Università degli studi di Bergamo

SCUOLA DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Informatica

Relazione progetto C++

Corso di Programmazione Avanzata

Giulia Allievi
Matricola: 1058231

Anno Accademico 2021-2022

Introduzione

L'applicazione progettata permette di visualizzare e gestire le informazioni relative a delle visite mediche, mediante l'inserimento di alcune informazioni riguardanti l'esame, il paziente sul quale questo viene effettuato e il medico che lo esegue.

Funzionamento

L'applicazione è formata da 8 classi: Persona, Dottore, Paziente, Esame, Pet, Mr, PetMr e Metodi (una classe di gestione). Una persona può essere o un paziente o un dottore, entrambi ereditano dalla classe persona in modo pubblico. Entrambe le classi sono caratterizzate da un ID, la classe Dottore possiede un campo specializzazione mentre la classe Paziente un campo per memorizzare la categoria alla quale appartiene. La categoria può assumere uno fra quattro valori possibili, che sono bambino, adulto, anziano e sconosciuto. Le classi di Paziente e Dottore ereditano i campi di nome, cognome, codice fiscale e anno di nascita dalla classe Persona. Le operazioni che si possono svolgere con questi oggetti sono l'inserimento, la visualizzazione e l'ordinamento secondo alcuni criteri.
Gli esami possono essere di quattro diversi tipi: un esame "generico", Pet, Mr e Pet+Mr, che un esame in cui vengono effettuati in successione un esame Pet e un esame Mr. Ogni esame è caratterizzato da un numero progressivo che lo identifica, dal dottore che lo esegue e dal paziente che si sottopone a quell'esame. Un esame Pet possiede un campo per memorizzare la sua durata, così come per gli esami Mr, ma quest'ultimi hanno anche un campo aggiuntivo per memorizzare l'intensità. Gli esami di tipo Pet+Mr, dato che sono un "ibrido" dei due esami precedenti, possiederanno entrambi i campi.
La classe Metodi raccoglie invece tutte le funzionalità che che vengono offerte dal programma e che sono poi richiamate nel main del programma. Nel main è implementato il menù: l'utente digita la lettera corrispondente all'operazione che intende eseguire, sarà quindi richiamato il metodo corrispondente a quella determinata azione.

Diagramma delle classi

Di seguito si riportano i diagrammi delle classi dell'applicazione.
- Gerarchia degli utenti (classi Persona, Paziente e Dottore):

