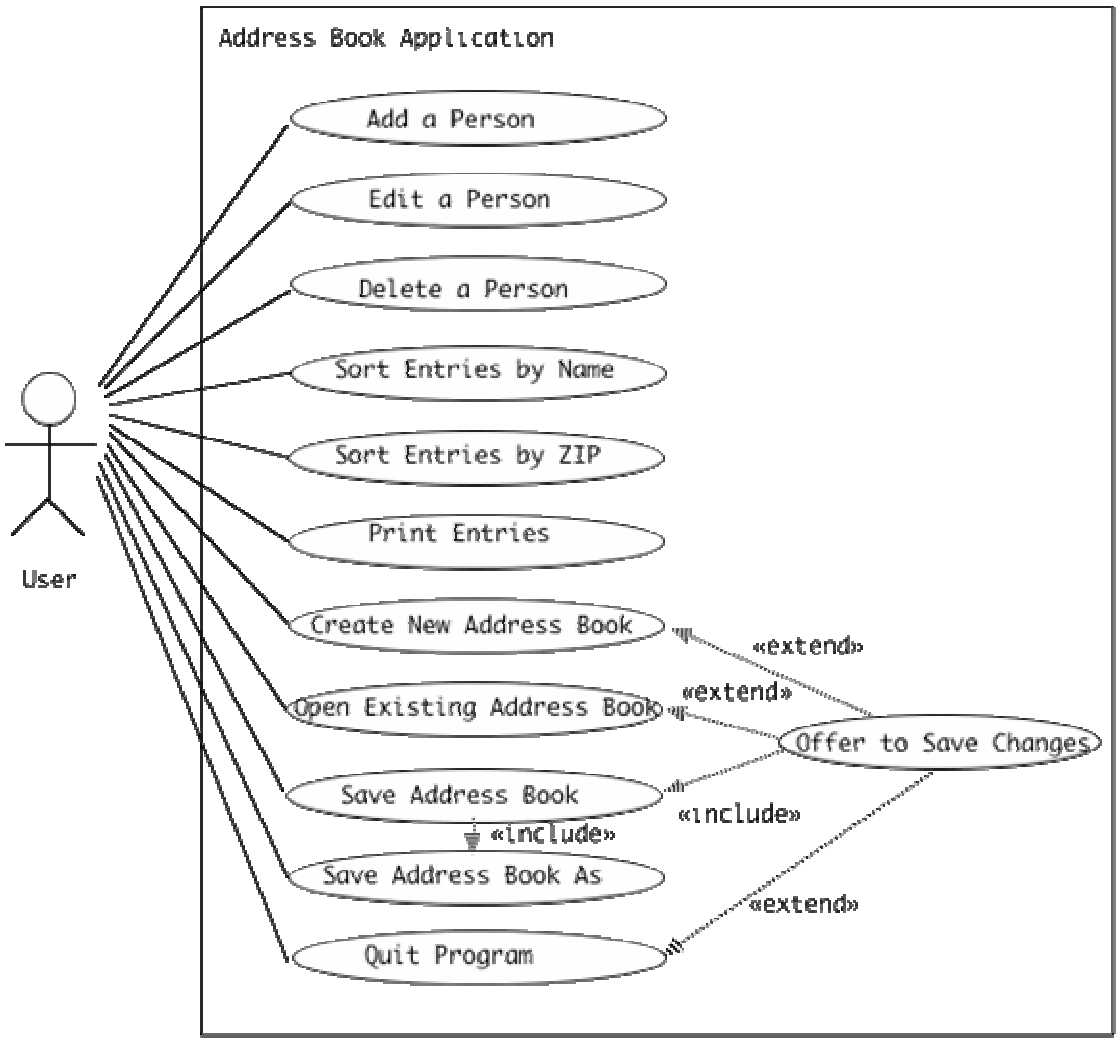
Use Cases for a Simple Address Book

In the following, use cases are listed in the natural order that a user would think of them. In the actual File menu, items that correspond to the various use cases will be listed in the traditional order, which is slightly different.



(Click on a use case above to go to the flow of events for that use case)

Flows of Events for Individual Use Cases

Add a Person Use Case

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.

[Sequence Diagram]

Edit a Person Use Case

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.

[Sequence Diagram]

Delete a Person Use Case

The Delete a Person use case is initiated when the user highlights a name in the list of names in the main window and then clicks the "Delete" button. A dialog box appears, asking the user to confirm deleting this particular individual. 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 deleted, and the person's name is deleted from 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.

[Sequence Diagram]

Sort Entries by Name Use Case

The Sort Entries by Name use case is initiated when the user clicks the Sort by Name button in the main window. The entries in the address book are sorted alphabetically by name, and the list in the main window is updated to reflect this order as well.

[Sequence Diagram]

Sort Entries by ZIP Use Case

The Sort Entries by ZIP use case is initiated when the user clicks the Sort by ZIP button in the main window. The entries in the address book are sorted by zip code, and the list in the main window is updated to reflect this order as well.

[Sequence Diagram]

Print Entries Use Case

The Print Entries use case is initiated when the user chooses "Print" from the File menu. A save file dialog is displayed, and the user is allowed to choose a file to print the labels to. (If the user cancels the file dialog, the Print operation is canceled.) The current contents of the address book are written out to the specified file (in their current order) in "mailing label" format. No information maintained by the program is changed.

[Sequence Diagram]

Create New Address Book Use Case

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.)

[Sequence Diagram]

Open Existing Address Book Use Case

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. Once the user chooses a file, the current address book is replaced by the result of reading in the specified address book. This results in the list of names in the main window being replaced by the names in the address book that was read, the file that was opened becoming the current file, and its name being 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.)

[Sequence Diagram]

Save Address Book Use Case

The Save Address Book use case is initiated when the user chooses "Save" from the File menu. (The Save option is grayed out unless changes have been made to the address book since the last New, Open, Save, or Save As ... operation was done.) If there is a current file, the current address book is saved to this file. (If attempting to write the file results in an error, a dialog box is displayed warning the user of the error.) If there is no current file, the Save Address Book As .. use case is done instead. In all cases, the current address book and window list are left unchanged.

[Sequence Diagram]

Save Address Book As ... Use Case

The Save Address Book As ... use case is initiated when the user chooses "Save As ..." from the File menu. (The Save As ... option is always available.) A save file dialog is displayed and the user is allowed to choose the name of a file in which to save the address book. (If the user cancels the file dialog, the Save As ... operation is canceled.) The current address book is saved to the specified file, and the file to which it was saved becomes the current file and its name is displayed as the title of the main window. (If attempting to write the file results in an error, a dialog box is displayed warning the user of the error, and the current file and main window title are unchanged.) In all cases, the current address book and window list are left unchanged.

