SE 211
Software Specification and Design II

Address Book Example

About The Example

- Case study developed by
  Prof. Russell C. Bjork, Gordon College
- http://www.cs.gordon.edu/courses/cs211/AddressBookExample/

Requirements

- Create a program that can be used to maintain an address book (that is, a collection of entries, each containing a person's *first name*, *last name*, *address*, *city*, *state*, *zip*, and *phone number*).

- Features:
  - **Add** a new person to an address book.
  - **Edit** existing information about a person (except the person's name).
  - **Delete** a person.
  - **Sort** the entries in the address book by last name (with ties broken by first name), or by ZIP code (with ties broken by name).
  - **Print out** all the entries in the address book in "mailing label" format.

Requirements (cont'd)

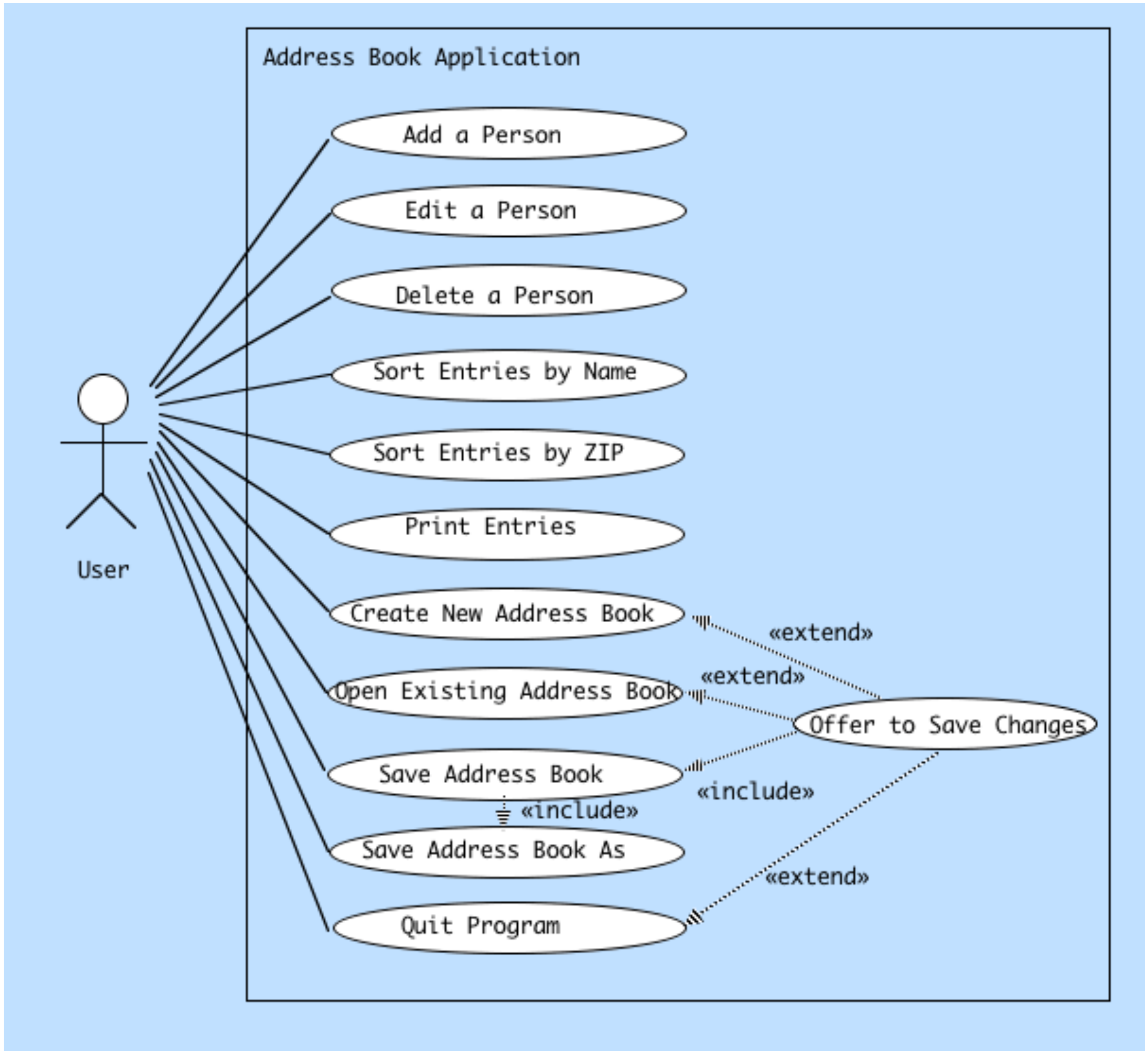
- The GUI must contain the following **File** menu options:
  - **Create** a new address book
  - **Open** a file containing an existing address book
  - **Close** an address book
  - **Save** an address book
  - **Save As...** an address book
  - **Quit** to close open address books and terminate the program.

