Quang Minh Nguyen - Project Portfolio

Project: NeoXPro Manager

NeoXPro is a desktop address book application used for teaching Software Engineering principles. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 6 kLoC.

Code contributed: [Functional code] [Test code] [Unused code] {give links to collated code files}

Enhancement Added: `phone`

External behavior

Locating persons by phone number.

Start of Extract [from: User Guide]

Locating persons by phone number: `phone` (since v1.1)

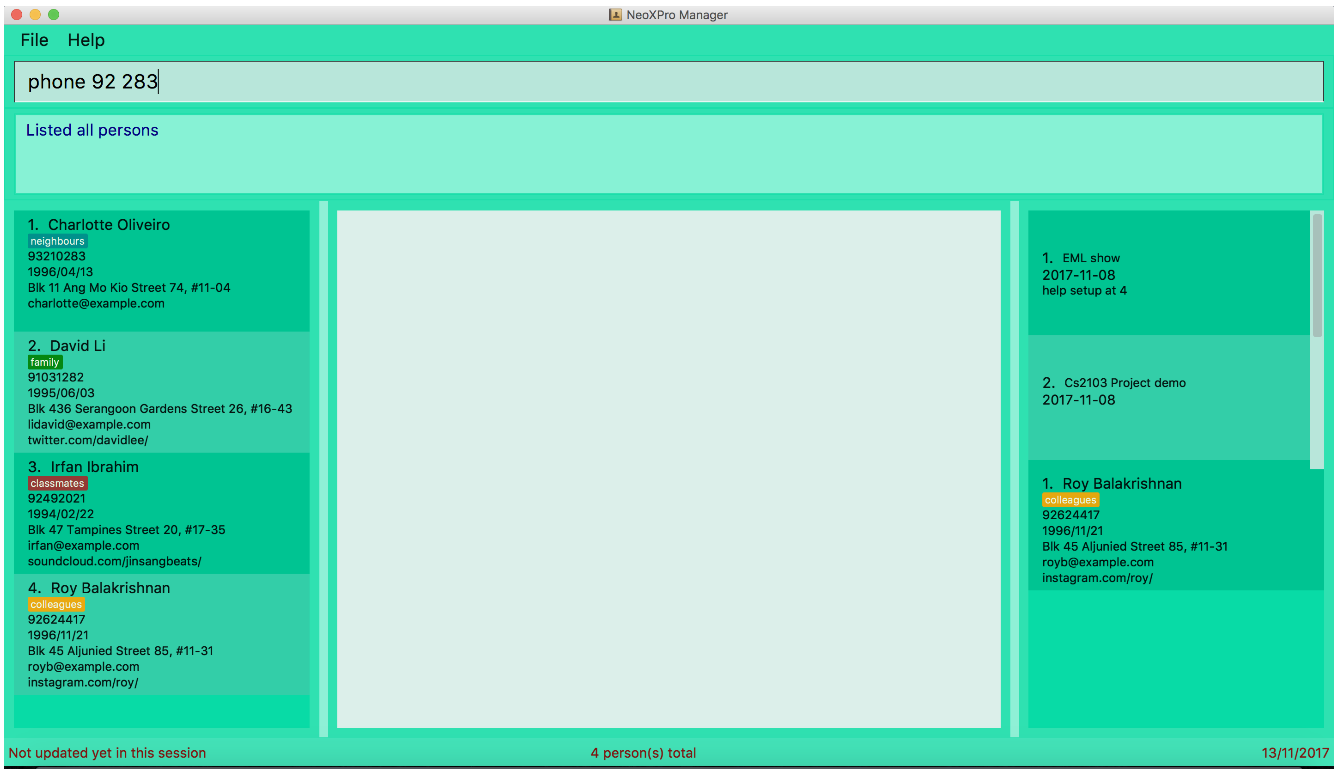
Finds person whose phone numbers partially match with a number in the specified list
Format: `phone NUMBER [MORE_NUMBERS]`

- The order of numbers being queried does not matter.
- Only the phone number is searched.
- Partial string number will be matched with phones e.g. `12345` will match `123456`

Examples:

- `phone 92`
  Returns persons with phone numbers containing 92.
- `phone 92 65`
  Returns persons with phone numbers containing `92` or `283`.

The second example is illustrated below:



phone 92 283

Listed all persons

1. Charlotte Oliveiro
neighbours
93210283
1996/04/13
Blk 11 Ang Mo Kio Street 74, #11-04
charlotte@example.com

2. David Li
family
91031282
1995/06/03
Blk 436 Serangoon Gardens Street 26, #16-43
lidavid@example.com
twitter.com/davidlee/

3. Irfan Ibrahim
classmates
92492021
1994/02/22
Blk 47 Tampines Street 20, #17-35
irfan@example.com
soundcloud.com/jinsangbeats/

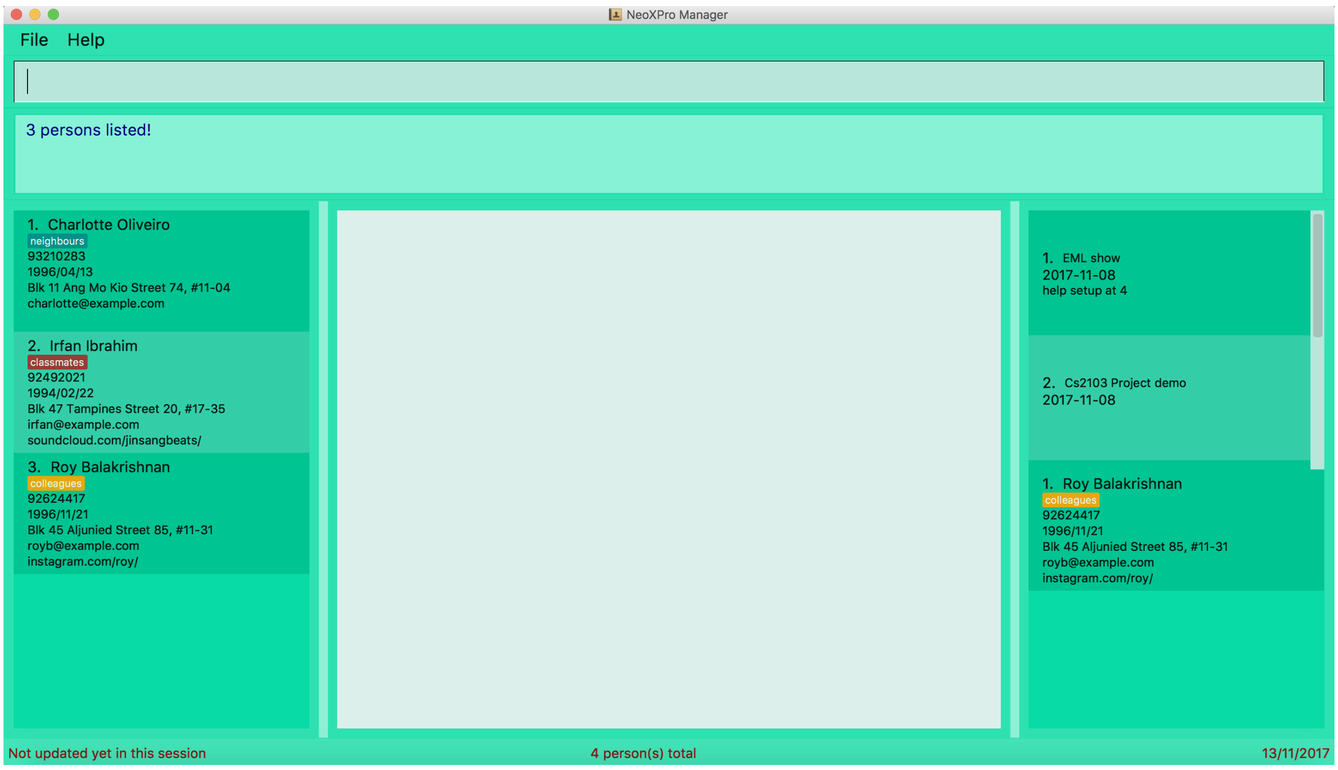
4. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

1. EML show
2017-11-08
help setup at 4

2. Cs2103 Project demo
2017-11-08

1. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

Not updated yet in this session | 4 person(s) total | 13/11/2017



3 persons listed!