[Sequence Diagram]

Quit Program Use Case

The Quit Program use case is initiated when the user chooses "Quit" from the File menu, or clicks the close box for the main window. In either case, if the current address book contents have been changed since the last New, Open, Save, or Save As ... operation was done, the Offer to Save Changes extension is executed. Unless the user cancels the operation, the program is terminated.

[Sequence Diagram]

Offer to Save Changes Extension

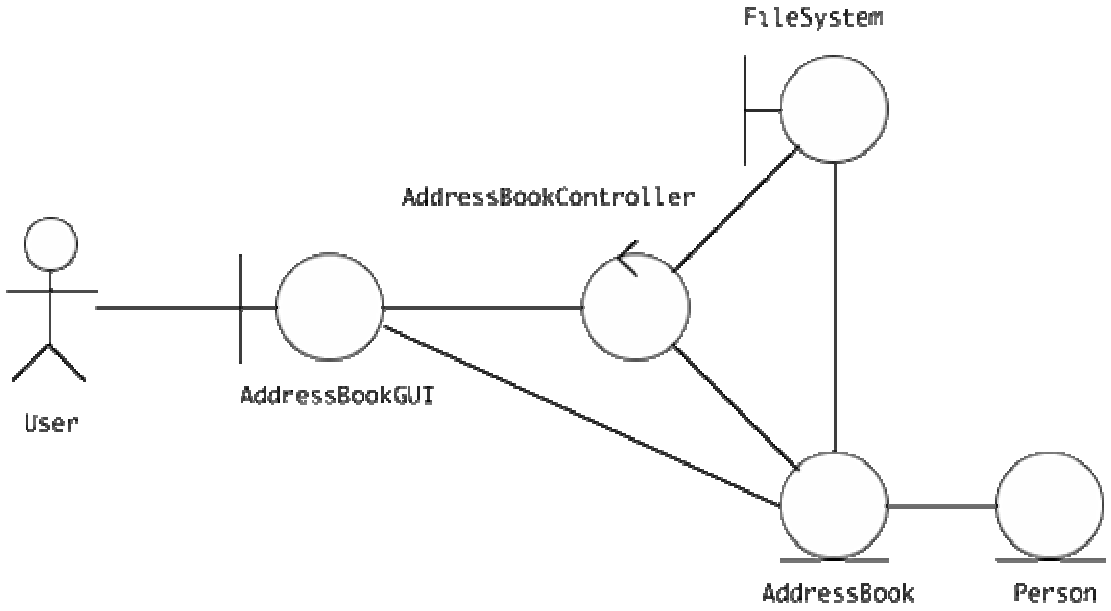
The Offer to Save Changes extension is initiated from within the Create New Address Book, Open Existing Address Book, or Quit program use cases, if the current address book has been changed since the last successful New, Open, Save, or Save As ... operation was done. A dialog box is displayed, informing 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.

[Sequence Diagram]

Analysis

An initial reading of the use cases suggests that the following will be part of the system.

- A single entity object representing the current address book that the program is working with (`AddressBook`).
- An arbitrary number of entity objects, each representing one of the people that is in the current address book (`Person`).
- A boundary object representing the interface between the address book system and the human user (`AddressBookGUI`).
- A boundary object representing the interface between the address book system and the file system on disk (`FileSystem`).
- A controller object that carries out the use cases in response to user gestures on the GUI (`AddressBookController`). (For a problem of this small size, a single controller is sufficient.)



The various use cases work with these objects, as follows:

- The Add a Person Use Case involves getting the new information from the user, and then telling the AddressBook object to add a new person with this information to its collection
- The Edit a Person Use Case involves displaying the current information about the desired person (obtained from the AddressBook), then allowing the user to enter new information for the various fields, then telling the AddressBook object to make the changes.
- The Delete a Person Use Case involves asking the user to confirm deletion, and then telling the AddressBook object to remove this person from its collection.
- The Sort Entries by Name Use Case involves telling the AddressBook object to rearrange its collection in order of name.
- The Sort Entries by ZIP Use Case involves telling the AddressBook object to rearrange its collection in order of ZIP.
- The Create New Address Book Use Case involves creating a new AddressBook object.
- The Open Existing Address Book Use Case involves getting a file specification from the user, and then telling the FileSystem object to read in an AddressBook object from this file.
- The Save Address Book Use Case involves determining whether or not the current AddressBook object has a file it was last read from / saved to; if so, telling the FileSystem object to save the current AddressBook object to this file. (If not, the Save Address Book As ... Use Case is done instead.)
- The Save Address Book As ... Use Case involves getting a file specification from the user, and then telling the FileSystem object to save the current AddressBook object to this file.

- The Print Address Book Use Case involves telling the AddressBook object to print out its collection in order.
- (The Quit Program Use Case does not involve any of the other objects)
- (The Offer to Save Changes Extension may involve performing the Save Address Book Use Case.)

CRC Cards for the Address Book Example

Responsibilities are assigned to the various classes based on the use of the model-view-controller design pattern. The two entity classes (AddressBook and Person) serve as the model. The GUI class (AddressBookGUI) serves as the view. The controller class (AddressBookController) serves, of course, as the controller.

The view (AddressBookGUI) needs to be made an observer of the model (specifically, AddressBook) so that it always reflects the current state of the model - specifically, the list of names, the title, and its saved/needs to be saved status.

Using CRC cards to assign responsibilities to various classes for the tasks required by the various use cases leads to the creation of the following cards.

- Class AddressBook
- Class AddressBookController
- Class AddressBookGUI
- Class FileSystem
- Class Person

Class AddressBook

The CRC Cards for class AddressBook are left as an exercise to the student

[Links for this class]

Class AddressBookController

The basic responsibility of an AddressBookController object is to carry out the various use cases.

Responsibilities	Collaborators
Allow the user to perform the Add a Person Use Case	AddressBook
Allow the user to perform the Edit a Person Use Case	AddressBook
Allow the user to perform the Delete a Person Use Case	AddressBook
Allow the user to perform the Sort Entries by Name Use Case	AddressBook
Allow the user to perform the Sort Entries by ZIP Use Case	AddressBook
Allow the user to perform the Create New Address Book Use Case	AddressBook
Allow the user to perform the Open Existing Address Book Use Case	FileSystem
Allow the user to perform the Save Address Book Use Case	AddressBook FileSystem
Allow the user to perform the Save Address Book As ... Use Case	FileSystem
Allow the user to perform the Print Entries Use Case	AddressBook
Perform the Offer to Save Changes Extension when needed by another Use Case	AddressBook

[Links for this class]

Class AddressBookGUI

The basic responsibility of a GUI object is to allow interaction between the program and the human user.

