A Simple Address Book

This is an example from an OOD textbook. The goal of the exercise is to have you use and think about the OO design process. In particular,

- What kinds of decisions get made?
- When do different kinds of decisions get made?
- What criteria are used to make design decisions?
- How the decisions are communicated to stakeholders?

Exercise 1: Is this a good design

Walk through the handout to understand how the design is derived. You should understand how use-case-driven OO design works:

- Walk through the design's class diagram and UML class specifications to understand the structure and function of the design
- Discuss the good and bad points of the design to arrive a team judgment
- Justify your answer: what is good about it (or bad) and why?

Using these Pages

Probably the best way to start using these pages is to begin with the requirements and work through the entire analysis, high-level design, and detailed design process.

1. Begin with the Requirements and User Interface document.

   This exercise starts with a statement of the overall software requirements, without attempting to discuss the process of actually arriving at them.

2. Then view the Use Cases and Further Analysis.

   Object Oriented Analysis typically begins by identifying the use cases that follow from the requirements, and detailing a flow of events for each. Further analysis identifies the key classes that are suggested by the use cases, and considers how each use case can be carried out by an interaction between objects belonging to these classes.
   - The Use Case document has a Use Case Diagram and a series of flows of events, one for each use case. Each use case also has a link to a Sequence Diagram (part of the Design phase) that shows how it is realized; these links can be followed while studying the design phase to see how the analysis phase flows into the design phase.
   - The Further Analysis document deals with both the "big picture" and the details of the various use cases. The former is provided by an Analysis Class Diagram, with each class having a link to its CRC card; the latter by a discussion of how the key objects would need to interact in order to implement the use case.

   These two documents represent two different ways of viewing the overall system, which continue into the next phase. The Use Case document presents a use-case centric view of the system, focusing on the specific functions it provides. The Further Analysis document presents a class centric view of the system, focusing on how will be built.

3. This example uses CRC Cards and Sequence Diagrams for high level Design. There are certainly other tools that might be used - e.g. the ATM Example referred to above makes use of Collaboration Diagrams and State Charts as well.

- The responsibilities of each class that arise from the use cases are recorded on a CRC card for each class. The CRC cards could be created by "walking through" each use case, assigning the responsibility for each task to some class.
- There is a Sequence Diagram for each use case, showing how the use case is realized by interaction of the major objects.
- A Class Diagram shows how the various classes are related to one another. It also shows additional classes that were "discovered" during the process of creating the Sequence Diagrams - i.e. classes needed to actually build the system, though not evident in the original analysis.

Requirements Statement

The software to be designed is a program that can be used to maintain an address book. An address book holds a collection of entries, each recording a person's first and last names, address, city, state, zip, and phone number.

It must be possible to add a new person to an address book, to edit existing information about a person (except the person's name), and to delete a person. It must be possible to sort the entries in the address book alphabetically by last name (with ties broken by first name if necessary), or by ZIP code (with ties broken by name if necessary). It must be possible to print out all the entries in the address book in "mailing label" format.

It must be possible to create a new address book, to open a disk file containing an existing address book to close an address book, and to save an address book to a disk file, using standard New, Open, Close, Save and Save As ... File menu options. The program's File menu will also have a Quit option to allow closing all open address books and terminating the program.

The initial requirements call for the program to only be able to work with a single address book at a time; therefore, if the user chooses the New or Open menu option, any current address book will be closed before creating/opening a new one. A later extension might allow for multiple address books to be open, each with its own window which can be closed separately, with closing the last open window resulting in terminating the program. In this case, New and Open will result in creating a new window, without affecting the current window.

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.

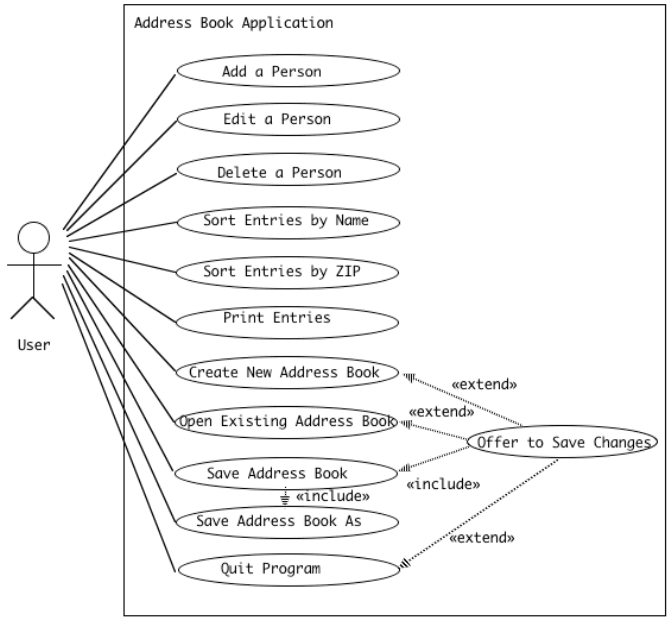
The program will keep track of the file that the current address book was read from or most recently saved to, will display the file's name as the title of the main window, and will use that file when executing the Save option. When a New address book is initially created, its window will be titled "Untitled", and a Save operation will be converted to Save As ... - i.e. the user will be required to specify a file.

User Interface

Because this is to be a "standard GUI" style application, some attention needs to be given to the user interface at this point. A user interface like the following might be adopted. Not shown in the screen shot is a File menu with New, Open, Close, Save, Save As ..., Print, and Quit options. For the "Edit" and "Delete" buttons, the user must first select a person in the scrolling list of names, and then can click the appropriate button to edit/delete that person.

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.



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.

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.

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.

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.

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.

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.

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.)

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.)

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.

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.

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.

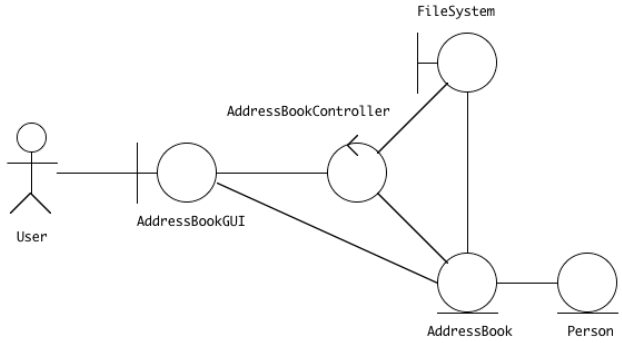
Offer to Save Changes Extension

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.

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 are 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

---

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

---

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

---

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

---

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	

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 zipß).

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.