1. Charlotte Oliveiro
neighbours
93210283
1996/04/13
Blk 11 Ang Mo Kio Street 74, #11-04
charlotte@example.com

2. Irfan Ibrahim
classmates
92492021
1994/02/22
Blk 47 Tampines Street 20, #17-35
irfan@example.com
soundcloud.com/jinsangbeats/

3. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

1. EML show
2017-11-08
help setup at 4

2. Cs2103 Project demo
2017-11-08

1. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

Not updated yet in this session | 4 person(s) total | 13/11/2017

Locating persons by phone number: `phone` (since v1.1)

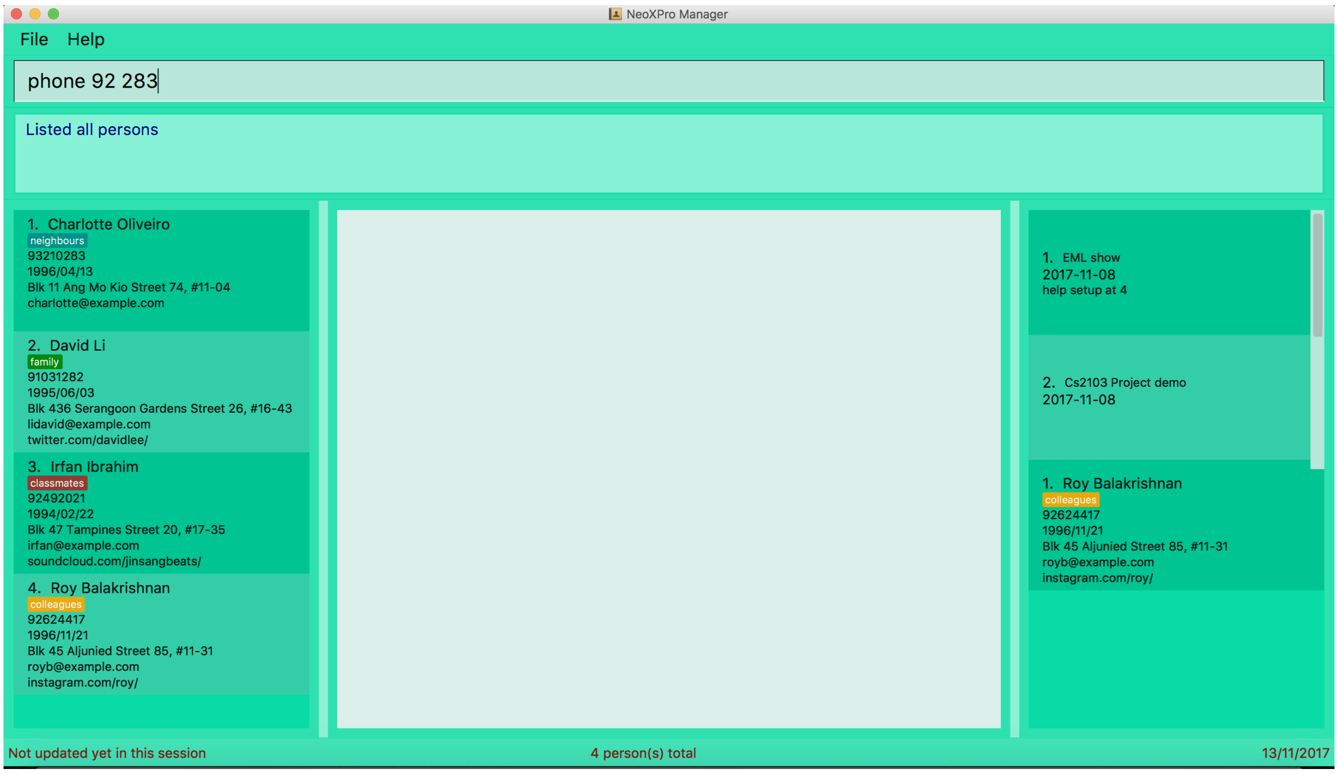
Finds person whose phone numbers partially match with a number in the specified list
Format: `phone NUMBER [MORE_NUMBERS]`

- The order of numbers being queried does not matter.
- Only the phone number is searched.
- Partial string number will be matched with phones e.g. `12345` will match `123456`

Examples:

- `phone 92`
  Returns persons with phone numbers containing 92.
- `phone 92 65`
  Returns persons with phone numbers containing `92` or `283`.

The second example is illustrated below:



phone 92 283

Listed all persons

1. Charlotte Oliveiro
neighbours
93210283
1996/04/13
Blk 11 Ang Mo Kio Street 74, #11-04
charlotte@example.com

2. David Li
family
91031282
1995/06/03
Blk 436 Serangoon Gardens Street 26, #16-43
lidavid@example.com
twitter.com/davidlee/

3. Irfan Ibrahim
classmates
92492021
1994/02/22
Blk 47 Tampines Street 20, #17-35
irfan@example.com
soundcloud.com/jinsangbeats/

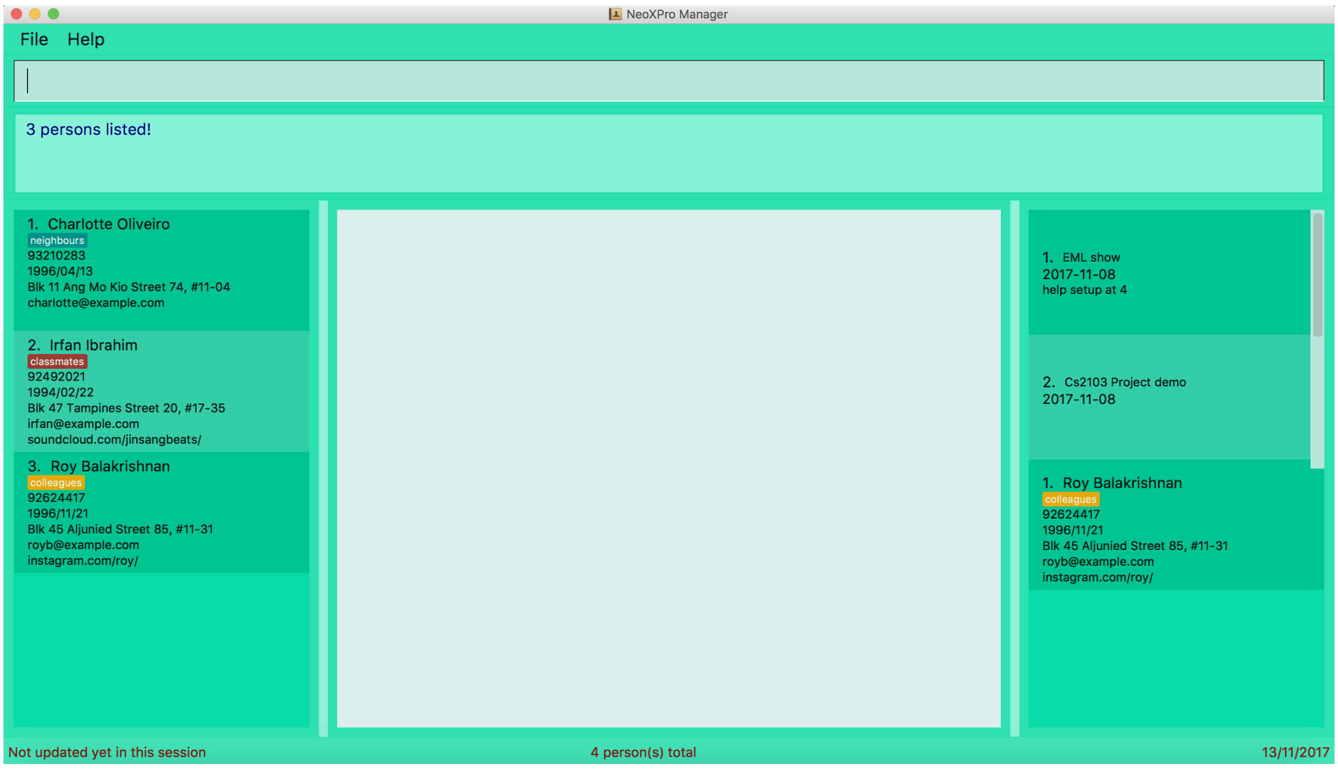
4. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