Responsibilities	Collaborators
Keep track of the address book object it is displaying	
Display a list of the names of persons in the current address book	AddressBook
Display the title of the current address book - if any	AddressBook
Maintain the state of the "Save" menu option - usable only when the address book has been changed since the last time it was opened / saved.	AddressBook
Allow the user to request the performance of a use case	AddressBookController

[Links for this class]

Class FileSystem

The basic responsibility of a FileSystem object is to manage interaction between the program and the file system of the computer it is running on.

Responsibilities	Collaborators
Read a stored address book from a file, given its file name	AddressBook
Save an address book to a file, given its file name	AddressBook

[Links for this class]

Class Person

The basic responsibility of a Person object is to maintain information about a single individual.

Responsibilities	Collaborators
Create a new object, given an individual's name, address, city, state, ZIP, and phone	
Furnish the individual's first name	
Furnish the individual's last name	
Furnish the individual's address	
Furnish the individual's city	
Furnish the individual's state	
Furnish the individual's ZIP	
Furnish the individual's phone number	
Update the stored information (except the name) about an individual	

[Links for this class]

Sequence Diagrams for the Address Book Example

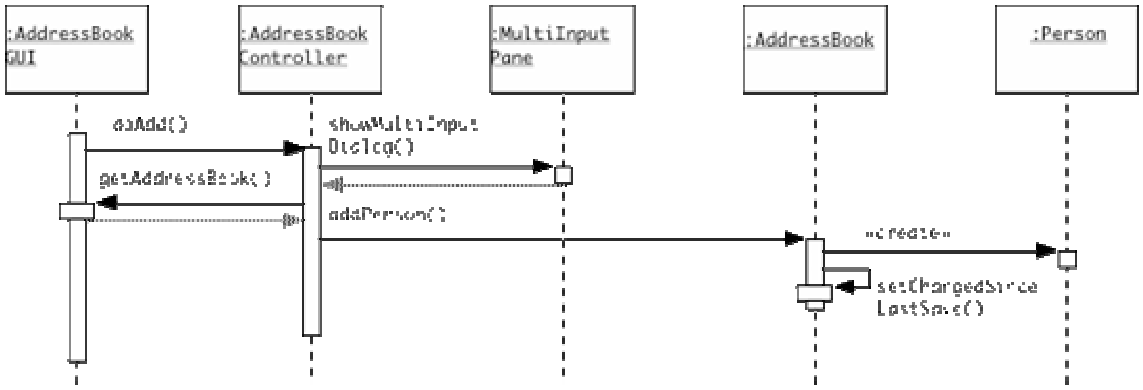
Each of the use cases discovered in the analysis of the system will be realized by a sequence of operations involving the various objects comprising the system.

- Sequence Diagram Realizing the Add a Person Use Case
- Sequence Diagram Realizing the Edit a Person Use Case
- Sequence Diagram Realizing the Delete a Person Use Case
- Sequence Diagram Realizing the Sort Entries by Name Use Case

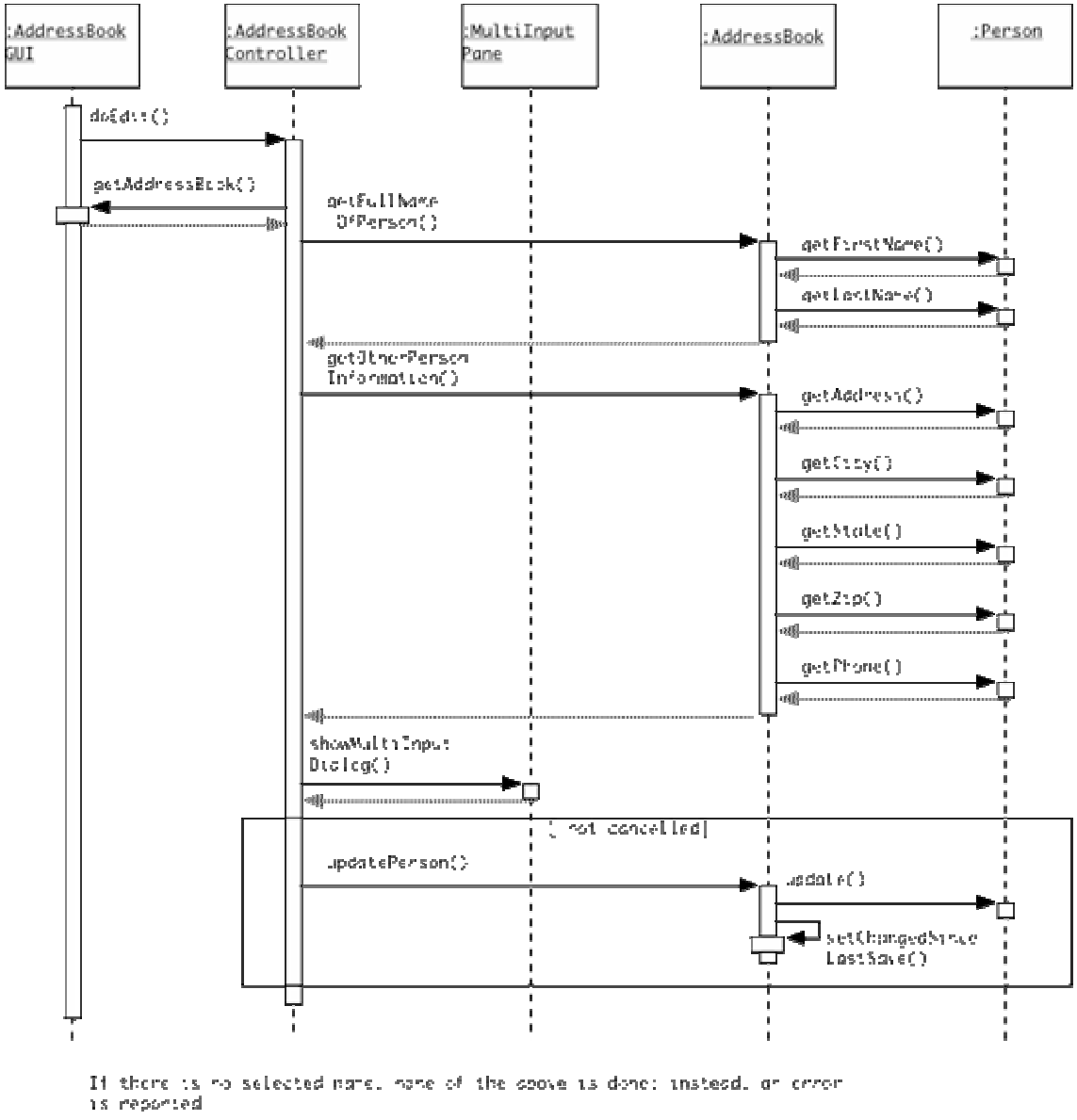
- Sequence Diagram Realizing the Sort Entries by ZIP Use Case
- Sequence Diagram Realizing the Print Entries Use Case
- Sequence Diagram Realizing the Create New Address Book Use Case
- Sequence Diagram Realizing the Open Existing Address Book Use Case
- Sequence Diagram Realizing the Save Address Book Use Case
- Sequence Diagram Realizing the Save Address Book As ... Use Case
- Sequence Diagram Realizing the Offer to Save Changes Extension