- Initially, the program should be able to work with a single address book at a time. As a result, if the user chooses the **New** or **Open** menu option, any current address book will be closed before creating/opening a new one.

Requirements (cont'd)

- The program will keep track of whether any changes have been made to an address book since it was last saved, and will offer the user the opportunity to save changes when an address book is closed either explicitly or as a result of choosing to create/open another or to quit the program.
- The program will keep track of the file that the current address book was read from, or most recently saved to. It will display the file's name as the title of the main window, and will use that file when executing the **Save** option.
- When a **New** address book is initially created, its window will be titled "Untitled", and a **Save** operation will be converted to **Save As** ... (that is, the user will be required to specify a file).

Use Cases


Address Book Application
User
Add a Person
Edit a Person
Delete a Person
Sort Entries by Name
Sort Entries by ZIP
Print Entries
Create New Address Book
Open Existing Address Book
Save Address Book
Save Address Book As
Quit Program
Offer to Save Changes
«extend»
«extend»
«include»
«include»
«extend»

Use Case: Add a Person

- The Add a Person use case is initiated when the user clicks the "**Add**" button in the main window.
- A dialog box appears, with title "New Person", containing fields for the user to fill in the new person's first and last names and other information.
- The box can be dismissed by clicking either "OK" or "Cancel".
- If the "OK" button is clicked, a new person is added to the end of the address book, and the person's name is added to the end of the list of names in the main window.
- If the "Cancel" button is clicked, no changes are made either to the address book or to the main window.

Use Case: Edit a Person

- The Edit a Person use case is initiated when the user either highlights a name in the list of names in the main window and then clicks the "**Edit**" button, or the user double-clicks a name.
- In either case, a dialog box, with title "Edit person's name", appears containing current information about the person selected, (except the person's name, which appears only in the title).
- The user can then edit the individual fields.
- The box can be dismissed by clicking either "OK" or "Cancel".
- If the "OK" button is clicked, the entry in the address book for the selected person is updated to reflect any changes made by the user.
- If the "Cancel" button is clicked, no changes are made to the address book.

Use Case: Create New Address Book

- The Create a New Address Book use case is initiated when the user chooses "**New**" from the File menu.
- If the current address book contents have been changed since the last successful **New**, **Open**, **Save**, or **Save As** ... operation was done, the Offer to Save Changes extension is executed.
- Unless the user cancels the operation, a new empty address book is then created and replaces the current address book. This results in the list of names in the main window being cleared, the current file becoming undefined, and the title of the main window becomes "Untitled". (NOTE: These conditions will also be in effect when the program initially starts up.)

Use Case: Open Existing Address Book

- The Open Existing Address Book use case is initiated when the user chooses "**Open**" from the File menu.
- If the current address book contents have been changed since the last successful **New**, **Open**, **Save**, or **Save As** ... operation was done, the Offer to Save Changes extension is executed.
- Unless the user cancels the operation, a load file dialog is displayed and the user is allowed to choose a file to open.

Use Case: Open Existing Address Book (cont'd)

- Once the user chooses a file, the current address book is replaced by the result of reading in the specified address book. As a result:
  - The file that was opened will become the current file.
  - The names in the address book that was read will replace the list of names in the main window.
  - The name of the file will be displayed as the title of the main window.
- If the user cancels the file dialog, or attempting to read the file results in an error, the current address book is left unchanged. If the cancellation results from an error reading the file, a dialog box is displayed warning the user of the error.

Use Case: Offer to Save Changes Extension

- The Offer to Save Changes extension is initiated from within the Create New Address Book, Open Existing Address Book, or Quit Program use cases, if the current address book has been changed since the last successful **New**, **Open**, **Save**, or **Save As** ... operation was done.
- A dialog box is displayed, informing the user that there are unsaved changes, and asking the user whether to save changes, not save changes, or cancel the operation.
- If the user chooses to save changes, the Save Address Book use case is executed (which may result in executing the Save Address Book As ... use case if there is no current file).
- If the user chooses not to save changes, the original operation is simply resumed.
- If the user chooses to cancel (or cancels the save file dialog if one is needed), the original operation is canceled.