1. EML show
2017-11-08
help setup at 4

2. Cs2103 Project demo
2017-11-08

1. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

Not updated yet in this session
4 person(s) total
13/11/2017


NeoXPro Manager

File Help

3 persons listed!

1. Charlotte Oliveiro
neighbours
93210283
1996/04/13
Blk 11 Ang Mo Kio Street 74, #11-04
charlotte@example.com

2. Irfan Ibrahim
classmates
92492021
1994/02/22
Blk 47 Tampines Street 20, #17-35
irfan@example.com
soundcloud.com/jinsangbeats/

3. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

1. EML show
2017-11-08
help setup at 4

2. Cs2103 Project demo
2017-11-08

1. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

Not updated yet in this session 4 person(s) total 13/11/2017

End of Extract

Justification

Searching a contact by partial chain of number is a convenient and efficient feature especially when user has to deal with a large contact list.
For example, a user Michael wants to search for his best friend John Cena in NeoXPro. Unfortunately there are 6 contacts with the same name John Cena stored in NeoXPro so using `find` command confuses Michael a lot with 6 output contacts. However, using `phone` command, Michael can directly search for his best friend immediately via his phone.

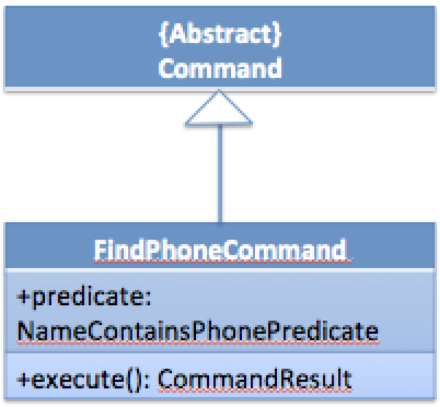
Implementation

Start of Extract [from: Developer Guide]

Phone command

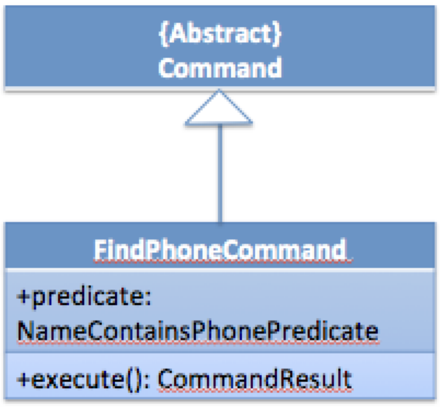
The `phone` command utilize the same implementation as the `Find` command for name. Instead of logic execute search via `Name` attribute of `Person`, the command search for the `phone` attribute.

The `phone` command is handled by the class `FindPhoneCommand` that inherits from `Command` class.
`Name` and `Phone` API structure is roughly similar that they allow to extract value of the object. The search algorithm utilizes a class `NameContainsPhonesPredicate implements Predicate<ReadOnlyPerson>` which allows the algorithm to use Java `Predicate` class method.
The diagram demonstrating `phone` command structure is illustrated here:


{Abstract}
Command
FindPhoneCommand
+predicate: NameContainsPhonePredicate
+execute(): CommandResult

The phone command utilize the same implementation as the Find command for name. Instead of logic execute search via Name attribute of Person, the command search for the phone attribute.

The phone command is handled by the class FindPhoneCommand that inherits from Command class.
Name and Phone API structure is roughly similar that they allow to extract value of the object. The search algorithm utilizes a class NameContainsPhonesPredicate implements Predicate<ReadOnlyPerson> which allows the algorithm to use Java Predicate class method.
The diagram demonstrating phone command structure is illustrated here:


{Abstract}
Command
FindPhoneCommand
+predicate: NameContainsPhonePredicate
+execute(): CommandResult

End of Extract

Enhancement Added: Profile Page

External behavior

Allow user to optionally store the profile page of each person in contact list. The profile page for a person will appear in UI as long as it is stored. When a person is selected, his/her profile page will be loaded in UI.

Start of Extract [from: User Guide]

Adding a person: `add`

Adds a person to the address book
Format: `add n/NAME [p/PHONE_NUMBER] [e/EMAIL] [b/BIRTHDAY] [a/ADDRESS] [pr/PROFILE_PAGE] [t/TAG]…`

TIP A person can have any number of tags (including 0). All fields are optional except the person name.

Examples:

- `add n/John Doe p/98765432 e/johnd@example.com b/1995/5/21 a/John street, block 123, #01-01 pr/www.facebook.com/john`
- `add n/Betsy Crowe t/friend e/betsycrowe@example.com a/Newgate Prison p/1234567 b/1999/10/10 t/criminal`

Adds a person to the address book
Format: `add n/NAME [p/PHONE_NUMBER] [e/EMAIL] [b/BIRTHDAY] [a/ADDRESS] [pr/PROFILE_PAGE] [t/TAG]…`

TIP A person can have any number of tags (including 0). All fields are optional except the person name.

Examples:

- `add n/John Doe p/98765432 e/johnd@example.com b/1995/5/21 a/John street, block 123, #01-01 pr/www.facebook.com/john`
- `add n/Betsy Crowe t/friend e/betsycrowe@example.com a/Newgate Prison p/1234567 b/1999/10/10 t/criminal` End of Extract

Justification

With the profile page stored for each person, NeoXPRO enhances the user's experience.

Implementation

Start of Extract [from: Developer Guide]

Add Profile Page

The profile page parameter is facilitated by the class `ProfilePage`.
In order to add this parameter profile page, we make modifications to UI, Logic, Model and Storage components.