---

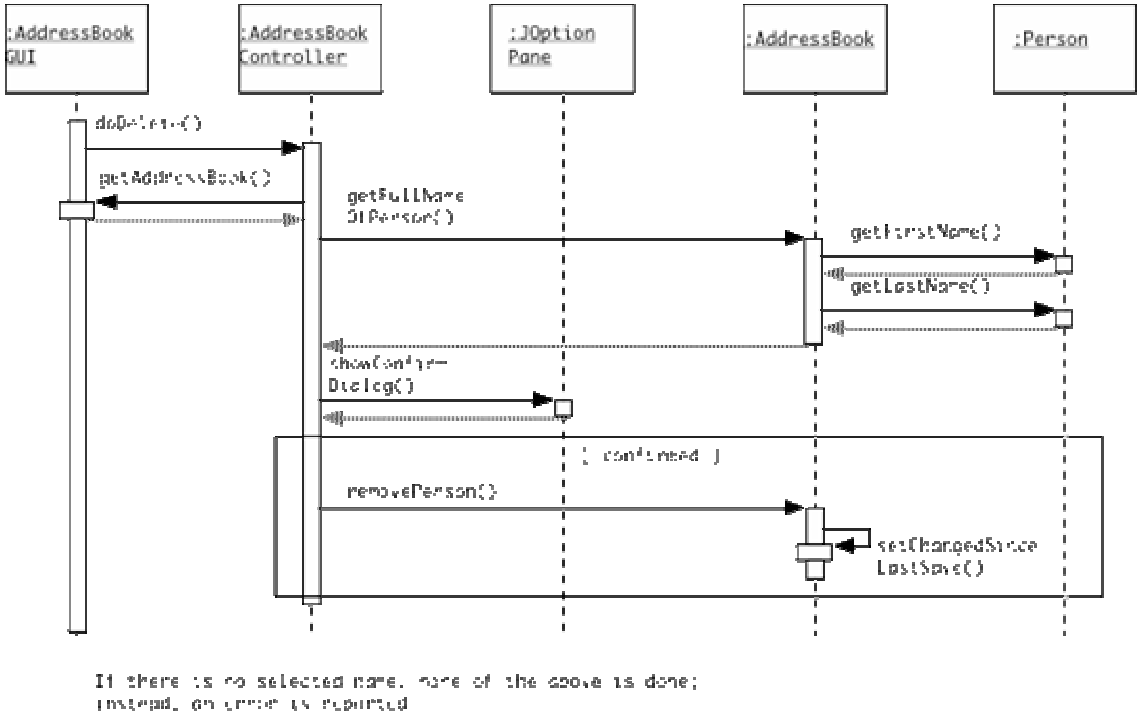
Add a Person Use Case Sequence Diagram



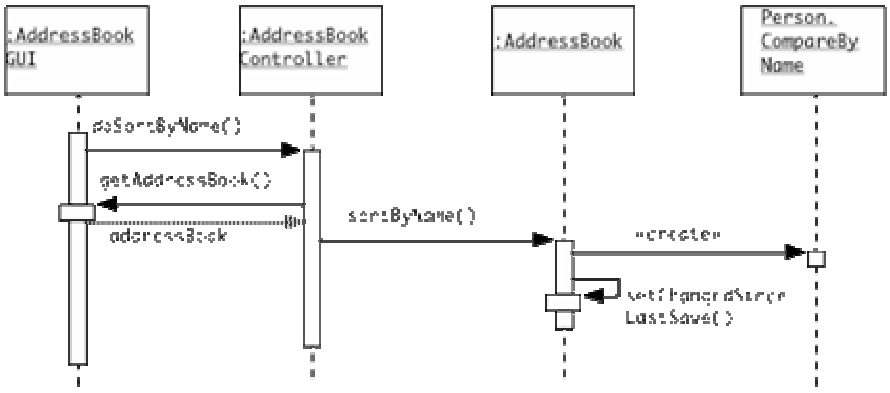
Edit a Person Use Case Sequence Diagram



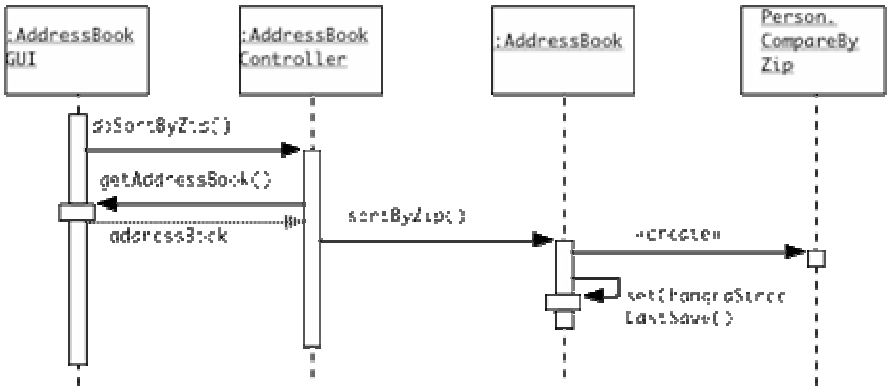
Delete a Person Use Case Sequence Diagram



