Exercise No. 11
Address Book – Part I

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 and in the second part 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.

The requirements from the `AddressBook` class are:

- Each entry in the address book (`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, can be null.
  - **telephone**: a string, can be null.
  - **address**: consists of several strings that stand for: street, city, zip code and country. Can 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 for which the name does not 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. A contact with the same name should appear in the address book before the call.
  - **`public Iterator<Contact> 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<Contact> search(String prefix)`** – return an iterator over the contacts in the address book for which the name starts 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:

- Choose the underlying data structure to be the representation of the `AddressBook`. This data structure should be a standard `java.util` collection or map. You may consult java libraries API (http://java.sun.com/j2se/1.5.0/docs/api/) to find more about the services offered by the different classes.

- Implement the `AddressBook` class as described above. Add appropriate exceptions to its methods.
- 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.
  - **`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.
  - **`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>`".

  Write this application based on the model-view separation paradigm, that is separating the logic of address book management from the logic of address book presentation (*modularization*). This is done by using two separate classes. If sometime in the future we would like to use cool widgets to present our address book, we wouldn't have to change *anything* inside `AddressBook` class, we only need to implement a new class for the new representation. This way we keep `AddressBook` class 'clean' and increase the *reuse* in our system.