- `UI Component:`
  `PersonCard`, which resides in UI component, frist creates a label for profile page parameter:

```
@FXML
    private Label profile;
```

Then `PersonCard` binds the label with the value of `ProfilePage` object. If the input value for `ProfilePage` object is null, it make the label dissapear from UI by setting its visibility to FALSE:

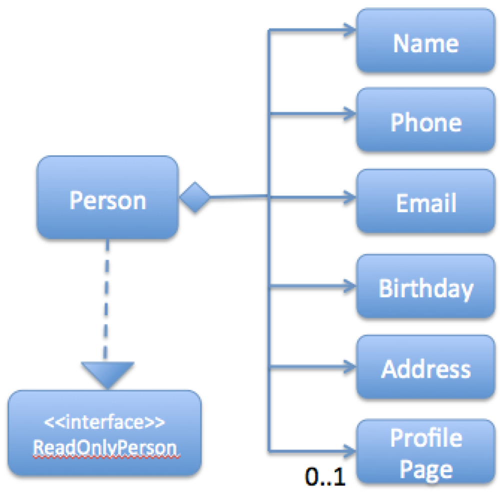
```
private void bindListeners(ReadOnlyPerson person) {
    //... binding other labels ...

    if (!person.profilepageProperty().toString().equals("")) {
        profile.textProperty().bind(Bindings.convert(person.profilepageProperty()));
        profile.setVisible(true);
    } else {
        profile.setVisible(false);
    }
    //... binding other labels ...
}
```

In order to make `select` command load a person's `ProfilePage` object if it exists, we modify method `handlePersonPanelSelectionChangedEvent()` in `BrowserPanel` that is in charge of updating a person panel:

```
 private void handlePersonPanelSelectionChangedEvent(PersonPanelSelectionChangedEvent
event) {
     //...
     if (person.getProfilePage().hasProfilePage()) {
         loadProfilePage(person);
     } else {
         loadPersonPage(person);
     }
 }
```

- `Logic Component:`
  The prefix for profile page `pr/` is added to `CliSyntax`. Then the following files `AddCommand`, `EditCommand`, `AddCommandParser` and `EditCommandParser` are modified so that `add` and `edit` commands accept the new parameter profile page.
  `AddCommandParser` does not check for profile page prefix `pr/` as this is an optional parameter.
- `Model Component:`
  Class `ProfilePage` is used to store profile page property of class `Person`. Class `Person`, which resides in model component and implements `ReadOnlyPerson` interface, form a composition association with `ProfilePage` .
  As profile page is an optional parameter, a `Person` can be linked to 0 or 1 `ProfilePage` object.
  The relationship is illustrated in the following diagram:



- `Storage Component:`
  `xmlAdaptedPerson` file is used to save information of a person in xml format.
  The `required` parameter of `@XmlElement` element which stores profile page information is set to false to make this property optional:

```
@XmlElement(required = false)
    private String profile = "";
```

The profile page parameter is facilitated by the class `ProfilePage`.
In order to add this parameter profile page, we make modifications to UI, Logic, Model and Storage components.

- `UI Component:`
  `PersonCard`, which resides in UI component, frist creates a label for profile page parameter:

```
@FXML
    private Label profile;
```

Then `PersonCard` binds the label with the value of `ProfilePage` object. If the input value for `ProfilePage` object is null, it make the label dissapear from UI by setting its visibility to FALSE:

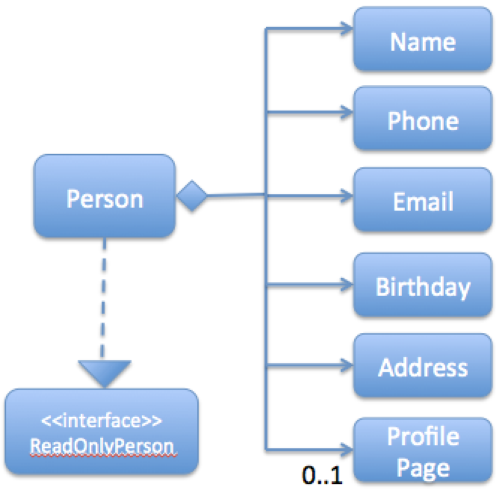
```
private void bindListeners(ReadOnlyPerson person) {
    //... binding other labels ...

    if (!person.profilepageProperty().toString().equals("")) {
        profile.textProperty().bind(Bindings.convert(person.profilepageProperty()));
        profile.setVisible(true);
    } else {
        profile.setVisible(false);
    }
    //... binding other labels ...
}
```

In order to make `select` command load a person's `ProfilePage` object if it exists, we modify method `handlePersonPanelSelectionChangedEvent()` in `BrowserPanel` that is in charge of updating a person panel:

```
 private void handlePersonPanelSelectionChangedEvent(PersonPanelSelectionChangedEvent
event) {
     //...
     if (person.getProfilePage().hasProfilePage()) {
         loadProfilePage(person);
     } else {
         loadPersonPage(person);
     }
 }
```

- `Logic Component:`
  The prefix for profile page `pr/` is added to `CliSyntax`. Then the following files `AddCommand`, `EditCommand`, `AddCommandParser` and `EditCommandParser` are modified so that `add` and `edit` commands accept the new parameter profile page.
  `AddCommandParser` does not check for profile page prefix `pr/` as this is an optional parameter.
- `Model Component:`
  Class `ProfilePage` is used to store profile page property of class `Person`. Class `Person`, which resides in model component and implements `ReadOnlyPerson` interface, form a composition association with `ProfilePage` .
  As profile page is an optional parameter, a `Person` can be linked to 0 or 1 `ProfilePage` object.
  The relationship is illustrated in the following diagram:



- `Storage Component:`
  `xmlAdaptedPerson` file is used to save information of a person in xml format.
  The `required` parameter of `@XmlElement` element which stores profile page information is set to false to make this property optional:

```
@XmlElement(required = false)
    private String profile = "";
```

End of Extract

Enhancement Added: Improve `delete` command

External behavior

Using 'delete' command, user can now delete multiple persons from the contact list at a time.

Start of Extract [from: User Guide]

Deleting a person : `delete`

Deletes a list of specified persons from the address book.
Format: `delete INDEX [MORE_INDICES]`

> - Deletes the persons at the specified `INDEX’s.
> - The index refers to the index number shown in the most recent listing.
> - The index **must be a positive integer** 1, 2, 3, …

Examples:

- `list`
  `delete 2 1`
  Deletes the 1st and 2nd person in the address book.
- `find Betsy`
  `delete 1`
  Deletes the 1st person in the results of the `find` command.

Deletes a list of specified persons from the address book.
Format: `delete INDEX [MORE_INDICES]`

> - Deletes the persons at the specified `INDEX’s.
> - The index refers to the index number shown in the most recent listing.
> - The index **must be a positive integer** 1, 2, 3, …

Examples:

- `list`
  `delete 2 1`
  Deletes the 1st and 2nd person in the address book.

- `find Betsy`
  `delete 1`
  Deletes the 1st person in the results of the `find` command. End of Extract

---

Justification

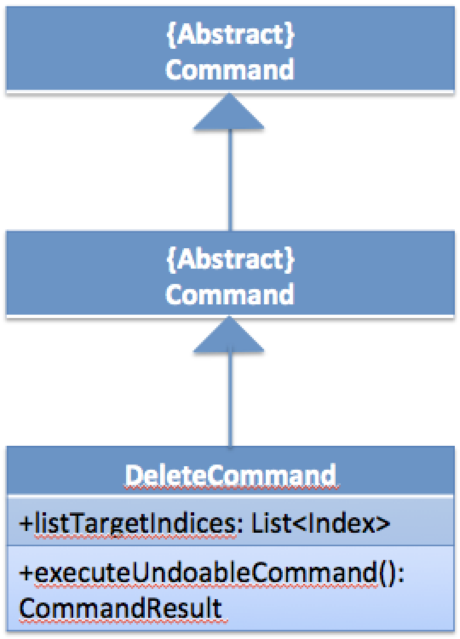
Users now don't have to manually delete each contact at a time.

