Assignment No. 6

This is the first of a two-part assignment. In this part of the assignment you will write a class for maintaining an address book. In the second part (assignment no. 7) you will implement a graphical user interface (GUI) for this class. This division of the assignment into two parts is motivated by the model-view separation paradigm, which dictates that the model of an application (logic and functionality) should be separated from the visual representation (the GUI in our case). The rationale behind this approach is that visual representation tends to change a lot. Model-view separation ensures us that view changes will not affect the basic model and enables us to maintain one model for several different views.

Another purpose of this part of the assignment is to familiarize you with the JUnit tool for writing unit tests.

The requirements from the `AddressBook` class are:

- Each entry in the address book (defined in a class named `Contact`) will consist of the following fields:
  - **name**: a string representing last and first names separated by a comma, for example "Smith, John". This field is mandatory and thus cannot be null.
  - **email**: a string (may be null)
  - **telephone**: a string (may be null)
  - **address**: consists of four strings that stand for: street, city, zip code and country that all or some of them may be null
- The `AddressBook` class will support the following operations:
  - **`public void add(Contact c)`** – add a new contact. A new contact is a contact whose name doesn't already exist in the address book.
  - **`public Contact get(String name)`** – return the contact of the given name.
  - **`public void delete(String name)`** – delete the contact of the given name.
  - **`public void modify(Contact c)`** – replace an existing contact with a new one one. The name field of the contacts should be the same.
  - **`public Iterator getContacts()`** – return an iterator over the contacts in the address book , sorted by names alphabetically.
  - **`public int getCount()`** – return the number of contacts in the address book
  - **`public Iterator search(String prefix)`** – return an iterator over the contacts in the address book whose names start with the given prefix. The order is by names, alphabetically.
  - **`public void save(String filename)`** – save the whole address book to the given file (using serialization).
  - **`public void load(String filename)`** – load an address book from the given file name (using serialization)

Your assignment is:

- Implement the `AddressBook` class as described above. Add appropriate exceptions to its methods (e.g. when trying to get delete or modify non-existing contact).
- Define the contract of the class and write JUnit test cases for checking it (i.e. black-box tests). Note that your test cases should pass any implementation that satisfies the specification of the routing table (and not only your implementation). The test cases will be checked by running them

on different implementations including buggy implementations to see whether they report errors/failures.

- In a different package write a textual user-interface application for the `AddressBook` class. The application will get as an argument a filename. Each line is the given file should be one of the following records:
  - **`n`** – for creating a new address book
  - **`l <filename>`** - for loading an address book from a file
  - **`a <name>;<email>;<telephone>;<street>;<city>;<zip>;<country>`** - for adding a new contact to the address book. Note that only the name filed is mandatory. The rest of the fields can be null, but the order of fields and the ";"s are significant.
  - **`p <name prefix>`** - for printing to the standard output the first contact whose name starts with the given prefix
  - **`x`** – for printing all the contacts in the address book
  - **`d <name>`** - for deleting the contact of the given name
  - **`m <name>;<email>;<telephone>;<street>;<city>;<zip>;<country>`** - for replacing an existing contact with the given one. Note that only the name filed is mandatory. The rest of the fields can be null, but the order of fields and the ";"s are significant.
  - **`s <filename>`** - for saving the address book to a file

Here is an example for an input file for the textual user-interface application:

```
n
a Smith, John;smith@gmail.com;03-6404324;23 Laskov St.;Tel-Aviv;56743;Israel
a Stein, Rita;rita@gmail.com;03-5524324;;;;
a Altman, Rebecca;rebecca@gmail.com;03-9414324;;Tel-Aviv;42732;Israel
a Altman, David;david@gmail.com;;;;
p Altman
p Altman, Rebecca
x
d Stein, Rita
m Altman, David; david@gmail.com; 03-9414324;;;;
x
s addresses.data
```

Note that the first line must be "`n`" or "`l <filename>`".
Note also that the textual user-interface will be replaced by a GUI in the assignment 7.