Additional exercise for June

Create a project called **HospitalManagement** in IntelliJ. We are going to implement an application to manage the appointments in a hospital with different doctors and nurses. To begin with, define these packages in the project:

- `hospital.people`
- `hospital.management`

Staff hierarchy

We are going to deal with two different types of staff in the hospital. Both will inherit from a parent, abstract class called `Staff`, with these attributes:

- `id`: a String with the ID of this person
- `name`: the full name of this person
- `email`: the email of this person

You must define, at least, a constructor to set all these values and the corresponding getters. Also, you must override `toString` method to show the information in one line, with the format that you prefer.

Then, there will be two subtypes of `Staff`:

- `Doctor`, which will represent all the doctors of the hospital. For each doctor, we need to store his/her medical specialty, such as "surgery", or "oncology".
- `Nurse`, which will represent all the nurses. For each nurse, we are going to store the floor number in which he/she is currently working.

Define the constructors to set all the attributes of these classes (relying on parent class), along with the getters and the `toString` method, that must include the same information than parent class, along with the specific information of each subclass.

All these classes must be placed inside `hospital.people` package.

The patients

There can be a lot of patients coming to this hospital. To manage them, we are going to define a class called `Patient`, inside `hospital.people` package, with the following information:

- `id`: a String with the ID of this patient
- `name`: the full name of this patient
- `phoneNumber`: the phone number of this patient

You must define the constructor to set all these attributes, the corresponding getters and a `toString` method to return a String representation of this patient.

File management

Inside the `hospital.management` package, define a class called `FileManagement` with these methods:

- A method called `readStaff`, that will read a file called "*staff.dat*" and return a generic list of `Staff` objects, including both doctors and nurses. The format of this file will consist in a list of lines, one per person, indicating its staff type (*Doctor/Nurse*), followed by all the information separated by `#`. For instance:

```
Doctor#1231253A#John Doe#john.doe@gmail.com#Surgery
Nurse#2512523E#Elijah Reeve#ereeve@outlook.com#3
Doctor#3114767K#Susan Dean#susand1987@gmail.com#Oncology
...
```

- A method called `readPatients` that will read a file called "*patients.dat*" and return a list of `Patient` objects. The format of this file must include one patient per line, with his/her information separated by `#`:

```
21251235G#Helen Rogers#611223344
13312335I#Paul Peters#677889900
...
```

Making appointments

At the beginning, the application must load the lists of staff and patients (two separate lists).

In order to make appointments, patients must enter the application with their ID. If this ID is valid, then the program must ask them to choose a doctor from the list (only a doctor). To do this, the program must show the list of doctors sorted by name in ascending order, and the patient must write a valid ID for the chosen doctor (if ID is not valid, the patient must try again until it writes a valid ID).

Once the patient chooses a doctor, then he/she must choose a date (typing it with the format *yyyy-mm-dd*, for instance, *2021-06-14*). Finally, the system will randomly choose a nurse for this appointment.

With all this information (Patient information, chosen doctor, date and nurse), the program must create a text file with all the information. For instance:

```
Patient: Helen Rogers, 21251235G, 611223344
Doctor: John Doe (Surgery)
Date: 2021-06-14
Nurse: Elijah Reeve
```

The file name must be the same than patient's ID, with *.txt* extension. For instance, "*21251235G.txt*".

> **NOTE**: you should (must) add a new method called `writeAppointment` in `FileManagement` class to store this information. Pass the appropriate parameters to this method to complete the task.

After writing the appointment file, the application must finish.

Evaluation criteria

- Project and packages properly defined: 0,5 points
- Parent, abstract `Staff` class with appropriate attributes, constructor and method(s): 1 points
- Staff subclasses, with appropriate constructors, method overriden, etc: 1 points (0,5 points each)
- `Patient` class with its corresponding attributes and methods: 1 points
- `FileManagement.readStaff` method to load a list of *Staff* objects: 1,5 points
- `FileManagement.readPatients` method to load a list of patients: 1,5 points
- Patient access to the application: 1 point
- Appointment management (asking data to patient and writing the file): 2 points
- Code cleanliness and reusability: 0,5 points

> **VERY IMPORTANT**: every exercise that does not compile will be automatically evaluated to 0.