Implementation

Start of Extract [from: Developer Guide]

Delete command

`delete` command supports modifying the state of address book by deleting all persons whose indices are specified in the input. It inherits from `UndoableCommand`.

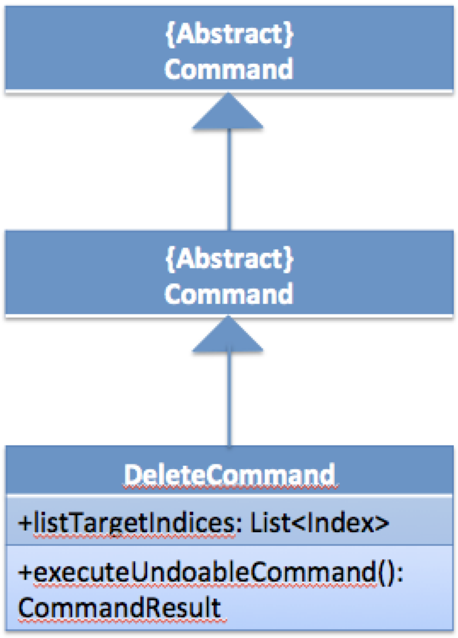

{Abstract}
Command
{Abstract}
Command
DeleteCommand
+listTargetIndices: List<Index>
+executeUndoableCommand(): CommandResult

The implementation of `delete` contains 2 classes: `DeleteCommand` and `DeleteCommandParser` inside the logic component.
`DeleteCommandParser`, the parser of `delete`, parses user's input into the variable `input: List<Index>` that store a list of `Index`. `DeleteCommand`, which handles the logic of `delete` command, then iteratively remove any `Person` object with `Index` specified in `input`.

```
public class DeleteCommandParser implements Parser<DeleteCommand> {
    public DeleteCommand parse(String args) throws ParseException {
        try {
            List<Index> input= new ArrayList<Index>();
            // ... Parser logic ...
            return new DeleteCommand(input);
        } catch (IllegalValueException ive) {
            // throw exception here
        }
    }
}
```

And finally, we add the `delete` command to the class 'AddressBookParser' so that `delete` command is recognized whenever invoked.

`delete` command supports modifying the state of address book by deleting all persons whose indices are specified in the input. It inherits from `UndoableCommand`.


{Abstract}
Command
{Abstract}
Command
DeleteCommand
+listTargetIndices: List<Index>
+executeUndoableCommand(): CommandResult

The implementation of `delete` contains 2 classes: `DeleteCommand` and `DeleteCommandParser` inside the logic component.
`DeleteCommandParser`, the parser of `delete`, parses user's input into the variable `input: List<Index>` that store a list of `Index`. `DeleteCommand`, which handles the logic of `delete` command, then iteratively remove any `Person` object with `Index` specified in `input`.

```
public class DeleteCommandParser implements Parser<DeleteCommand> {
    public DeleteCommand parse(String args) throws ParseException {
        try {
            List<Index> input= new ArrayList<Index>();
            // ... Parser logic ...
            return new DeleteCommand(input);
        } catch (IllegalValueException ive) {
            // throw exception here
        }
    }
}
```

And finally, we add the `delete` command to the class 'AddressBookParser' so that `delete` command is recognized whenever invoked. End of Extract

---

Enhancement Added: `export`

External behavior

Export the contact list of NeoXPRO into a text file.

Start of Extract [from: User Guide]

Exporting the data: `export`

Address book data is exported in text form to the input file path. Format: `export [File_Path]`

> - The [File_Path] must contain the file name of the exported file. E.g. `export c:\documents and settings\all users\desktop\exportedData`
> - The [File_Path] can be just the file name of the exported file instead of the file path. In this case, `export` command will export the file [File_Path] to the current directory.
> - The [File_Path] can be blank. In this case, `export` command will export the file with default name "exportFile.txt" to the current directory.

Examples:

- `export c:\documents and settings\all users\desktop\exportedData`
  Exports the file "exportedData" to the specified path.
- `export exportedData`
  Exports the file "exportedData" to the current directory of NeoXPro.
- `export`
  Exports the file "exportFile.txt" to the current directory of NeoXPro.

Address book data is exported in text form to the input file path. Format: `export [File_Path]`

- The [File_Path] must contain the file name of the exported file. E.g. `export c:\documents and settings\all users\desktop\exportedData`
- The [File_Path] can be just the file name of the exported file instead of the file path. In this case, `export` command will export the file [File_Path] to the current directory.
- The [File_Path] can be blank. In this case, `export` command will export the file with default name "exportFile.txt" to the current directory.

Examples:

- `export c:\documents and settings\all users\desktop\exportedData`
  Exports the file "exportedData" to the specified path.
- `export exportedData`
  Exports the file "exportedData" to the current directory of NeoXPro.
- `export`
  Exports the file "exportFile.txt" to the current directory of NeoXPro. End of Extract

---

Justification

With this feature, user can easily make use of the data in contact list.

Implementation

Start of Extract [from: Developer Guide]

Export Command

`export` access the file `addressbook.xml` to retrieve each person and his/her properties and write it on the file at the specified input. Its implementation contains 2 classes: `ExportCommand` and `ExportCommandParser`.
The parser `ExportCommandParser` of `export` parses the file path input as a `String` to `ExportCommand`.
`ExportCommand`, which handles the logic of `export` command, then writes the exported file on the specified input file path.

`export` access the file `addressbook.xml` to retrieve each person and his/her properties and write it on the file at the specified input. Its implementation contains 2 classes: `ExportCommand` and `ExportCommandParser`.
The parser `ExportCommandParser` of `export` parses the file path input as a `String` to `ExportCommand`.
`ExportCommand`, which handles the logic of `export` command, then writes the exported file on the specified input file path.

End of Extract

Enhancement Added: Improve `Select` command

External behavior

When a person is selected, his/her profile page is displayed (if it’s stored)

Start of Extract [from: User Guide]