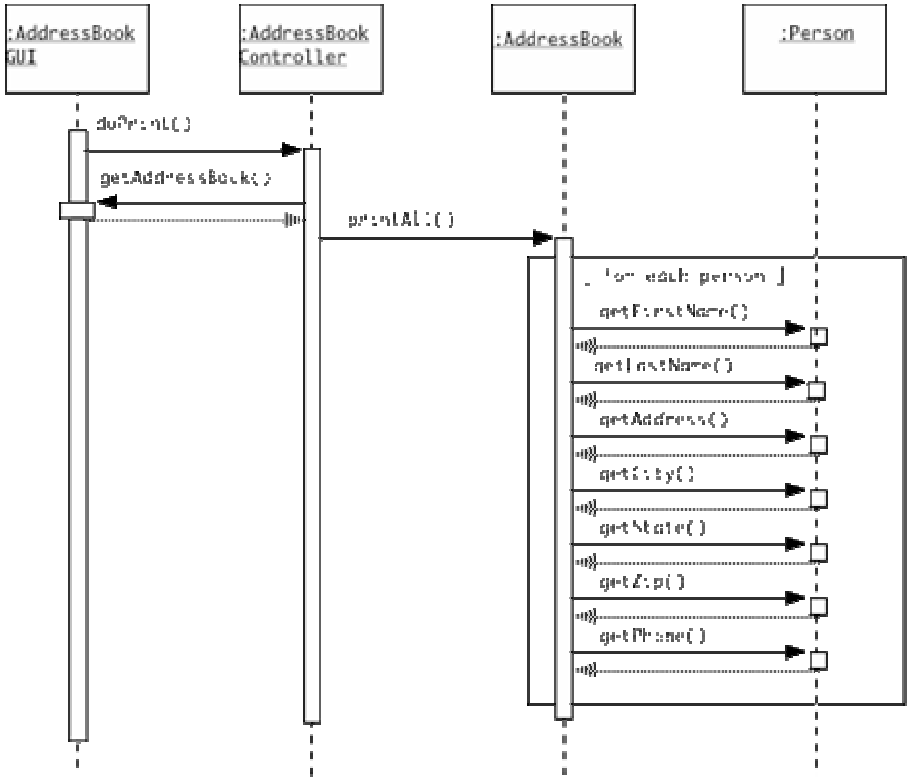
Sort Entries By Name Use Case Sequence Diagram



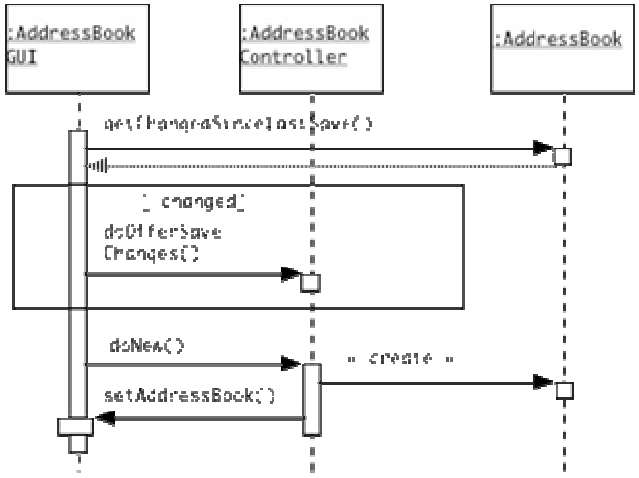
Sort Entries By Zip Use Case Sequence Diagram



