: Resizes the main window to the specified width and height in pixels
Format: `resize WIDTH HEIGHT`

NOTE	Restriction on WIDTH and HEIGHT: `300 < = WIDTH < = width of the screen display`, `230 < = HEIGHT < = height of the screen display`
NOTE	You **CANNOT** `undo` a `resize` command

If you want to resize your main window to 1280 * 720:

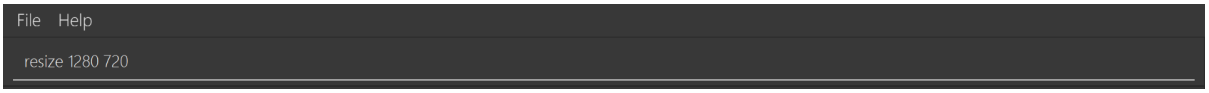
1. Type in
   `>> resize 1280 720`
   (See Figure 4.23.1)
   

   Figure 4.23.1
2. Press `Enter` and the main window will be resized to 1280 * 720

End of Extract

Justification

The resize command allows the user to enlarge/shrink the size of the main window conveniently using CLI so that he can view more contacts or make space in the screen for other applications.

Enhancement Proposed: A More Powerful `find` Command

- The user can choose to search from all contacts instead of the displayed ones using an argument `-a` which stands for `all`. For example, `find -a t/friends` would search for **ALL** contacts that are tagged with `friends`.
- The user can choose to search for contacts that satisfy all conditions using an argument `-s` which stands for `strict`. For example, `find -s n/John t/friends` would search for contacts that match **BOTH** the name `John` the tag `friends`.
- The user can choose to filter away unwanted contacts from the displayed list using an argument `-u` which stands for `unwanted`. For instance, `find -u t/friends` would filter **AWAY** contacts tagged with `friends`.

Enhancement Proposed: Rank Auto-Completion Suggestions by Occurrence

If a certain keyword is frequently used, it would show up first in the suggestion menu.

Other contributions

- Discovered a bug that causes the app to crash and provided a solution. (Issue #137)
- Discovered bugs during acceptance testing. (Issues #116, #117, #118)
- Proposed wildcard symbol feature for reuse. (Issue #132)
- Fix a bug on undo/redo. (Pull Request #172)