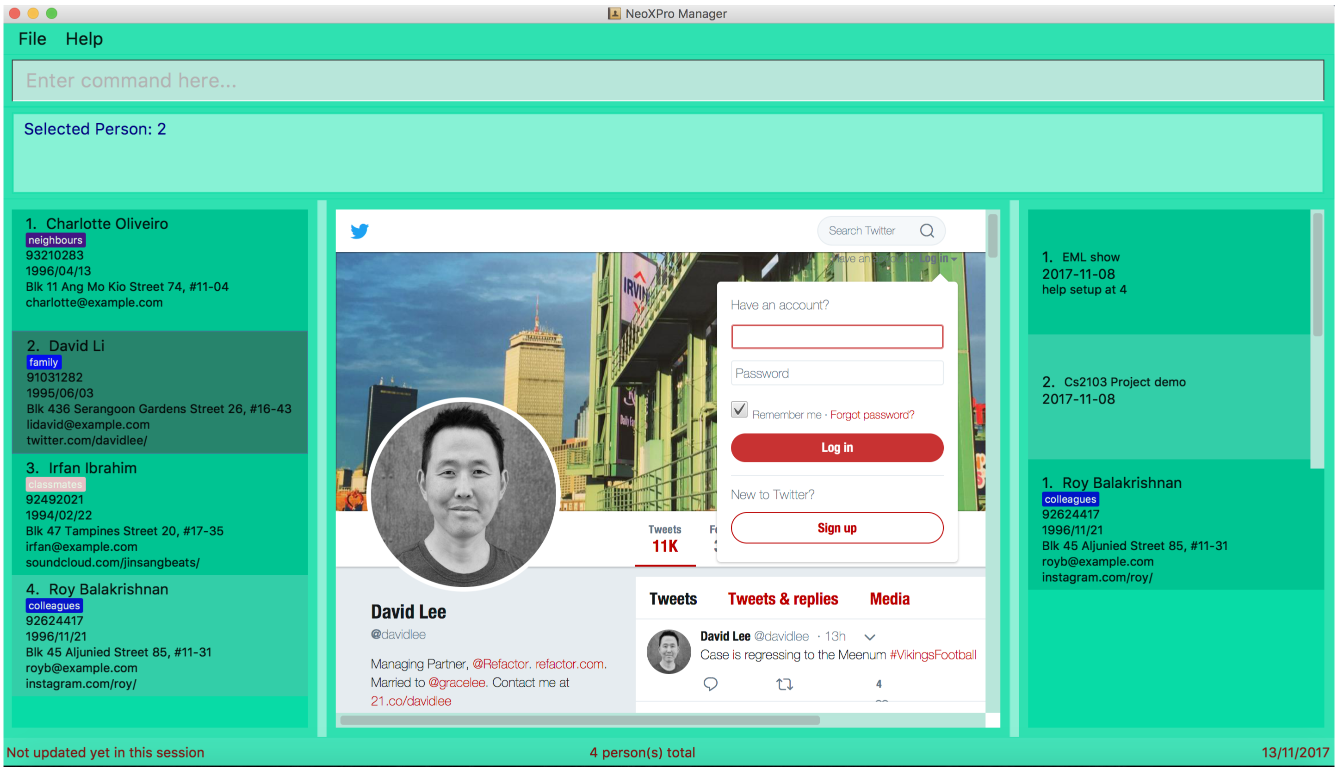
Selecting a person : `select`

Selects the person identified by the index number used in the last person listing.
Format: `select INDEX`

- Selects the person at the specified `INDEX`.
- If the person has a valid profile page, address book loads that profile page. Otherwise, it loads the Google search page of the person.
- The index refers to the index number shown in the most recent listing.
- The index **must be a positive integer** `1, 2, 3, …`

Examples:

- `list`
  `select 2`
  Selects the 2nd person in the address book.
  The 2nd person in the list has the profile page "twitter.com/davidlee"



Selected Person: 2

1. Charlotte Oliveiro
neighbours
93210283
1996/04/13
Blk 11 Ang Mo Kio Street 74, #11-04
charlotte@example.com

2. David Li
family
91031282
1995/06/03
Blk 436 Serangoon Gardens Street 26, #16-43
lidavid@example.com
twitter.com/davidlee/

3. Irfan Ibrahim
classmates
92492021
1994/02/22
Blk 47 Tampines Street 20, #17-35
irfan@example.com
soundcloud.com/jinsangbeats/

4. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

David Lee
@davidlee
Managing Partner, @Refactor. refactor.com.
Married to @gracelee. Contact me at
21.co/davidlee

David Lee @davidlee · 13h
Case is regressing to the Meenum #VikingsFootball

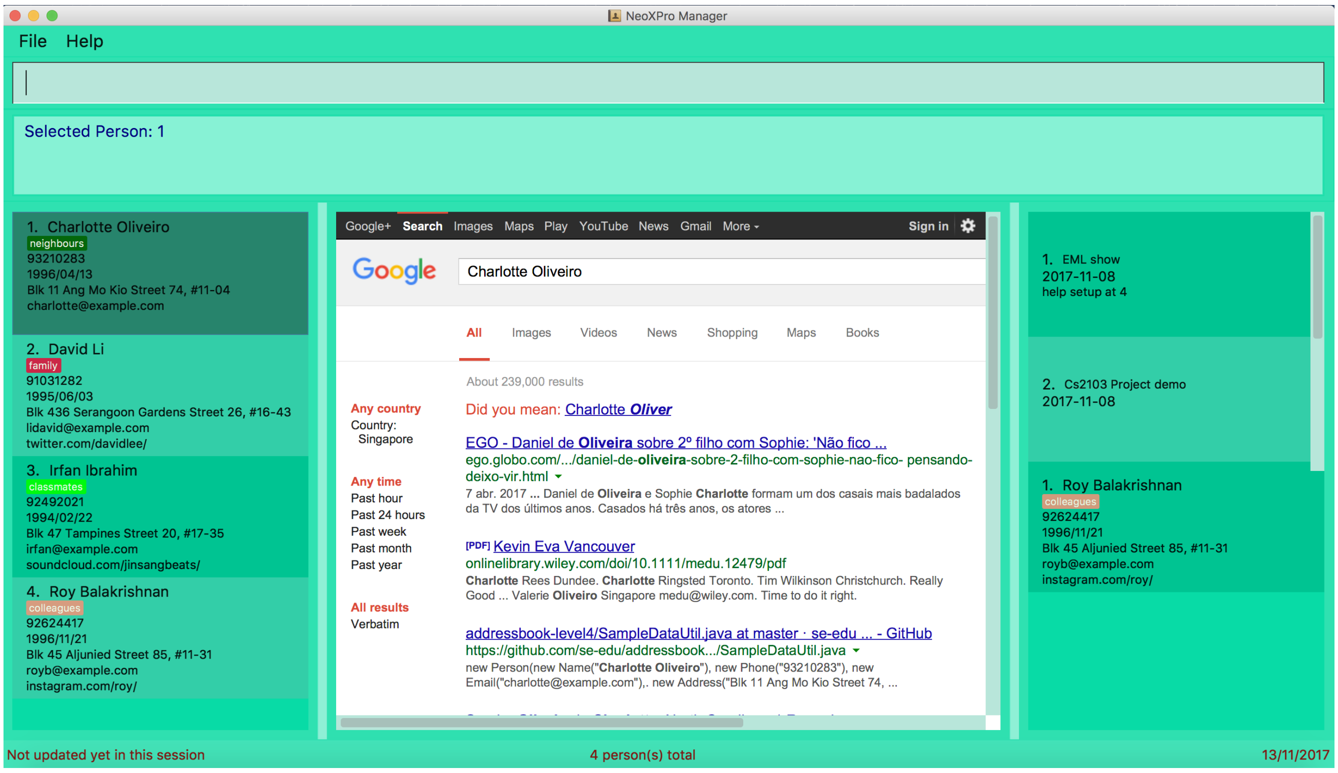
1. EML show
2017-11-08
help setup at 4

2. Cs2103 Project demo
2017-11-08

1. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

Not updated yet in this session
4 person(s) total
13/11/2017

- `find John`
  `select 1`
  Selects the 1st person in the results of the `find` command.
  The 1st person in the list does not have the profile page in address book.



Selected Person: 1

1. Charlotte Oliveiro
neighbours
93210283
1996/04/13
Blk 11 Ang Mo Kio Street 74, #11-04
charlotte@example.com

2. David Li
family
91031282
1995/06/03
Blk 436 Serangoon Gardens Street 26, #16-43
lidavid@example.com
twitter.com/davidlee/

3. Irfan Ibrahim
classmates
92492021
1994/02/22
Blk 47 Tampines Street 20, #17-35
irfan@example.com
soundcloud.com/jinsangbeats/

4. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

Charlotte Oliveiro

About 239,000 results

Did you mean: Charlotte ***Oliver***

EGO - Daniel de **Oliveira** sobre 2º filho com Sophie: 'Não fico ...
ego.globo.com/.../daniel-de-**oliveira**-sobre-2-filho-com-sophie-nao-fico- pensando-deixo-vir.html
7 abr. 2017 **...** Daniel de **Oliveira** e Sophie **Charlotte** formam um dos casais mais badalados da TV dos últimos anos. Casados há três anos, os atores ...

