## System Documentation

I tackled the problem by subdividing it into the required methods. Further, I used different classes for the methods for easy error debagging and readability. I have 6 classes:

Nokia, AddRecord, SearchByName, SearchByPhoneNumber, DeleteRecord && PhoneVaildator

### NOKIA

All other classes and methods, *AddRecord(), SearchByName(), SearchByPhoneNumber() && DeleteRecord(),* are called here.

Takes in one int parameter and matches it to a method using a switch-case.

The methods are wrapped in a while and try block, to create a persistent execution unit on the terminal.

### ADDRECORD

Takes in two parameters, name and phone number. Before further progress, inputs are checked to ensure they are not null. Then the method checks if Phonebook.txt exists. If not, create a new PhoneBook.txt.

Then Phone number is passed through the phone validator to check if it has been used before and if is valid (is at least 10 characters long and does not contain letters)

Upon successful validation, the program uses a file writer mechanism to append the validated details as a key-value pair, delimited by a colon (":"), to the "PhoneBook.txt" file.

### SEARCHBYNAME

Takes a String parameter. Checks if input is not null, then removes any whitespaces, and converts it to lower case, for ease to work with.

Subsequently, employing a while loop, the method iterates through records to identify a matching name. Upon discovering a match, it captures associated numbers, storing them in an array, then prints them out.

**subscribers are maximum of 10 phone numbers. To make it dynamic, I would need to use an arraylist.**

### SEARCHBYPHONENUMBER

Takes a String as parameter. Does initial check to ensure no null inputs and removes white spaces as well as “+” sign. This is to allow for versatile terminal input acceptance.

Runs a check on the number using the methods in phone validator class.

If the input passes successfully, then I do the reverse of SearchByPhone. I search for the phone number and return a string with the matching number.

### DELETERECORD

Takes in two string parameters and does an initial check on the number using methods in phone validator class; of course, after checking that they are not null.

If the record passes and is located, it is marked for deletion. A new file content (input) is constructed, excluding the deleted record.

Then it writes the modified content back to the file, effectively erasing the specified subscriber's record.

### PHONEVALIDATOR

This constitutes the core of my code, encapsulating a class with four essential methods:

i. PhoneNumberExists: Verifies the existence of the phone number, incorporating error handling by triggering an exception if the number is already present.
ii. PHONE_NUMBER_PATTERN: A constant representing a regular expression pattern.
iii. ValidPhoneNumber: Validates whether the phone number meets the specified length criteria.
iv. ValidPhoneNumber2: Confirms that the input for the phone number solely consists of integer values.