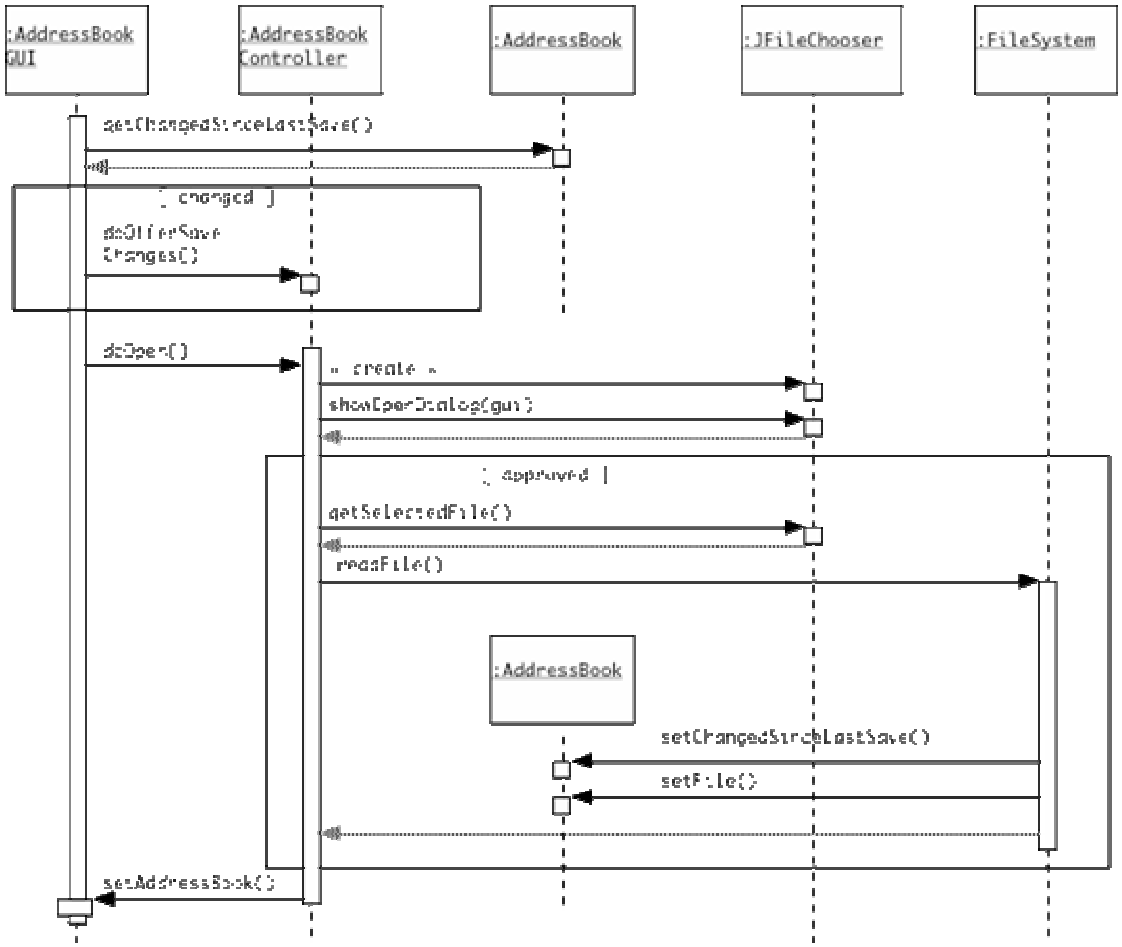
Print Entries Use Case Sequence Diagram



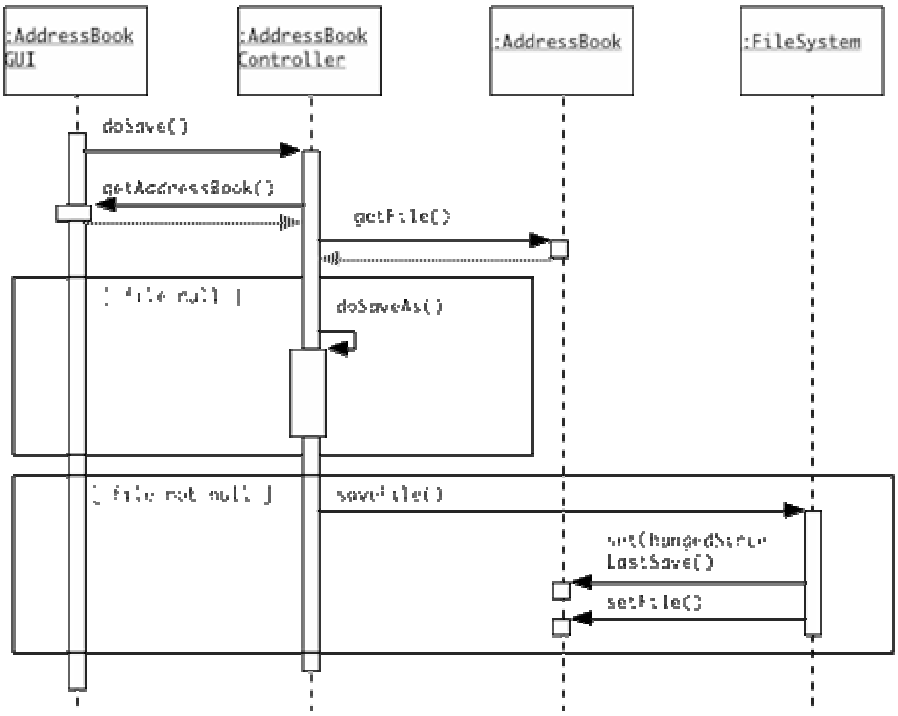
Create New Address Book Use Case Sequence Diagram



Open Existing Address Book Use Case Sequence Diagram



Save Address Book Use Case Sequence Diagram

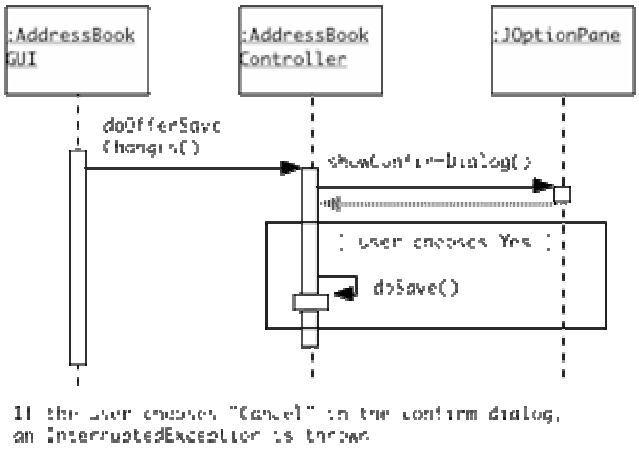


Save Address Book As ... Use Case Sequence Diagram

The Sequence Diagram for the Save Address Book As ... Use Case is left as an exercise to the student

---

Offer To Save Changes Extension Use Case



Class Diagram for the Address Book Example

Shown below is the class diagram for the Address Book Example. To prevent the diagram from becoming overly large, only the name of each class is shown - the attribute and behavior "compartments" are shown in the detailed design, but are omitted here.

The diagram includes the classes discovered during analysis, plus some additional classes discovered during design. (In a more significant system, the total number of classes may be about five times as great as the number of classes uncovered during analysis.)

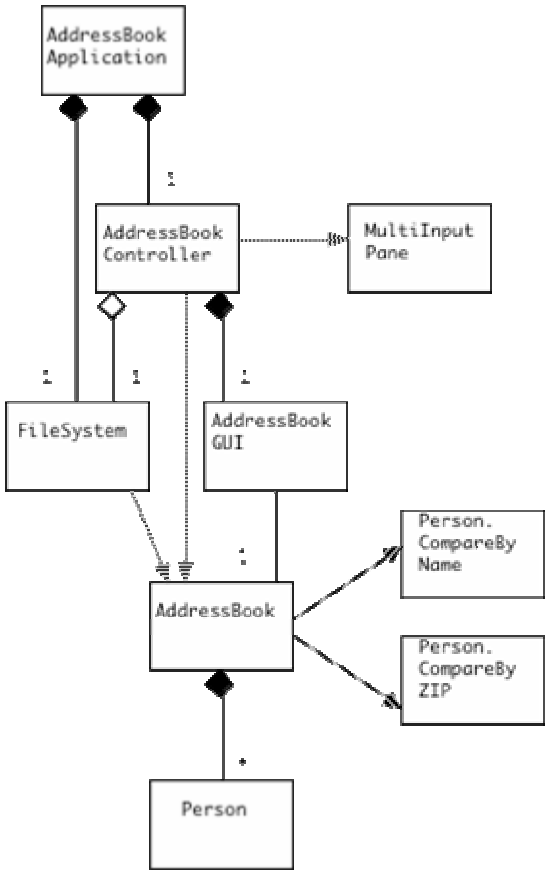
- AddressBookApplication - main class for the application; responsible for creating the FileSystem and GUI objects and starting up the application.
- MultiInputPane - a utility class for reading multiple values at a single time. (Design not further documented, but javadoc is included.)
- Person.CompareByName - Comparator for comparing two Person objects by name (used for sorting by name).
- Person.CompareByZip - Comparator for comparing two Person objects by zip (used for sorting by name).

The following relationships hold between the objects:

- The main application object is responsible for creating a single file system object and a single controller object.
- The file system object is responsible for saving and re-loading address books
- The controller object is responsible for creating a single GUI object.
- The controller object is responsible for initially creating an address book object, but the GUI is henceforth responsible for keeping track of its current address book - of which it only has one at any time.

- The GUI object and the address object are related by an observer-observable relationship, so that changes to the address book content lead to corresponding changes in the display
- The address book object is responsible for creating and keeping track of person objects, of which there can be many in any given address book.
- A MultiInputPane object is used by the controller to allow the user to enter multiple items of data about a person.
- A comparator object of the appropriate kind is used by the address book object when sorting itself.