[PDF] Kevin Eva Vancouver
onlinelibrary.wiley.com/doi/10.1111/medu.12479/pdf
Charlotte Rees Dundee. **Charlotte** Ringsted Toronto. Tim Wilkinson Christchurch. Really Good ... Valerie **Oliveiro** Singapore medu@wiley.com. Time to do it right.

addressbook-level4/SampleDataUtil.java at master · se-edu ... - GitHub
https://github.com/se-edu/addressbook.../SampleDataUtil.java
new Person(new Name("**Charlotte Oliveiro**"), new Phone("93210283"), new Email("charlotte@example.com"),. new Address("Blk 11 Ang Mo Kio Street 74, ...

1. EML show
2017-11-08
help setup at 4

2. Cs2103 Project demo
2017-11-08

1. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

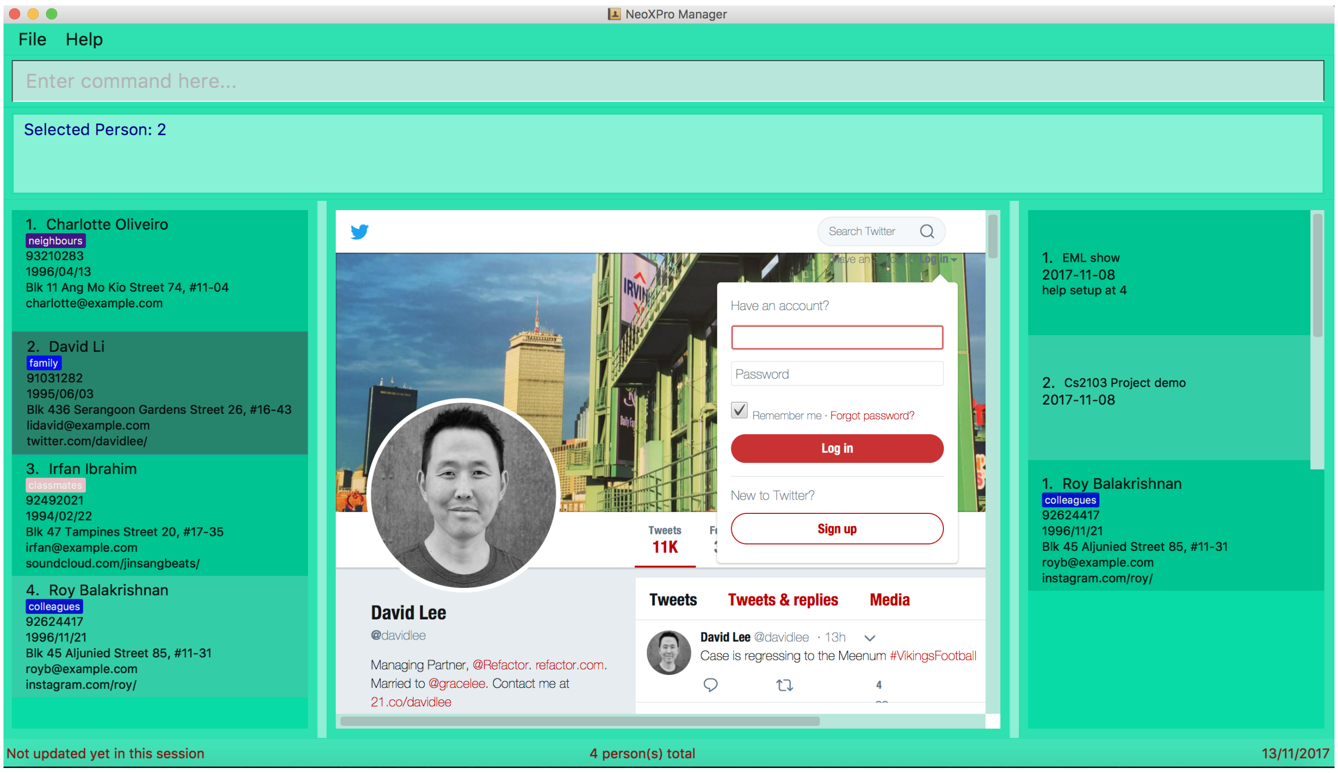
Not updated yet in this session
4 person(s) total
13/11/2017

Selects the person identified by the index number used in the last person listing.
Format: `select INDEX`

> - Selects the person at the specified `INDEX`.
> - If the person has a valid profile page, address book loads that profile page. Otherwise, it loads the Google search page of the person.
> - The index refers to the index number shown in the most recent listing.
> - The index **must be a positive integer** `1, 2, 3, …`

Examples:

- `list`
  `select 2`
  Selects the 2nd person in the address book.
  The 2nd person in the list has the profile page "twitter.com/davidlee"



Selected Person: 2

1. Charlotte Oliveiro
neighbours
93210283
1996/04/13
Blk 11 Ang Mo Kio Street 74, #11-04
charlotte@example.com

2. David Li
family
91031282
1995/06/03
Blk 436 Serangoon Gardens Street 26, #16-43
lidavid@example.com
twitter.com/davidlee/

3. Irfan Ibrahim
classmates
92492021
1994/02/22
Blk 47 Tampines Street 20, #17-35
irfan@example.com
soundcloud.com/jinsangbeats/

4. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

David Lee
@davidlee
Managing Partner, @Refactor. refactor.com. Married to @gracelee. Contact me at 21.co/davidlee

