oggetto Relazione progetto Programmazione a Oggetti

gruppo Bresolin Gianluca, mat. 2034316

titolo Clinica Medica

Introduzione

Clinica Medica è un gestionale per studi medici che permette di creare, modificare, cancellare e visualizzare i pazienti che vengono accolti e gli operatori che vi lavorano al suo interno. Gli operatori che vi possono essere si distinguono in tre categorie: gli infermieri, i dottori e i primari, ciascuno in grado di svolgere un lavoro differente in base alle relative responsabilità che detiene.

Clinica Medica non presenta il solo scopo di gestire il carico di lavoro tra il personale in base ai pazienti ricevuti attraverso una visione complessiva della situazione attuale dello studio medico, ma fornisce inoltre una panoramica generale sullo stato di ciascun paziente, mantenendo per ogni persona che è stata visitata uno storico composto dall'intera documentazione delle visite: tale documentazione viene elaborata dagli operatori a partire dalla fase di accettazione fino a quella di dimissione.

Ho scelto questo progetto poiché ho pensato che le diverse tipologie di operatori potessero essere un buon pretesto per utilizzare il polimorfismo in maniera non banale, come richiesto nelle specifiche, attraverso l'esecuzione di lavori differenti a seconda del tipo di operatore selezionato.

Descrizione del modello

Il modello logico si suddivide in due parti: le gestione delle persone, che comprende sia i pazienti che gli operatori dello studio medico, e la parte che permette di gestire i vari lavori che vengono svolti durante la visita attraverso l'oggetto *Medical_history* che costituisce la cartella medica. Vi è infine la classe *File_manager* che si occupa della realizzazione della persistenza dei dati.

La gerarchia riguardante le persone parte appunto da una classe *Person* che rappresenta le informazioni comuni tra i pazienti e gli operatori, ovvero un nome, un cognome e una data di nascita, con i relativi metodi *get* e *set*.

Successivamente la gerarchia si sviluppa in due rami: il primo ramo è costituito dalla classe *Patient* mentre il secondo viene realizzato a partire dalla classe astratta *Worker.*

La classe *Patient* rappresenta un paziente vero e proprio e presenta quindi l'attributo aggiuntivo *codice_fiscale* assieme alla *lista_lettere_dimissioni* realizzata attraverso un container templetizzato *Container* implementato nativamente, come richiesto dalle specifiche. Quest'ultimo attributo permette di mantenere lo storico di tutte le documentazioni prodotte dalle visite eseguire dal paziente all'interno dello studio medico.
La classe *Patient* offre inoltre i relativi metodi *get* e *set* per gli attributi nuovi che presenta rispetto alla base *Person*, con la differenza che per la *lista_lettere_dimissioni* possiamo aggiungere una *std::string* e verificare se il container è vuoto attraverso il metodo *Empty_lettere_dimissioni*.

La classe *Worker* costituisce la radice della sotto-gerarchia che riguarda gli operatori: è una classe astratta in quanto presenta il metodo virtuale puro *work* che verrà implementato successivamente nella gerarchia.

Volutamente la classe non presenta gli *operator ==* e *!=* marcati *virtual* in quanto in questo modo non viene concesso l'inserimento di un operatore avente stesso nome, stesso cognome e stessa data di nascita ma tipo di operatore differente rispetto ad un operatore già inserito.
La classe *Worker* presenta inoltre un altro metodo virtuale *accept* che permette di accettare un visitor chiamato *WorkerVisitorInterface,* il quale verrà successivamente implementato nelle classi concrete della sotto-gerarchia degli operatori.

La sotto-gerarchia degli operatori si articola in tre tipi disponibili: *Nurse, Doctor* e *Primary*.

La classe *Nurse* costituisce gli/le infermieri/e che, in quanto tali, hanno il compito di svolgere la fase di accettazione che consiste nell'accogliere il paziente, stabilendo il suo codice di gravità che verrà affidato alla sua cartella clinica, il quale permetterà di inserirla nella corretta area all'interno del gestionale dello studio medico. Tale lavoro affinché venga svolto richiede una *std::string* come attributo aggiuntivo proprio di ciascun *Nurse*, permettendo la sua gestione attraverso i rispettivi metodi *set* e *get*.

La classe *Doctor* deriva come la classe *Nurse* pubblicamente e direttamente dalla classe *Worker* e costituisce i/le dottori/dottoresse: la mansione di questi operatori consiste nel definire i sintomi che il paziente presenta e le relativa diagnosi. Per svolgere questo tipo di lavoro la classe presenta gli attributi aggiuntivi *symptons* e *diagnosis* di tipo *std::string,* con i relativi metodi di *set.*

Come ultima classe nella gerarchia degli operatori troviamo *Primary* che rappresenta l'operatore primario/a: questo tipo di ruolo all'interno dello studio medico presenta la funzionalità ulteriore rispetto ad un medico di poter dimettere un paziente, ponendo termine alla sua cartella medica, in quanto detiene una posizione dotata di maggior responsabilità.
Un primario, come si può capire dalla descrizione precedente, è comunque in grado di poter svolgere una visita: per questo motivo la classe *Primary* deriva direttamente dalla classe *Doctor,* permettendoci grazie all'*operatore di scope* di effettuare eventualmente come lavoro la mansione di visita che viene svolta da un semplice dottore.
Per far ciò, *Primary* non richiede attributi aggiuntivi ma necessita semplicemente dell'*override* del metodo *work*.

