# Address Book Case Study (Version 1.0)

We want to build an address book that is capable of storing name, address & phone number of a person. The feature list includes

- Add – to add a new person record
- Delete – to delete an existing person record by name
- Search – to search a person record by name
- Exit – to exit from application

**Approach for Solving Problem**
Building a small address book generally involves 3 steps. Let us briefly discuss each step and write a solution code for each step.

**Step1 – Make PersonInfo class**
First of all you need to store your desired information for each person. For this you can create a user-defined data type (i.e. a class). Make a class *PersonInfo* with name, address and phone number as its attributes.

**Step2 – Make AddressBook class**
Take the example of daily life; generally address book is used to store more than one person records and we don't know in advance how many records are going to be added into it.
So, we need some data structure that can help us in storing more than one *PersonInfo* objects without concerning about its size. Any appropriate collection object like ArrayList can be used to achieve the above functionality.
This class will also provide *addPerson*, *deletePerson* & *searchPerson* methods. These methods are used for adding new person records, deleting an existing person record by name and searching among existing person records by name respectively.
The *addperson* method first takes input for name, address and phone number and than construct a *PersonInfo* object by using the recently taken input values. Then the newly constructed object is added to your collection object.
The *searchPerson* & *deletePerson* methods are using the same methodology i.e. first they search the required record by name and than prints his/her detail or delete the record permanently from your collection object. Both the methods are taking string argument, by using this they can perform their search or delete operation. You can use for-each loop for iterating the whole container.

**Step3 – Make Test class (driver program)**
This class will contain a *Main* method and an object of *AddressBook* class.
Build an option menu by using switch selection structure.
Call appropriate methods of *AddressBook* class.

# Address Book Case Study (Version 1.1)

**Adding Persistence Functionality**
Hopefully, your address book you built previously is giving you the required results except one i.e. persistence. You might have noticed that after adding some person records in the address book; if you exit from the program next time on re-executing address book all the previous records are no more available.
To overcome the above problem, we will modify our program so that on exiting/starting of address book, all the previously added records are available each time. To achieve this, we have to provide the persistence functionality. Currently, we will accomplish this task by saving person records in some text file. Supporting simple persistence by any application requires handling of two scenarios. These are
. On start up of application – data (person records) must be read from file
. On end/finish up of application – data (person records) must be saved in file
**Step 1:**
. Establish a data channel with a file by using streams
. Start reading data (person records) from file line by line
. Construct *PersonInfo* objects from each line you have read
. Add those *PersonInfo* objects in arraylist.
. Close the stream with the file
. Perform these steps while application is loading up
We will read records from a text file named *persons.txt*. The person records will be present in the file line by line (i.e. one record per line). Each person record will be on a separate line. Person's name, address & phone number is separated using comma (,).
We will modify our *AddressBook.java* by adding a new method *loadPersons* into it. This method will provide the implementation of all above mentioned steps.
First, we have to connect with the text file.
The next line of code will read line from file by using *readLine( )* method. The condition of while loop is used to check whether the file is reached to end (returns null) or not. In while loop we will start to reads the line from a file, tokenize that line into three substrings followed by constructing the *PersonInfo* object by using these tokens and adding these objects to the arraylist. This process continues till the file reaches its end.

```
String tokens = line.split(“,”);
```

The next three lines of code are simple assignments statements. The tokens[0] contains the name of the person because the name is always in the beginning of the line, tokens[1] contains address of the person and tokens[2] contains the phone number of the person.
**Step 2:**
Build a string for each *PersonInfo* object by inserting commas (,) between name & address and address & phone number. Write the constructed string to the file line by line at the exit time of program.

# Address Book Case Study (Version 1.2)

**You have to modify the version 1.1 in such a way that instead of reading and writing complete person information on file we can save and retrieve *PersonInfo* object directly from a file. i.e Serialization**