Medical clinic management program

The goal of this project is to develop a basic information system for a small clinic. This clinic has two doctors, a number of nurses and a number of patients. Each patient is allocated to a doctor but the nurses work for both doctors. In this clinic, we have two doctors. Joanne Healthy and Johnny Hefty and an undetermined number of nurses. The UML class diagram is provided below.

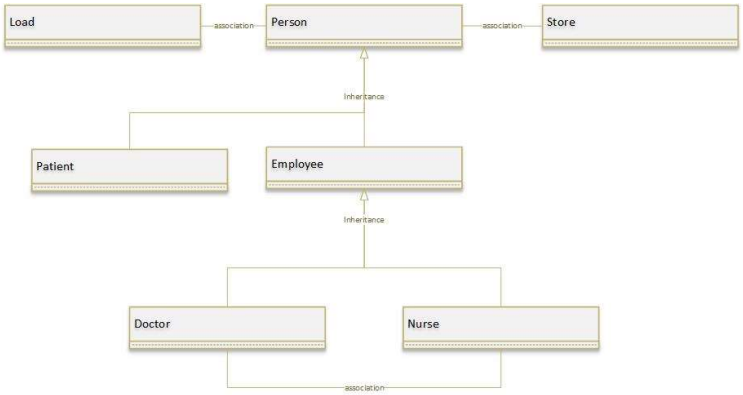

Figure 1. Simplified UML class diagram

The system has two starting modes: creation of a new clinic or loading existing the data from files and modification of this data. Please note that the id of the nurses, patients and doctors are independent. This means that patient, a nurse and a doctor may have the same id (e.g. 1).

1. Class Person

The class person has the following attributes:

fName, lName, stAddress, zipCode, city, dayBirth, monthBirth, yearBirth, SSN, gender

2. Class Patient

idPatient, allergy (bool), prescriptions (array of strings).

3. Class Employee

salary

4. Class Nurse

Specialty, practitioner (bool), nbNurses, idNurse.

- bool reward(): this function checks if a nurse is involved in more than ten appointments with either doctors. It returns true or false. If true, she will get 10% bonus.

5. Class Doctor

idDoctor, speciality, degree, medicalSchool, boardCertified (bool), patients (dynamic array), appointments (dynamic array).

Function:

- addNurse() this function calls the constructor of nurse with all the necessary parameters. It creates a new nurse and increments the number of nurses in the clinic.
- delNurse(nurseObject) it removes the object nurse and decrements the number of nurses in the clinic. Removing an object from an array is done by displacing the items after it one after the other tile the last object in the array. This function calls then the destructor of nurse given as a parameter.
- addPatient(): adds a new patient to the array of the doctor's patient.
- addAppointment(idPatient, hour, day, month, year, idNurse): adds a new appointment to the schedule of the doctor. You may assume that the doctor gives appointments for the five coming working days. For every day, there are eight time-slots with one-hour break between 12-1PM. You should display an error message if you are trying to add a new appointment in an already taken time slot. Obviously, your scheduling array should be a 2D array. You should also check that the nurse is not involved in another appointment with the other doctor in the same time. This means that you should take the object of the other doctor as a parameter, as well, to be able to access his schedule through the *checkNurse* function.
- checkNurse(idNurse, hour, day, month, year, idNurse) returns boolean to indicate if a nurse has an appointment in the given time parameters.
- cancelAppointment(idPatient): cancels an appointment from the schedule of the doctor.
- deleteAppointment(idPatient, hour, day, month, year): replaces the existing time variables with the newly provided ones.

6. Class LoadObjects

At the end of every session the program should store the patients', nurses' and doctors' data in a .txt files. This class should include a function that loads the existing data from the file. You can merge this class with the StoreObjects class if you want.

The *loadData* function should do the following. First, we should open the file clinicData.txt. This file should contain the following information:

nbNurses, Dr1Capacity, Dr1NbPatients, Dr2Capcity, Dr2NbPatients. For example, if we may have a file with the following data: 3 30 25 40 29. In this case, we have three nurses, the first doctor's capacity is 30, and he has 25 patients. If the file is empty or if it does not exist, that means that we should build the clinic from scratch.

Then we should make the names of the files. For example, imagine we have three nurses. Then we should make three file names for nurses: "nu" + "1" + ".txt" open the file and read the information and instantiate a nurse with the right parameters then do the same with "nu"+"2"+".txt" and "nu"+"3"+".txt". Obviously, we need a loop for this work.

For the doctors, you may find the data in the files dr1.txt and dr2.txt. For the patients, you may generate the files as follows: dr1_pat1.txt, dr1_pat2.txt, dr1_pat3.txt, etc. In this example, Dr1 has 25 patients. For the second doctor, obviously the files should start with dr2. Based on the number

of patients obtained from clinicData.txt, you need to generate the names of the files using a loop and open every file and create the object with the parametrized constructor. The doctors' data may be retrieved in files called dr1.txt and dr2.txt. The parameters should obviously be extracted from the file. Please remember to close all the files.

7. Class StoreObjects

This class has access to the information in the other classes and stores every object in a separate file. The names of the files should follow the rules indicated above.

8. PS

To test that a file is empty you can use the following code:

```
if (myfile.peek() == std::ifstream::traits_type::eof())
{
	cout << "file is empty"<< endl;
	//do your processing

}
```

To know that a file does not exist you may use: `if(myFile.fail())` or `if(myFile.is_open.())`