What classes do we need for this program?

Classes

- AddressBook: A single entity object representing the current address book that the program is working with.

Classes

- AddressBook: A single entity object representing the current address book that the program is working with.
- Person: An arbitrary number of entity objects, each representing one of the people that is in the current address book.

Classes

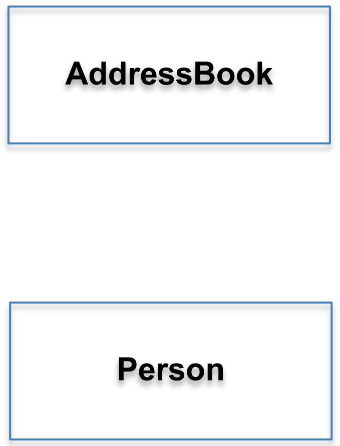
- AddressBook: A single entity object representing the current address book that the program is working with.
- Person: An arbitrary number of entity objects, each representing one of the people that is in the current address book.
- AddressBookGUI: An object representing the interface between the address book system and the human user.

Classes

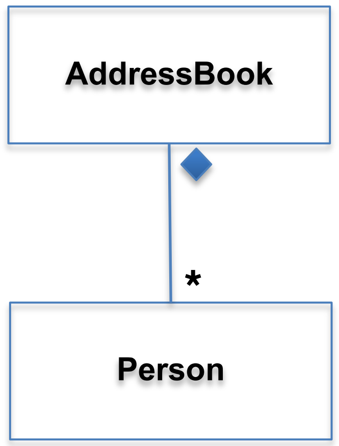
- AddressBook: A single entity object representing the current address book that the program is working with.
- Person: An arbitrary number of entity objects, each representing one of the people that is in the current address book.
- AddressBookGUI: An object representing the interface between the address book system and the human user.
- AddressBookController: An object that carries out the use cases in response to user interaction with the GUI.

What are the relationships among these classes?