David Lee @davidlee · 13h
Case is regressing to the Meenum #VikingsFootball

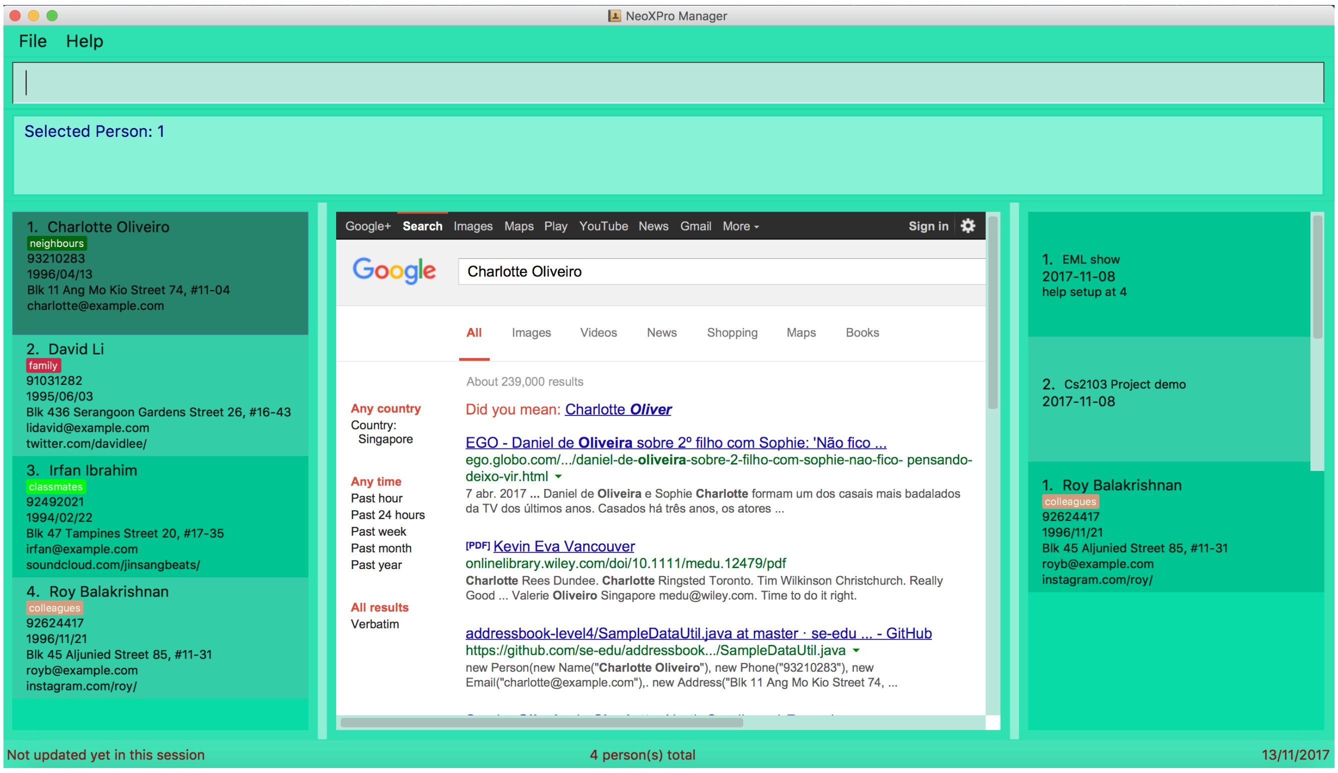
1. EML show
2017-11-08
help setup at 4

2. Cs2103 Project demo
2017-11-08

1. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

Not updated yet in this session
4 person(s) total
13/11/2017

- `find John`
  `select 1`
  Selects the 1st person in the results of the `find` command.
  The 1st person in the list does not have the profile page in address book.



Selected Person: 1

1. Charlotte Oliveiro
neighbours
93210283
1996/04/13
Blk 11 Ang Mo Kio Street 74, #11-04
charlotte@example.com

2. David Li
family
91031282
1995/06/03
Blk 436 Serangoon Gardens Street 26, #16-43
lidavid@example.com
twitter.com/davidlee/

3. Irfan Ibrahim
classmates
92492021
1994/02/22
Blk 47 Tampines Street 20, #17-35
irfan@example.com
soundcloud.com/jinsangbeats/

4. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

About 239,000 results

Did you mean: Charlotte *Oliver*

EGO - Daniel de Oliveira sobre 2º filho com Sophie: 'Não fico ...
ego.globo.com/.../daniel-de-oliveira-sobre-2-filho-com-sophie-nao-fico- pensando-deixo-vir.html
7 abr. 2017 ... Daniel de Oliveira e Sophie Charlotte formam um dos casais mais badalados da TV dos últimos anos. Casados há três anos, os atores ...

[PDF] Kevin Eva Vancouver
onlinelibrary.wiley.com/doi/10.1111/medu.12479/pdf
Charlotte Rees Dundee. Charlotte Ringsted Toronto. Tim Wilkinson Christchurch. Really Good ... Valerie Oliveiro Singapore medu@wiley.com. Time to do it right.

addressbook-level4/SampleDataUtil.java at master · se-edu ... - GitHub
https://github.com/se-edu/addressbook.../SampleDataUtil.java
new Person(new Name("Charlotte Oliveiro"), new Phone("93210283"), new Email("charlotte@example.com"),. new Address("Blk 11 Ang Mo Kio Street 74, ...

1. EML show
2017-11-08
help setup at 4

2. Cs2103 Project demo
2017-11-08

1. Roy Balakrishnan
colleagues
92624417
1996/11/21
Blk 45 Aljunied Street 85, #11-31
royb@example.com
instagram.com/roy/

Not updated yet in this session
4 person(s) total
13/11/2017

End of Extract

Justification

Let users easily access social media and improve experience. Employers can also make use of this feature to quickly access candidates' CVs or profile pages.

Implementation

Start of Extract [from: Developer Guide]

Add Profile Page

The profile page parameter is facilitated by the class `ProfilePage`.
In order to add this parameter profile page, we make modifications to UI, Logic, Model and Storage components.

- `UI Component:`
  `PersonCard`, which resides in UI component, frist creates a label for profile page parameter:

```
@FXML
    private Label profile;
```

Then `PersonCard` binds the label with the value of `ProfilePage` object. If the input value for `ProfilePage` object is null, it make the label dissapear from UI by setting its visibility to FALSE:

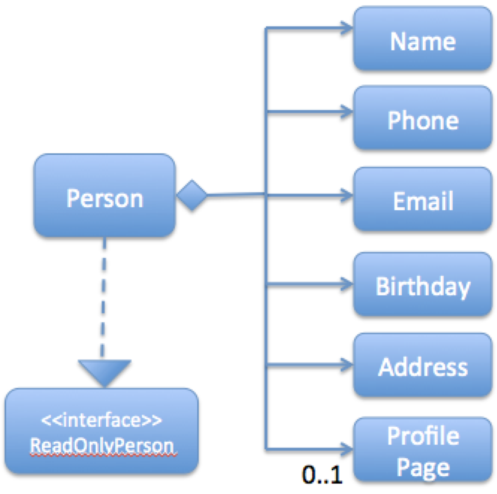
```
private void bindListeners(ReadOnlyPerson person) {
    //... binding other labels ...

    if (!person.profilepageProperty().toString().equals("")) {
        profile.textProperty().bind(Bindings.convert(person.profilepageProperty()));
        profile.setVisible(true);
    } else {
        profile.setVisible(false);
    }
    //... binding other labels ...
}
```

In order to make `select` command load a person’s `ProfilePage` object if it exists, we modify method `handlePersonPanelSelectionChangedEvent()` in `BrowserPanel` that is in charge of updating a person panel:

```
 private void handlePersonPanelSelectionChangedEvent(PersonPanelSelectionChangedEvent
event) {
     //...
     if (person.getProfilePage().hasProfilePage()) {
         loadProfilePage(person);
     } else {
         loadPersonPage(person);
     }
 }
```

- `Logic Component:`
  The prefix for profile page `pr/` is added to `CliSyntax`. Then the following files `AddCommand`, `EditCommand`, `AddCommandParser` and `EditCommandParser` are modified so that `add` and `edit` commands accept the new parameter profile page.
  `AddCommandParser` does not check for profile page prefix `pr/` as this is an optional parameter.
- `Model Component:`
  Class `ProfilePage` is used to store profile page property of class `Person`. Class `Person`, which resides in model component and implements `ReadOnlyPerson` interface, form a composition association with `ProfilePage` .
  As profile page is an optional parameter, a `Person` can be linked to 0 or 1 `ProfilePage` object.
  The relationship is illustrated in the following diagram:



- `Storage Component:`
  `xmlAdaptedPerson` file is used to save information of a person in xml format.
  The `required` parameter of `@XmlElement` element which stores profile page information is set to false to make this property optional:

```
@XmlElement(required = false)
    private String profile = "";
```

In order to make `select` command load a person's `ProfilePage` object if it exists, we modify method `handlePersonPanelSelectionChangedEvent()` in `BrowserPanel` that is in charge of updating a person panel:

```
 private void handlePersonPanelSelectionChangedEvent(PersonPanelSelectionChangedEvent
event) {
     //...
     if (person.getProfilePage().hasProfilePage()) {
         loadProfilePage(person);
     } else {
         loadPersonPage(person);
     }
 }
```

End of Extract

Other contributions

- Fix bugs due to merge conflicts for team (Pull requests #70, #45)
- Write additional tests to increase coverage by 1.4% (Pull requests #89, #99)