Hospital Database

The Developer Documentation

Developer: Türker Erbasi
Neptun Code: NVT5RE

1. Introduction

Used C++ techniques: Inheritance, File management, Exception management.

Hospital Database program aims to make archiving in hospitals easier and simpler. To achieve this, it uses six operations. These are: Listing the patients, listing the doctors, registration of patients, registration of doctors, searching patients by name, searching doctors by name.

2. Main Function/Menu

Menu consists of seven operations which are listing patients, listing the doctors, registration of patients, registration of doctors, searching patients by name, searching doctors by name and quit the program. The menu section is basically a *switch/case* condition which presents options to the user to select one and run the selected part of the code. Options are listed one to seven. (example: If the user selects option "2" then condition will run the function for listing the doctors.) All these cases are in a *do/while* conditional loop to check if the user still wants to continue to program or not. This conditional loop checks a character every time, if it still satisfies the condition of the *do/while* loop. If the character is "y" prints the menu section again and asks the user to choose one of the options. If the character is "n", then ends the program.

3. Classes

Base class "Human" consists of three protected variables which is two *strings* (name and surname) and one *integer* (age) and one constructor. There are two more derived classes which are "Patient" class and "Doc" (Doctor) class. These classes have some additional members. "Patient" class have three more strings to hold entrance date, exit date and illness of the patient. "Doc" class have two more additional members which are *strings*. These *strings* are to hold the date of starting and date of retiring.

4. Methods

"printInfo ()" method is used for printing the patient's or doctor's information to the *"pList.txt"* or *"dList.txt"* text file. Common working way is opening an already existing file with a file object and writes the data which class is currently holding. There is a *try/catch* also, so if something goes wrong while opening the text file, function can throw an integer for an exception. So, function can let the user know if file couldn't open properly. Works almost the same way for both classes. Only differences are they are using different file objects and different files to write.

```
15  void Patient::printInfo()
16  {
17      fstream patientFile;
18      patientFile.open("pList.txt", fstream::out | fstream::app);
19      // opening file only for write at the end of the file.
20      try
21      {
22          if (patientFile.is_open()) // checks if the file is associated.
23          {
24              patientFile << Human::name << endl;
25              patientFile << Human::surname << endl;
26              patientFile << Human::age << endl;
27              patientFile << Patient::enter << endl;
28              patientFile << Patient::exit << endl;
29              patientFile << Patient::illness << endl;
30              patientFile << endl;
31          }
32          else
33              throw 30; // integer number for the error.
34      }
35      catch (int e)
36      {
37          cout << "ERROR: File Could Not Opened. Error no:" << e << endl;
38          // if the file is not associated with the target file sends an error.
39      }
40  }
```

5. Functions

5.1. patientList () and doctorList () Functions

"patientList ()" and "doctorList ()" functions basically print the whole archive to the console screen to let user see what patients or what doctors are registered to the database. Functions open the relevant files then in *try/catch* they check if the relevant file is open, if the relevant file is open then both functions run until end of file, line by line. While the functions running line by line, they print each line to the console with the help of a temporary *string.* Return types are *void.*

```
35  void doctorList() // reading from doctor file.
36  {
37      fstream doctorFile; // file object of doctors.
38      doctorFile.open("dList.txt", fstream::in); // opening file only for read.
39      try
40      {
41          if (doctorFile.is_open()) // checks if the file is associated.
42          {
43              cout << "\n\t->The Doctor List Shown Below" << endl;
44              string str; // string to hold lines of txt file.
45              while (!doctorFile.eof()) // reads until end of the file.
46              {
47                  getline(doctorFile, str); // gets the text line by line.
48                  cout << str << endl; // printing to the console.
49              }
50          }
51          else
52              throw 20; // integer number for the error.
53      }
54      catch (int a)
55      {
56          cout << "ERROR: File Could Not Opened. Error no:" << a << endl;
57          // if the file is not associated with the target file sends an error.
58      }
59      doctorFile.close(); // closing the file.
60  }
```

5.2. askPatient () and askDoc () Functions

"askPatient" and "askDoc" functions ask the user for an input to create the relevant objects. Functions use basic commands, they ask for name, surname, age then entrance date for a patient or starting date for a doctor. Second part of the functions are asking the user if the user want to enter further information about patient or doctor. If the user presses "y" character, functions ask the further information about patient or doctor which are exit date of a patient and illness of a patient or retiring date for a doctor. If the user presses "n" character, then functions take the already initialized strings. After all information given to functions, they call the relevant constructors. Functions return "Patient" or "Doc" type objects.

5.3. patientSearchName () and doctorSearchName () Functions

Search functions for searching by name in a text file. These two functions ask for a name from the user, save the name in a *string*. Then functions open patient's or doctor's text file, define an *integer* that equal zero for match count. In *try/catch*, functions check if the relevant text file opened properly or not. If the text file is opened properly then functions continue to code with saving each line and checking if any of the lines are equal to name saved string until the end of the text file. If two strings equal (name string and line in the text file), then prints six lines for patients and five lines for the doctors by the help of a *for* loop. If the file is not opened properly then *try block* throws an *integer* for *catch block* for an exception management. Then catch block prints an *error* message on console screen.