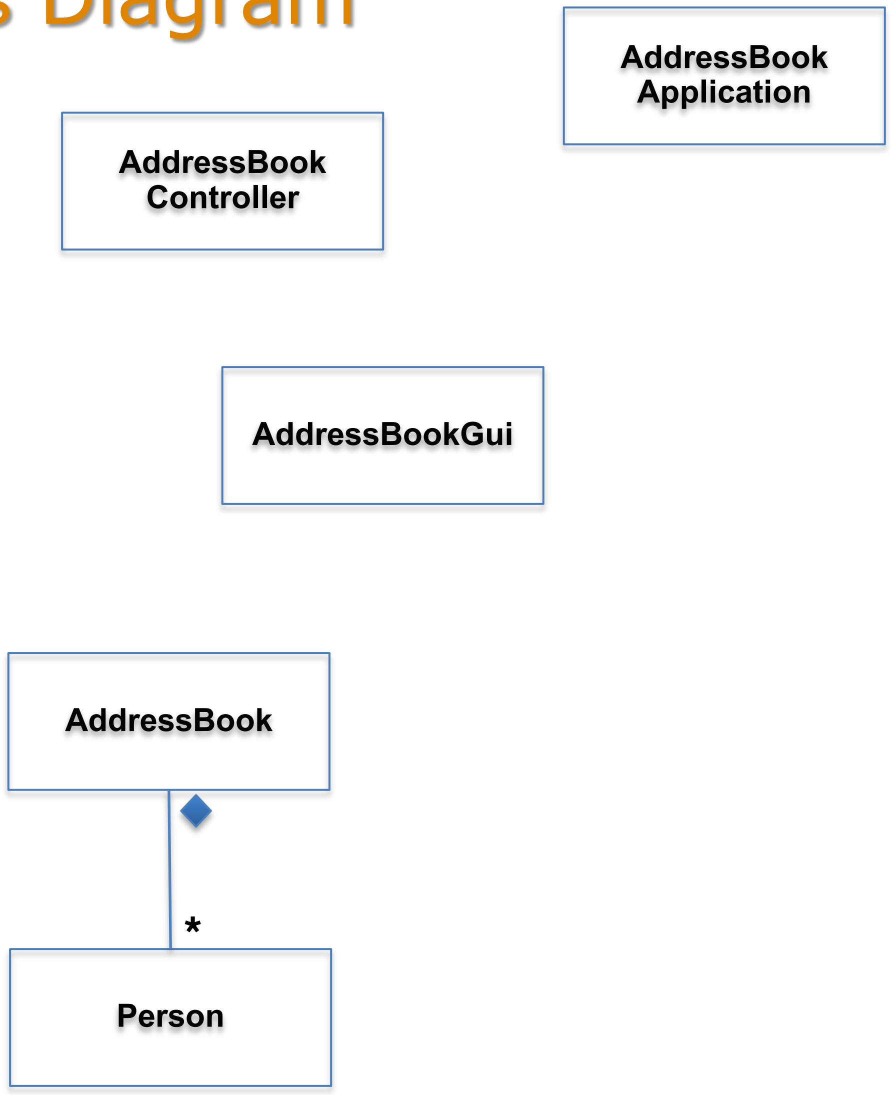
Class Diagram



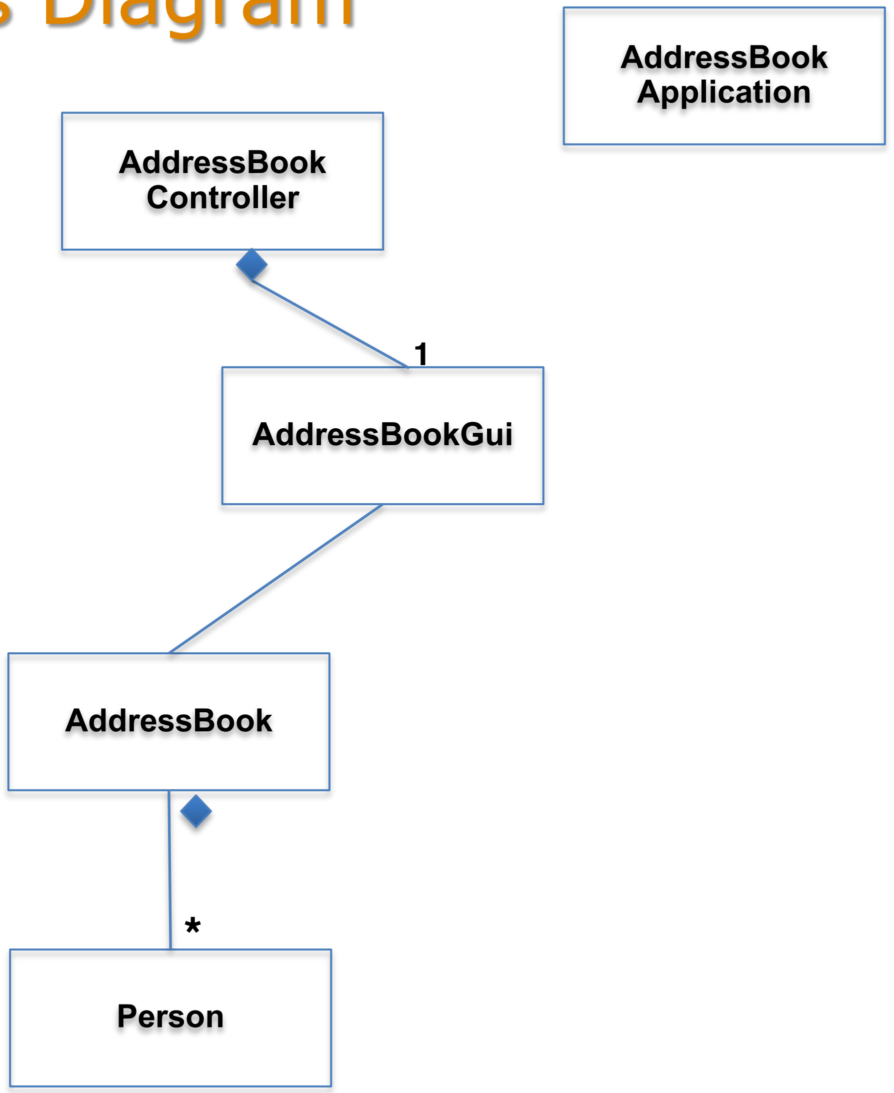
Class Diagram



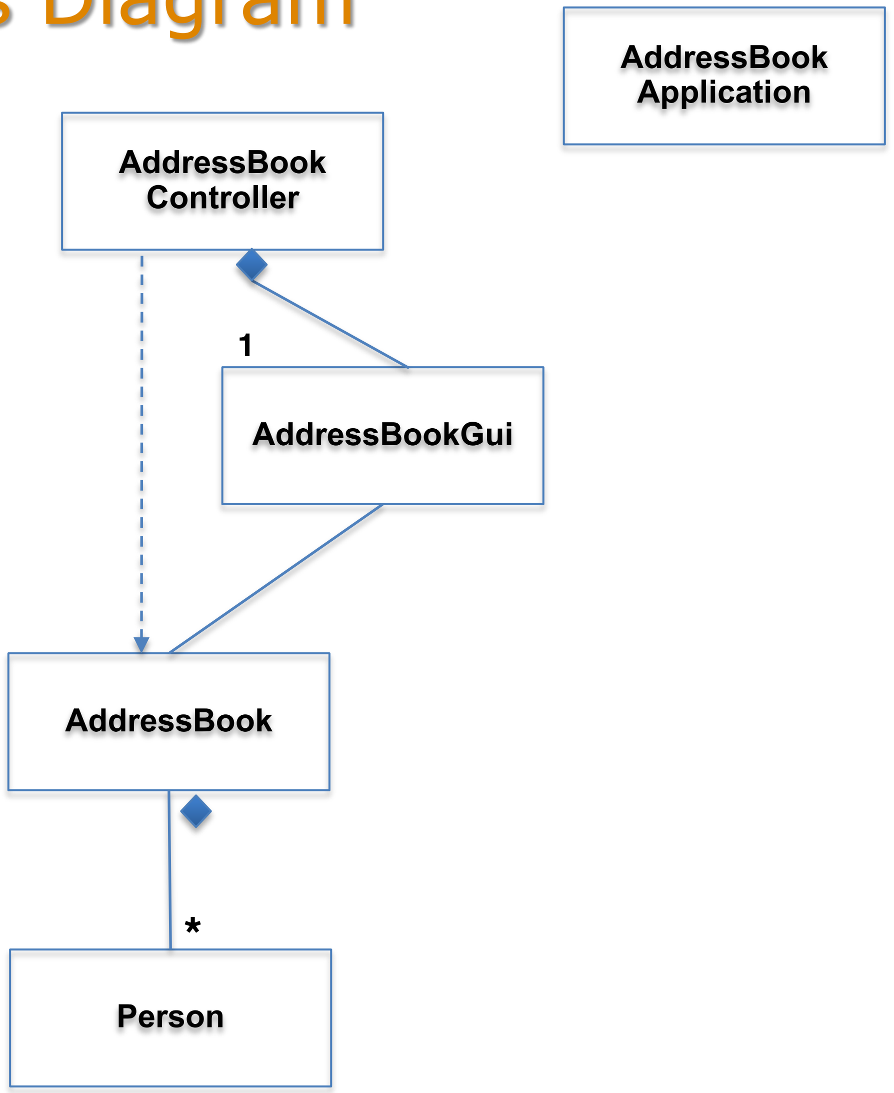
Class Diagram



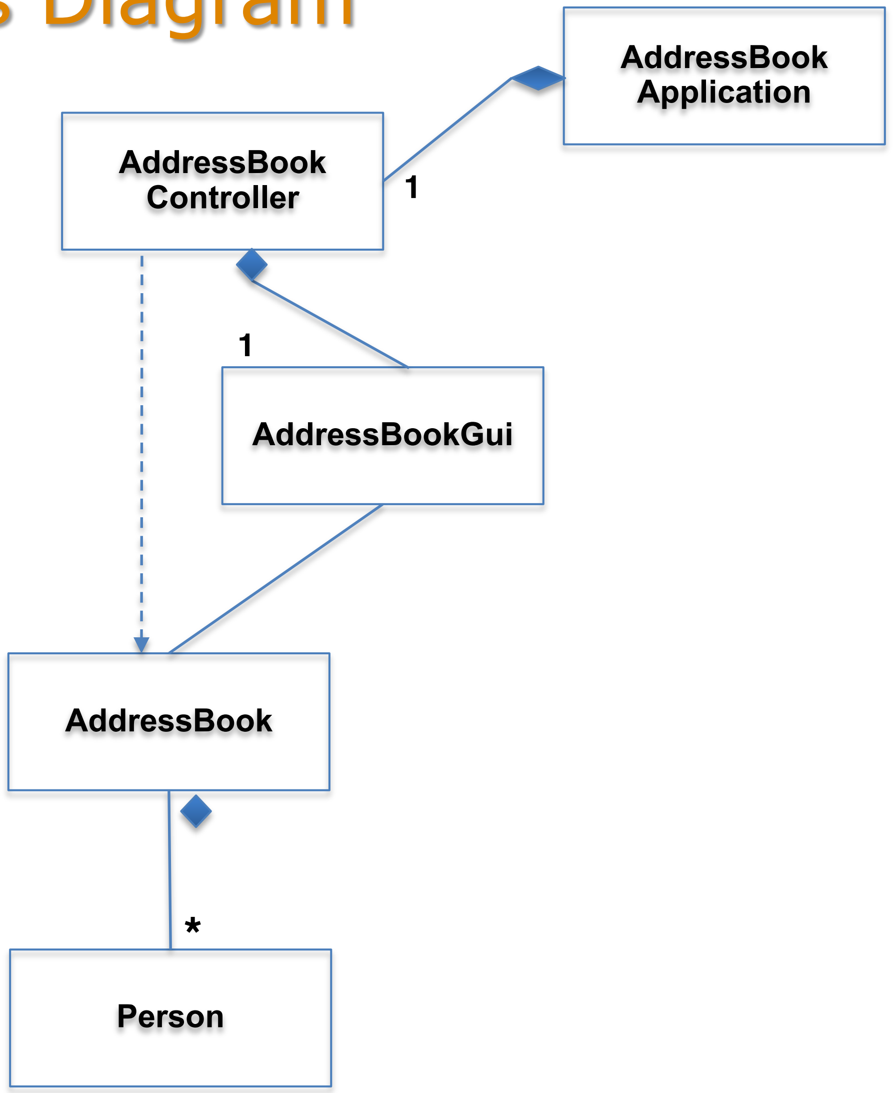
Class Diagram



Class Diagram



Class Diagram



What are the details for each class?

Class: Person

```
- firstName: String
-  lastName: String
-  …

+ Person(String firstName, String lastName, String address,… )
+ getFirstName() : String
