תוכנה I בשפת ג'אווה
בית הספר למדעי המחשב
אוניברסיטת תל אביב

Exercise No. 4
Address Book – Part I
Due Date: 25/12/2005

This is the first part of an address book application assignment. In this part of the assignment you will write a class for maintaining an address book. Later in the semester you will implement serialization (saving/loading data to/from files) and interactive update using a graphical user interface (GUI) for this class.

Differing the GUI implementation 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. Should be represented as an instance of the class `Address`.

- 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. 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 whose names start with the given prefix. The order is by names, alphabetically.

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. To help you, we wrote the interface `IAddressBook` (that can be downloaded from the course web site), that you need to implement (`AddressBook implements IAddressBook`).

- Test your `AddressBook` implementation using a designated class named `TestAddressBook` that will act as a unit test case with black-box tests for the `AddressBook` class. Design at least three test cases (scenarios) in which `AddressBook` is used. Define three methods in `TestAddressBook`, named `test1`, `test2`, `test3` that will conduct the test cases. Create a `main` method to call these methods. The test cases should not be trivial and should demonstrate testing the contract as well as corner cases. Note also that your black-box test cases should be general and pass any implementation that satisfies the specification of the `AddressBook` class and not only your implementation. In the context of the exercise, this means that the exercise-checker may check your test class by running it on different implementations including buggy implementations to see whether it reports errors/failures.

- Of course that you are advised to test *all* your classes and to use also white box testing but in this assignment you are required to submit only `TestAddressBook`.

- The `AddressBook` may be presented on the screen using a designated class, `TextualAddressBookViewer`. This class implements the `IAddressBookViewer` interface (this is a popular coding convention – interface name starts with 'I'), which contains 3 methods:

    - **`public void showAddressBook()`** – presents the attached address book. For this method to work an addressBook should already be attached.
    - **`public void showContact(Contact c)`** – presents the details of contact `c`.
    - **`public void attachAddressBook (AddressBook book)`** – attach `book` as the AddressBook to show later.

  `TextualAddressBookViewer` should print its output to the screen using standard printing methods of `System.out` (or any other standard printing method). It may maintain indentation, alignment and "pretty printings". `TextualAddressBookViewer` will not be tested by the automatic checker so creativity is allowed.

  Defining an interface for the presentation of the class is very important. We would like to separate the logic of address book management from the logic of address book presentation (*modularization*). This is done by using 2 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 (say `GUIAddressBookViewer`) which also implements `IAddressBookViewer` interface. This way we keep `AddressBook` 'clean' and increase the *reuse* in our system. You can download `IAddressBookViewer.java` from the course website.

- To help you test your classes, we wrote a sample client code, which can be downloaded from the course website.

What to hand in?

1. Softcopy:
   - `AddressBook.java`
   - `Contact.java`
   - `Address.java`
   - `TextualAddressBookViewer.java`
   - `TestAddressBook.java`

   All the code should be written under the package *il.ac.tau.cs.<your user name>.ex4*
   You should pack all the classes in *ex4.jar* and give it proper permissions as explained in the submission guidelines. Note that the `TextualAddressBookViewer` and `TestAddressBook` classes will be checked manually (would *not* be graded by the automatic checker).

2. Hardcopy:
   - Printout of all your classes.
   - Javadoc printout of all your classes. Documentation should include DbC assertions for all classes and methods.
   - On the first page of the hardcopy write your name, id and login.

Good luck