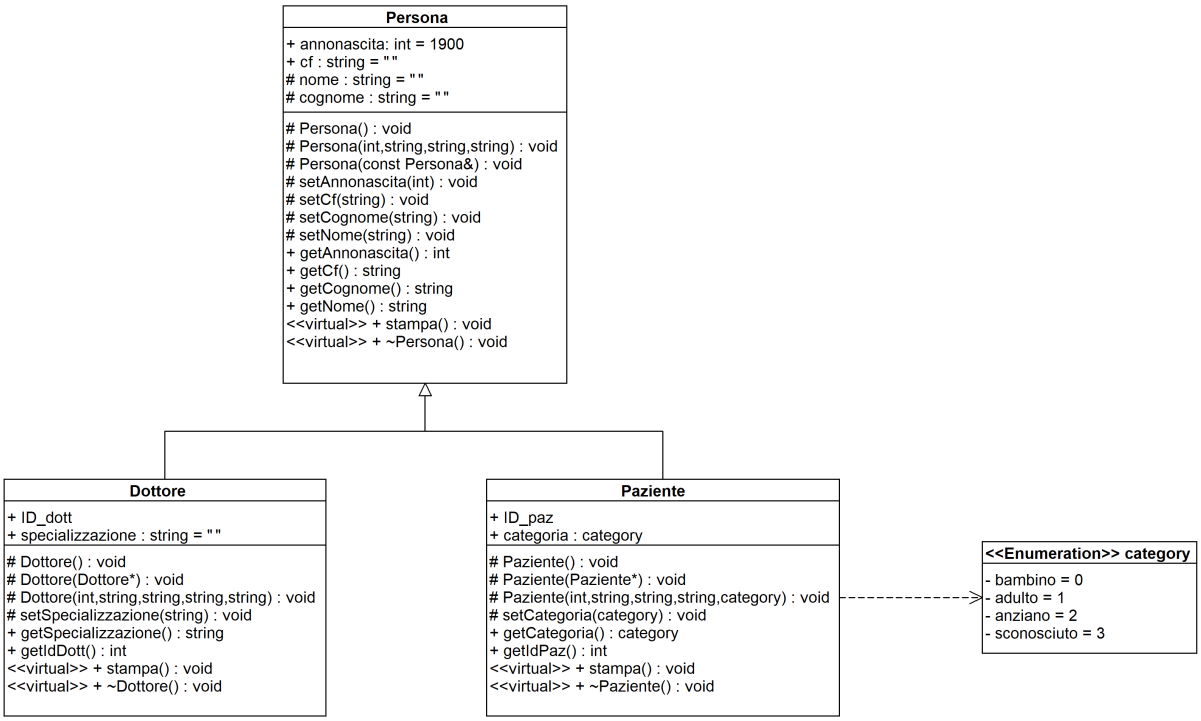

Figura 1: Diagramma delle classi - Gerarchia utenti.

- Gerarchia delle visite mediche (classi Esame, Pet, Mr e PetMr):