+ getLastName() : String
+ getAddress() : String
  …
```

Class: AddressBook

```
- collection: Person[]
-  file: File
-  changedSinceLastSave: boolean
  …

+ AddressBook()
+ addPerson(String firstName, String LastName, …)
+ getFullNameOfPerson(int index) : String
+ updatePerson(int index, String address, …)
+ getFile() : File
+ setFile(File fie)
+ getChangedSinceLastSave() : boolean
+ setChangedSinceLastSave(boolean changedSinceLastSave)
+ printAll()
+ sortByName()
  …
```

Class: AddressBookGui

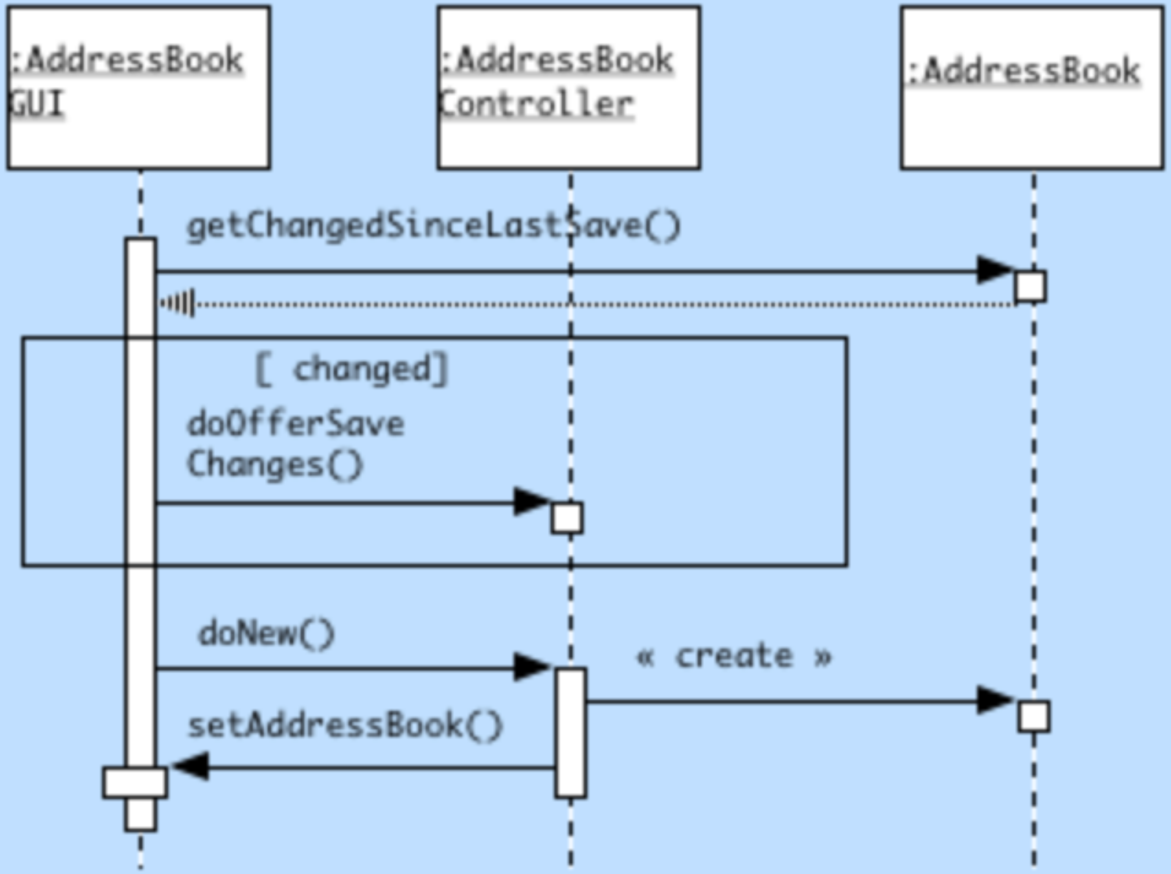
```
- controller: AddressBookController
- addressBook: AddressBook
-  nameList: Jlist
-  addButton: Jbutton
-  editButton: Jbutton
   …

+ AddressBookGui(AddressBookController controller,
                 AddressBook addressBook)
+ getAddressBook() : AddressBook
+ setAddressBook(AddressBook addressBook)
+ reportError(String message)
   …
```

