The `pinemerge` program: a Python design study

New Mexico Tech
Computer Center www.nmt.edu/tcc

This document describes a program that merges two or more Pine address books. Not only is it a useful program, but it is also a good case study in object-oriented design.

We will look first at the operation of the program, then proceed to an analysis of its inner workings.

Operation of `pinemerge.py`

Normally Pine keeps its address book in a file called `.addressbook` in your home directory. However, if you have more than one account, each will have a separate address book. This program can help you resolve them to a single version containing all the addresses from multiple versions.

Suppose you have two Pine address books named `b1` and `b2`. To merge them, run this command:

```
/u/john/bin/pinemerge b1 b2 >newbook
```

This will read files `b1` and `b2` and produce a new address book file named `newbook`.

If there are any entries in the input files that have the same nickname but different full names or addresses, the program will write a message for each conflict that will look something like this:

```
Duplicate nickname, omitted:
bill (Bill Gates) bill@microsoft.com
Conflicts with:
bill (Gates, William) billg@microsoft.com
```

This message shows the entry that will be ignored, followed by the entry that will be retained.

Once you have built your new addressbook, install it by following these steps:

1. Save a copy of your current `.addressbook` file in case of problems:

   ```
   cp .addressbook address.backup
   ```

2. Copy the new addressbook into place. If the new addressbok is called `newbook`, you would use this command:

   ```
   cp newbook .addressbook
   ```

3. Delete the `.addressbook.lu` file:

   ```
   rm .addressbook.lu
   ```

4. Bring up Pine and check your address book to make sure it looks okay. (If it doesn't, restore your original copy by copying the backup you made in step 1 back to `.addressbook`.)

The design of `pinemerge`

You can find the source for the `pinemerge` program at this location:

```
/u/john/tcc/src/python/pine/pinemerge.py
```

The format of a Pine address book is documented at this URL:

```
http://www.washington.edu/pine/tech-notes/low-level.html#addrbook
```

The reader is assumed to be familiar with the Python language.

When designing with an object-oriented language, the usual procedure is to look at the problem to see what things in the real world are being represented. Quite often there will be one object in your program for each such real-world object.

For this problem, there are two pretty obvious objects involved:

- An entry in an address book, consisting of the nickname, a full name, an address, and two optional fields (the `fcc` and `comments` fields).
- The address book as a whole is an object. It can be considered a container for address objects.

In the source, the address object is `Class Address` and the address book object is `Class AddressBook`.

We will examine the two principal objects first, and then discuss the operation of the program's main.

The `Address` *object*

Each instance of the `Address` class in the program represents one entry in an address book. This object hides all of the information about how an address book entry appears in the `.addressbook` file format.

An instance has these members:

- The `.nickname` field is the nickname, e.g., `"dave"`.
- The `.fullName` field is the fullname, e.g., `"Barry, Dave"`.
- The `.address` field is the e-mail address, e.g., `"dave@miami.foo"`.
- The `.fcc` field holds the "Fcc" string if there is one; it has the value `None` if there is none.
- The `.comments` field holds the comments string; it may be `None` if there are no comments.

The `Address` object has these methods:

`Address(`n`,`f`,`a`,`F`,`c`)`
: The constructor function takes three to five arguments. The n value is the nickname; f is the fullname; a is the e-mail address; F is the "Fcc" string if any; and c is the comment string if any.

`.__str__()` This method produces a list of one or more newline-terminated strings containing the address in the format used inside the Pine `.addressbook` file. Most entries will fit on a single line, but long comments or address lists will be wrapped according to rules described in the Pine web page mentioned above. The name `__str__` has special meaning to Python, and will be invoked when an object appears in a `print` statement or as the argument of the built-in `str()` function.

`.__cmp__()` This method name is also special to Python, and will be invoked by anyone who uses an `Address` object in a call to the built-in `cmp()` function. Defining this method also allows users of the class to sort a list of `Address` objects using the built-in `.sort()` method for list objects. The method sorts entries by full name, using the nickname as a tie-breaker.

`.show()` This method produces a single string showing the three principle parts of an address (nickname, full name, and address) in a human-readable format. This method is called to display addresses in error messages, and it can also be handy for debugging.

The `AddressBook` *object*

Each instance of an `AddressBook` object represents an entire Pine address book. One member is exported (that is, made available outside the class):

- `.nickMap` is a dictionary that maps nickname strings to the corresponding `Address` object for that nickname.

Methods on the `AddressBook` object include:

`AddrBook(`f`)` This is the constructor for the class. To read an existing address book file, supply its name as the f argument; the returned object will have in it all the addresses found in that file. If the argument is omitted, you will get back an empty `AddressBook` object.

`.insert(`a`)` Tries to add an `Address` object a. If the address book already has an entry with the same nickname, it raises a `ValueError` exception.

`.write(`o`)` The o argument must be a writeable Python file object. This method writes the entire address book's contents to o in the Pine address book format.

The main program

Given the existence of the `AddressBook` and `Address` objects, the operation of the program is straightforward.

1. We create an empty address book object called `outBook`. This will hold the merged addresses from all the various input files.
2. For each input file named on the command line, we read that input file into a new `AddressBook` object called `book`.
3. We extract each entry in `book` and try to add it to `outBook` by using the `.insert()` method. If that fails, we compare the entry to the existing one in `outBook`. If the two are the same, then there's no problem. If they're different, though, we print an error message to `sys.stderr`, using the `.show()` method to display the two entries.
4. Once we have added all the input address books to `outBook`, we use the `outBook` object's `.write()` method to write the merged address book, and we're done.

Written by John W. Shipman (`tcc-doc@nmt.edu`). This version printed 2003-10-08.