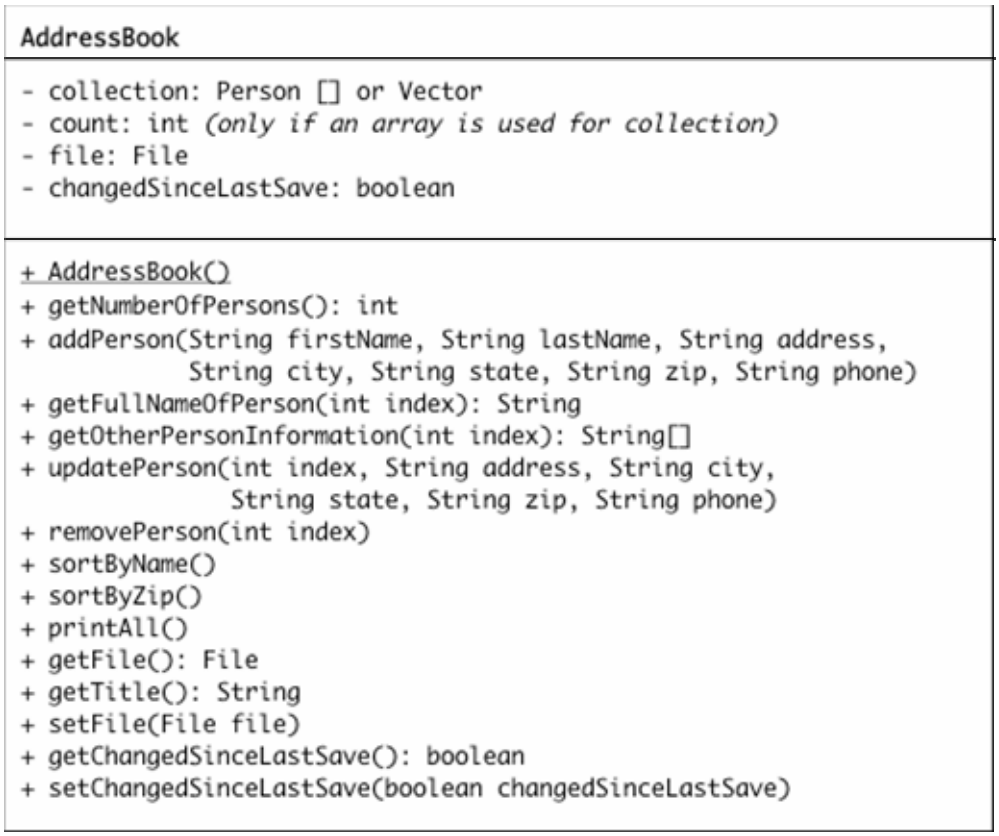
Click on a class icon for links to further information about it

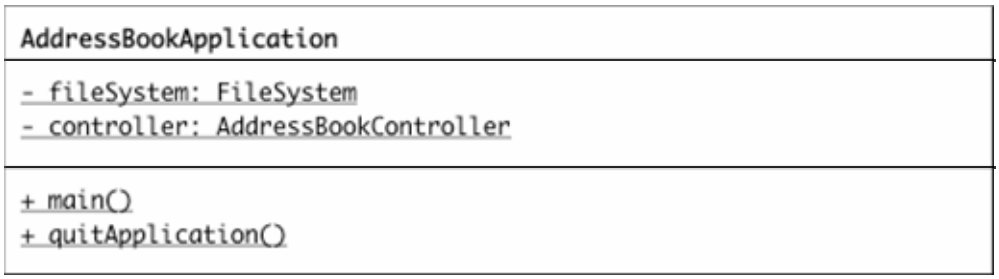


Detailed Class Design for the Address Book Example

Given below is a "three compartment" design for the classes appearing in the class diagram. This information was not included in that diagram due to size considerations; however, it could have been - in which case this document would have been unnecessary.

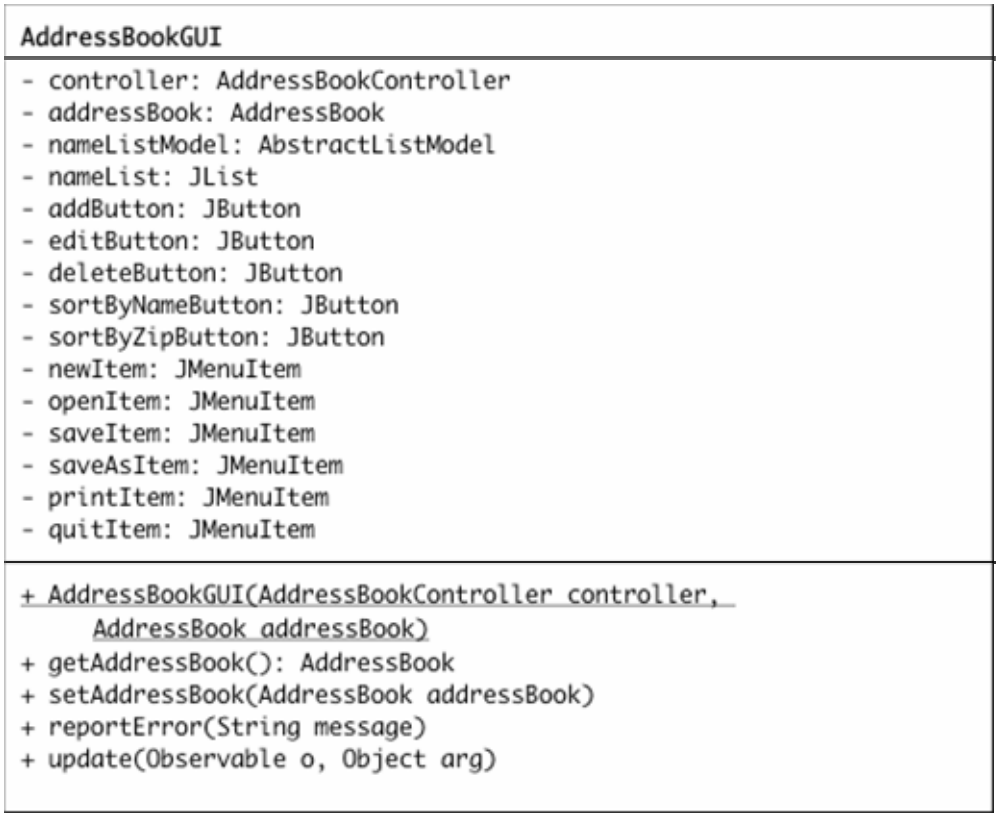
- Class AddressBook
- Class AddressBookApplication
- Class AddressBookController
- Class AddressBookGUI
- Class FileSystem
- Class Person
- (No detailed design is given for Comparator class Person.CompareByName)
- (No detailed design is given for Comparator class Person.CompareByZip)
- (No detailed design is given for utility class MultiInputPane)