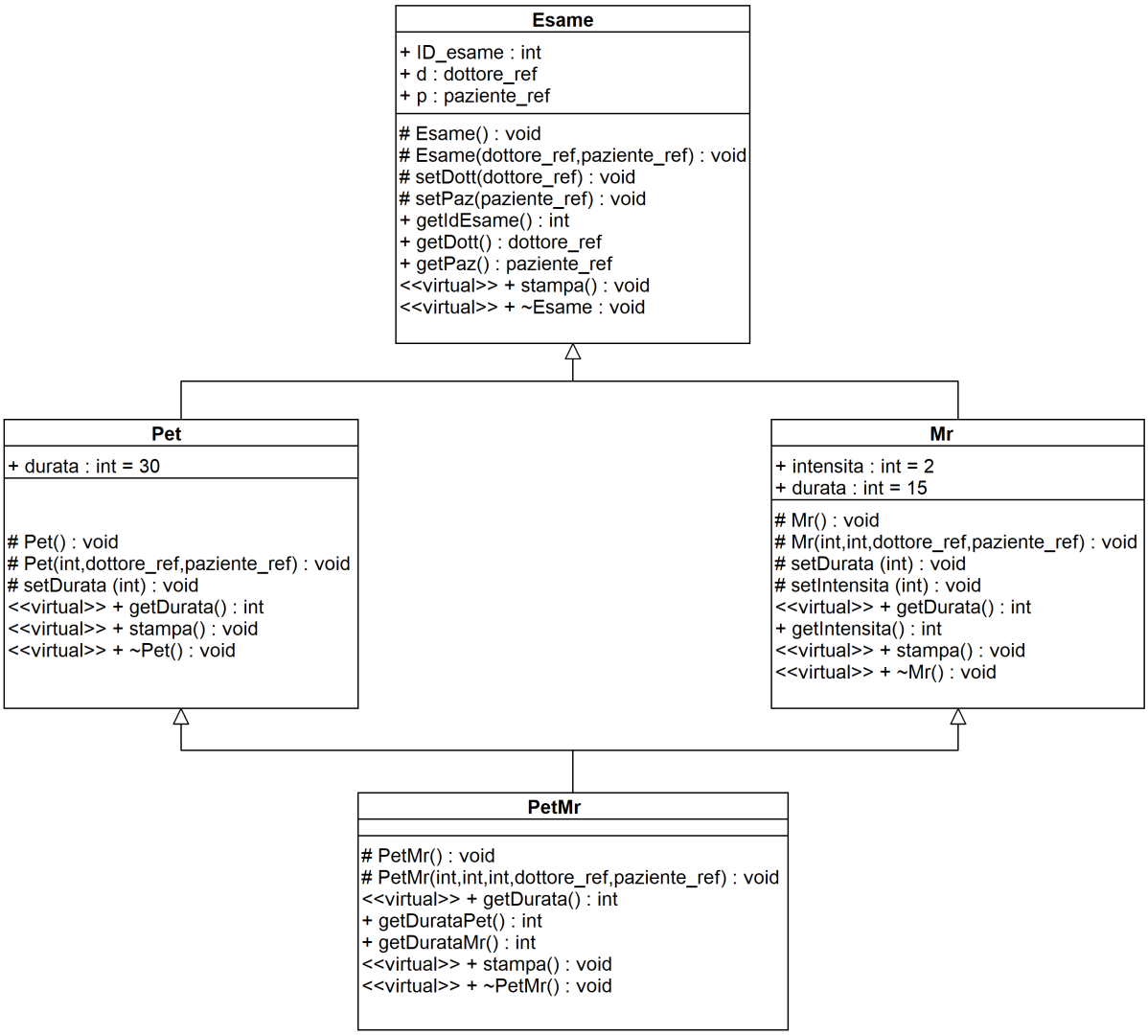

Figura 2: Diagramma delle classi - Gerarchia esami.

- Interfaccia dei metodi disponibili:

Figura 3: Diagramma delle classi - Metodi resi disponibili nel programma principale.

Scelte implementative

I costruttori sono stati dichiarati *protected*, in modo tale che possano essere richiamati solo dalle classi derivate. Tutte le classi di Utenti e Visite sono *friend* della classe Metodi, in modo tale che la classe di gestione possa accedere a tutti i campi e a tutti i metodi di cui necessita. In questa classe inoltre i metodi che si occupano della creazione di un nuovo oggetto sono stati dichiarati *private*, così che nel main non possano essere richiamati in modo illecito. Anche i metodi utilizzati per il controllo del valore delle variabili sono stati dichiarati *private*, mentre tutti gli altri metodi sono *public*.
La classe Metodi ha il costruttore privato perché mette a disposizione una singola istanza che verrà poi utilizzata nel main (design pattern *Singleton*).
Per ogni altra classe invece esiste più di un costruttore (*overloading* degli operatori): se non vengono passati i parametri vengono assegnati dei valori di default, in caso contrario si assegnano i parametri passati dall'utente. L'*overloading* viene anche sfruttato nella classe Metodi, nei metodi di creazione di un nuovo oggetto da inserire e nei metodi di controllo.
I metodi che si occupano della visualizzazione (stampa) sono stati dichiarati *virtual* per sfruttare il *dinamic binding*: in questo modo viene richiamato il metodo di stampa in base all'oggetto che esegue il metodo e non sulla base del tipo con cui questo oggetto viene dichiarato (a patto che si utilizzino dei puntatori, altrimenti il binding dimanico non verrà eseguito).
La gerarchia delle classi che descrivono gli esami forma un diamante, perché sia la classe Pet che la classe Mr hanno come classe base Esame, mentre la classe PetMr eredita sia dalla classe Pet che dalla classe Mr (mediante il meccanismo dell'*ereditarietà multipla* del C++). Per non avere problemi di name clash e duplicazione di campi nella classe derivata PetMr, le classi Pet e Mr ereditano in modo *virtual* dalla classe base Esame.
Al posto di utilizzare i puntatori tradizionali (*raw pointers*), sono stati utilizzati gli *smart pointers* della libreria memory. Questi puntatori si utilizzano come i puntatori raw, ma hanno il vantaggio che il programmatore non deve occuparsi della gestione della memoria come invece avviene per i puntatori tradizionali. Per semplificare il codice, per ogni classe viene dichiarato un tipo di dato che rappresenta uno *shared_pointer* così da utilizzare poi questo tipo di variabile nel programma.
Viene anche utilizzata la libreria STL, in particolare si utilizzano i *vector* come struttura dati per memorizzare la lista di ogni tipo di oggetto (Paziente, Dottore, Esami, Per, Mr e PetMr). Con i *vector* è possibile utilizzare gli *iterator* per scandirli. Gli *iterator* vengono utilizzati nel programma ogni volta che è necessario compiere una detrminata azione su tutti gli elementi della lista, come avviene per esempio nei metodi di stampa. I *vector*

possono anche essere ordinati tramite la funzione *sort()*, un algoritmo di ordinamento che richiede tre parametri (il primo e l'ultimo elemento da ordinare e un *comparator*, che indica come va effettuato l'ordinamento).
Per le stringhe si utilizza il tipo *string* messo a disposizione dall'omonima libreria, che permette di gestire le stringhe in modo facile e sicuro.