Class: AddressBookController

```
+ AddressBookCotroller (FileSystem filesystem)
+ doNew (AddressBookGui gui)
+ doAdd (AddressBookGui gui)
+ doPrint (AddressBookGui gui)
+ doSortByName (AddressBookGui gui)
  …
```

What about sequence diagrams?

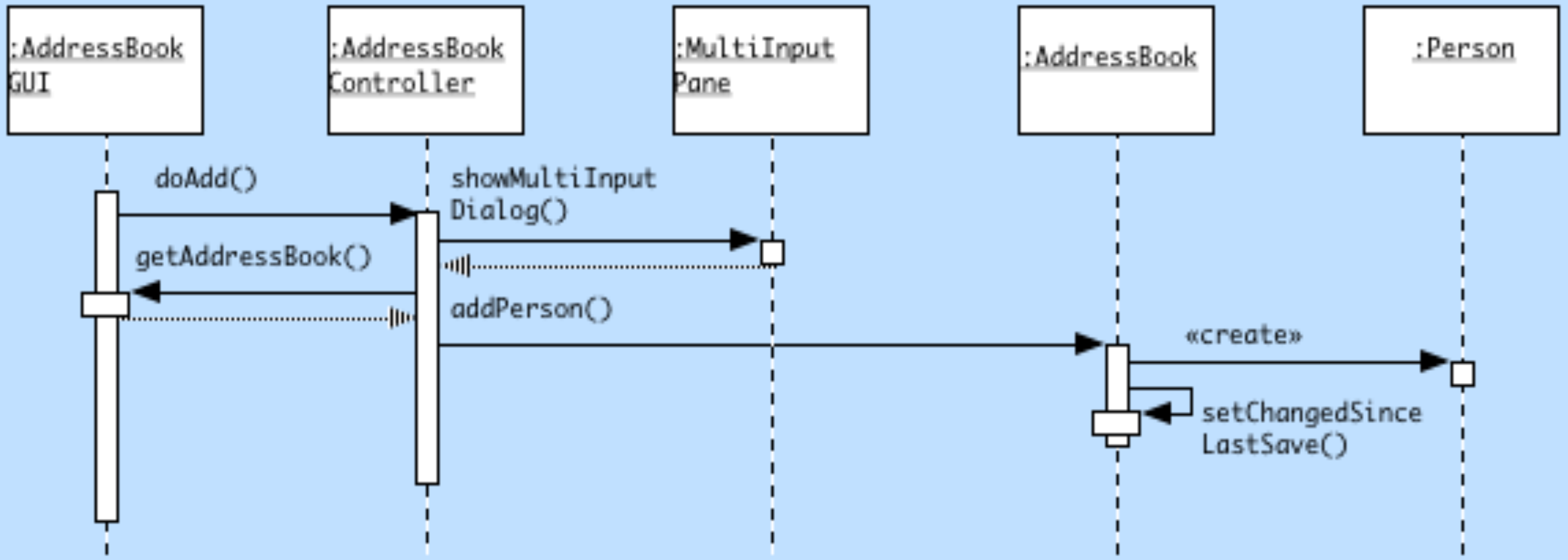
Create New Address Book Use Case Sequence Diagram



Add a Person Use Case Sequence Diagram

???