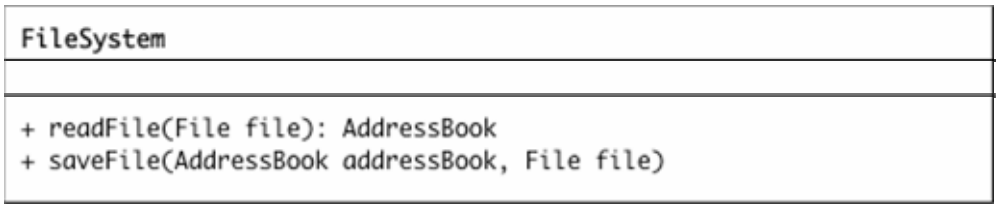


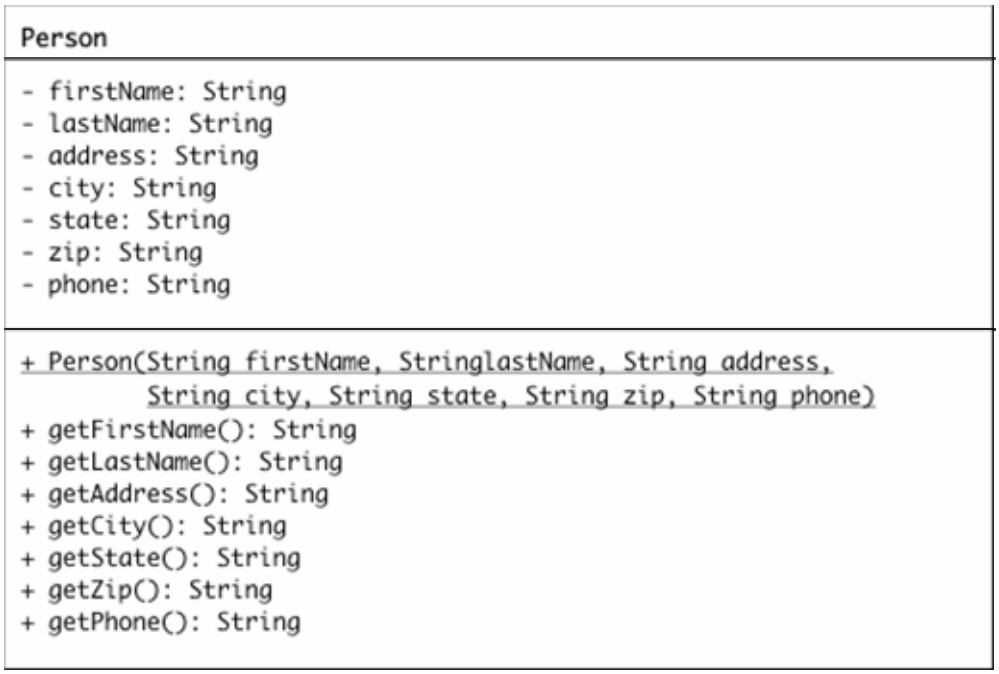


The detailed design of class AddressBookController is left as an exercise to the student

---







Code for Simple Address Book Example

As noted in the introduction, the writing of much of the code for this problem is an assignment in two closed labs and an open lab programming project in one of the courses I teach. This page includes links to portions of the code that are not assigned (and are, in fact, given to the students in the course.)

This page also provides access to Complete Javadoc Documentation for all of the classes.

- AddressBookApplication
- AddressBookGUI
- MultiInputPane
- Comparator Classes - inner classes in class Person

Executable

If you see this message, you don't have a Java-enabled browser. Sorry!

Important notes to the user:

- The Java mechanisms to protect against malicious code limit access to the file system on the host computer. For this reason, it is not possible to use the "Open", "Save", or "Save As" features of this demonstration - attempting to do so will result in an error dialog. (This will also happen if an attempt is made to save a file when performing a GUI window close or "Quit" operation.)
- "Print" will send its output to the Java console; how to access this depends on the brower.
- Closing the window or "Quit" will close the window, but will not, of course, actually quit out of the browser - hence what will be left is a blank page with no window showing. Reloading the page should restart the program - though this may be browser dependent.

Maintenance

This page lists various changes that might be made to the system. Modifying the various documents to incorporate one or more of these changes would make an interesting exercise for the reader. They are listed in order of estimated increasing difficulty.

- The Print Entries Use Case currently sends printed output to System.out. Alternately, it could send its output to a file, chosen by the user in response to a file dialog.

- It was noted in the orginal requirements that the program might be modified to allow multiple address books to be open at the same time - each in its own window. This might entail the following changes:

- The Create New Address Book and Open Existing Address Book Use Cases would no longer close the current address book. Instead, they would create a new copy of the GUI, with its own address book (either a newly created, empty one, or one read from a file specified by the user.) There would thus be two (or more) windows visible on the screen.

- A new Close Address Book Use Case would be added to allow the user to close a single window (and its associated address book). This could be initiated by a new Close option in the File menu, or by clicking the close box for the window. It would offer to save changes, if necessary, and then close the window. If the window that is closed is the last open window in the program, then the program should be terminated as well; otherwise, the program would continue running with the remaining window(s) visible.

- The code that is activated when the close box for the window is clicked would be the Close Address Book Use Case described above, instead of the Quit Program Use Case.

- The Quit Program Use Case (activated from the Quit item in the File menu) would need to cause all open windows to be closed, with appropriate offers to save changes, unless the user cancels the operation for any window. If the user cancels the save for any window, the entire use case would terminate at once, without attempting to close additional windows.

- A facility might be created that would allow the user to search for the occurrence of some character string anywhere in the information about a person. For example, searching for the string "Buffalo" might find Boris Buffalo or a person living in Buffalo, NY; searching for the string "0191" might find a person living in ZIP code 01915 or a person whose phone number is 555-0191, etc. This might entail adding two new use cases: Find and Find Again.

    - The Find Use Case could pop up a dialog asking the user to enter a character string, and would then search through all the people in the address book (starting at the beginning) until a person is found for which the string occurs anywhere in the stored information (in either name, in the address, etc.) This person would then be selected in the displayed list of names.

    - The Find Again Use Case could look for the next occurrence of the same string, beginning where the previous Find/Find Again left off.

      Of course, this option would not be available when a search is not in progress - e.g. when an address book is newly created or opened, or when the previous Find/Find Again did not find anyone. It would also be

reasonable to disable this option when any change is made to the address book (e.g. by Add, Edit, Delete, or a Sort).

To allow these two new use cases to be initiated, a new Search menu could be added with two choices, perhaps labelled Find and Find Again. In this case, Find Again would be grayed out when the Find Again Use Case is not available; and Find might also be grayed out when the address book is totally empty.

Some good practice in working with UML might come by modifying the various design documents (beginning with the use cases), not just changing the code.