Assessment

In this assessment task, we are going to implement a **Phonebook** application in Java.

A Phonebook application stores the contact information of your friends and their phone number and helps you search throughout the data, including the phone number as well as the first and last names.

Overview

The phonebook contains **entries** of contacts with a person's first and last names, one or more phone numbers, and a category assigned to each phone number.

Here is an example of two phonebook entries; each entry is separated by a 3-dashed line (`---`):

```
First name  : Harry
Last name   : Potter
MOBILE      : +374 55 123456
OFFICE      : +374 10 875803
HOME        : +374 10 689651
---
First name  : Sheldon
Last name   : Cooper
MOBILE      : +374 77 975892
OFFICE      : +374 11 573203
OFFICE      : +374 11 573204
---
First name  : Jon
Last name   : Snow
MOBILE      : +374 91 543216
```

The category of a phone number can be any of the `MOBILE`, `OFFICE`, and `HOME`. Note that we **DO NOT** restrict a person to have only one number per category as well as we **DO NOT** require a person to have a phone number for each of the categories. However, we limit each person to have at most 5 phone numbers in the Phonebook.

The Phonebook application works with data provided in a text file (a sample text file is included in the task description). Therefore, inserting phonebook entries is not part of the Phonebook application and is done simply by writing to a text file and placing it next to your application.

The application starts by asking for action from the user via the command line:

```
*** Phonebook Search Application ***

Choose your action:
1: Find by name
2: Find by number
> 1
```

```
Enter first or last name:
> Har

Record(s) found:
First names : Harry
Last names  : Potter
MOBILE      : +374 55 123456
OFFICE      : +374 10 875803
HOME        : +374 10 689651

*** Exit ***
```

As you noticed, the program searches either by first or last name and is able to search by partial tokens (e.g. Har qualifies for Harry and Harrison). You can specify a minimum length of 3 letters to search with.

If there is more than one record that qualifies for the provided token ("Har" is a search token), all the matching records should be displayed.

It should also be possible to search for names by providing a phone number:

```
*** Phonebook Search Application ***

Choose your action:
1: Find by name
2: Find by number
> 2

Enter the phone number:
> +37411573203

Record found:
First name  : Sheldon
Last name   : Cooper
MOBILE      : +374 77 975892
OFFICE      : +374 11 573203
OFFICE      : +374 11 573204

*** Exit ***
```

Since a phone number can belong to only one person, searching by phone number will return at most one result.

In this example, we provided an exact match (+37455123456) for the phone number. If you can implement the program in a way that it can also search by other variations of the phone number (e.g. '055 123456', or '123456', or '055123456') that will count as an additional feature and will be evaluated accordingly.

Implementation

- Your implementation should use OOP and design pattern concepts whenever possible. Avoid writing the entire code in one big class.
- Use Java enums in order to implement the phone number categories (MOBILE, OFFICE, HOME).
- Error handling: Since the application accepts input from the command line, it is possible for the user to provide incorrect input. Your program should handle such scenarios either by terminating the program or giving another chance to the user to repeat the step and provide a correct input.
- Input file: A file named `data.txt` contains the phonebook entries that should be used by the application. However, you should not assume that the contents of the file will always be correct. It is possible that someone provides an incorrect file (a file that does not contain data with the right format). Therefore, you should perform some basic validation when processing the `data.txt` file and display an error if the provided file is incorrect.
- Use comments to explain your thoughts whenever you find it useful.
- Start by implementing the minimum workable version and then focus on each part to improve and make it better. (*Make it work, then make it beautiful, then if you really, really have to, make it fast. 90 percent of the time, if you make it beautiful, it will already be fast. So really, just make it beautiful!* – Joe Armstrong)

Searching Algorithm

You are free to implement the searching part of the application in any method you prefer. We expect you to **describe the running time** of the algorithm that you use to find contact entries by name or by phone number.

BONUS POINT. It is encouraged that you try and use a data structure that will make the search process faster and more efficient. Again, you should describe and explain the running time of the algorithm.

DevOps

Write a shell script that removes any compiled classes, and compiles the java source files. Extend the shell script that will execute a Java application that you have developed that will process it by writing output to new files.

Use your created script. Make a Dockerfile where the entry point command runs the mentioned script. Name your script docker.entrypoint.sh, and create a docker container, which will use your newly created docker image. So, when you run the docker container, that output must be the same as your script's output.

Bonus Question

Data Structures and Algorithms

Implement a recursive function that determines whether the given array is sorted in descending order or not. Write comments to explain your code and state the complexity of the function in them.

Input: Array of numbers. Output: boolean.
Examples: IsDescending([6, 4, 3, 3, 1]) = true, IsDescending([6, 4, 3, 4, 1]) = false

Loops are prohibited, you should use recursion.