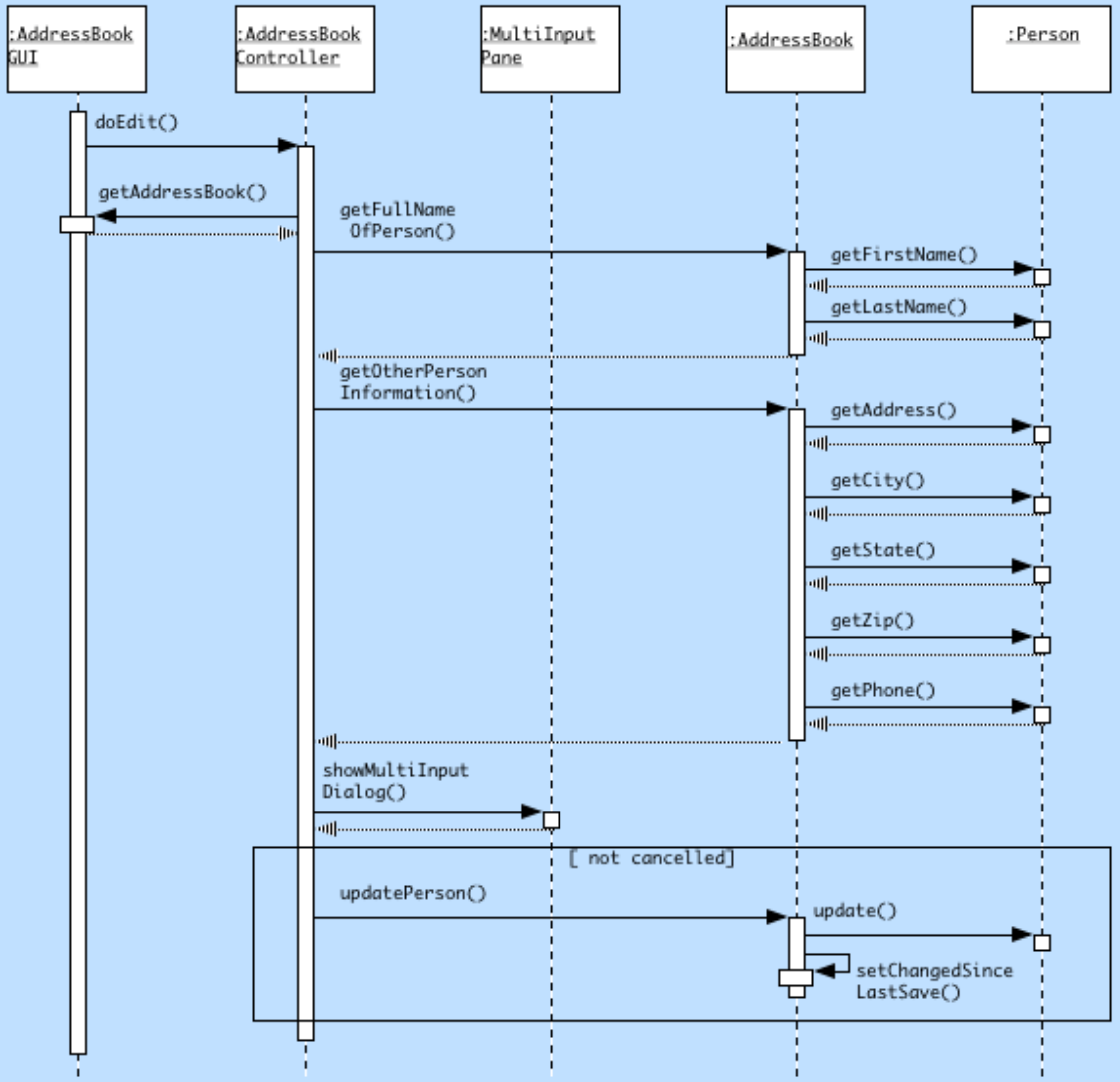
Add a Person Use Case Sequence Diagram



Edit a Person Use Case Sequence Diagram

???

Edit a Person Use Case Sequence Diagram



If there is no selected name, none of the above is done; instead, an error is reported