La classe che costituisce un nodo centrale, permettendo la gestione di ogni ciclo di un paziente e lo svolgere del lavoro degli operatori all'interno dello studio medico, prende il nome di *Medical_hisotry*, rappresentate appunto la cartella medica di ciascun paziente.
In quanto ad una cartella medica corrisponde un paziente, il costruttore di *Medical_history* richiede come unico parametro un puntatore a *Patient,* inizializzando già la *std::string documentation* che viene man mano costruita lungo il ciclo di visita all'interno dello studio medico dai vari operatori che la prendono in gestione, costituendo al suo termine la lettera di dimissioni che verrà aggiunta allo storico del paziente associato.
Inizialmente la cartella medica non presenterà alcun operatore associato, motivo per il quale il costruttore inizializza l'attributo puntatore a *Worker* col valore *nullptr*: successivamente nelle varie fasi di vita della cartella all'interno dello studio medico però tale puntatore può essere assegnato all'operatore a cui viene affidato il paziente, attraverso il metodo *Set_woker*. Altri metodi associati a questo attributo sono il relativo metodo *get,* il metodo *Remove_worker* che permette di rimuovere l'operatore associato restituendone il puntatore ed infine il metodo *Do_work* che attua la chiamata al metodo polimorfo *work* sul puntatore di tipo *worker*.

Ciascuna *Medical_history* presenta un attributo di tipo *Patients_state,* una enumerazione che permette di tenere traccia della fase in cui la cartella medica si colloca all'interno del suo ciclo di vita, attraverso le seguenti fasi:

- *Pre_acceptance*
- *Acceptance*
- *Visit*
- *End_visit*
- *Discharge*
- *End*

Associati a tale attributi vi sono i relativi metodi *get* e *set* (anche attraverso l'uso di *std::string)* ed un metodo *Change_State* che permette di cambiare lo stato nell'ordine cronologico corretto.
Come ultimo attributo ogni *Medical_history* presenta un *cod* di tipo *Medical_code,* un altro tipo di enumerazione che permette di rappresentare la gravità della cartella medica in una scala crescente attraverso i valori:

- *Unsettled*
- *White*
- *Green*
- *Blue*
- *Orange*
- *Red*

Tale attributo viene assegnato dal *work* svolto dall'operatore di tipo *Nurse* attraverso il metodo *set* relativo che la classe *Medical_history* rende disponibile assieme al metodo *get* (anche in formato *std::string*).
Quando una cartella medica viene creata ad essa non è ancora stato assegnato alcun operatore e risulta essere nello stato *Pre_acceptance*, motivo per il quale il suo codice non può essere ancora stabilito: per far fronte a questa situazione è stato appunto creato il codice *Unsettled* che difatto costituisce un codice speciale, privo di un significato reale in termini di gravità ma che piuttosto contraddistingue le cartelle mediche che devono ancora essere prese in carico da un operatore dello studio medico.
La classe *Medical_hisotry* infine ridefinisce gli *operator* == e *!*= in modo tale da effettuare un controllo semplicemente sul paziente associato a ciascuna di esse: in questo modo si evita di creare più cartelle mediche per lo stesso paziente, che differiscono magari semplicemente per la fase in cui si trovano o per il codice che gli è stato assegnato.

Come richiesto dalle specifiche, in questo progetto è stata implementata nativamente una classe *Container*: questo tipo di container consiste in una lista linkata tramite l'utilizzo di puntatori alla classe *Item* che permette di immagazzinare una informazione templetizzata, rendendo a disposizione la possibilità di utilizzare un iteratore costante per scorrere gli elementi contenuti in essa.
Il container presenta il metodo *copy* e *destroy* per permettere rispettivamente la copia e la distruzione profonda.
Nel caso in cui il container contenga puntatori ad oggetti, attraverso il metodo *clear_no_garbage* può essere effettuata l'eliminazione di tutti gli oggetti che sono puntati, evitando in questo modo di lasciare garbage nella memoria nel caso in cui si provveda alla eliminazione profonda della lista (la quale comporterebbe solamente l'eliminazione dei puntatori).
Il container presenta inoltre il metodo *pop* per permettere una rimozione con politica 'first-in, first-out' utilizzata per eseguire l'accettazione delle cartelle mediche.

Altro scopo ha invece il metodo *remove*: questa funzione esegue una rimozione dell'elemento passato come parametro facendolo tornare come risultato e, in caso non sia presente, solleva una eccezione; questo metodo viene utilizzato per rimuovere l'operatore assegnato ad una cartella medica dalla lista degli operatori disponibili.
Il metodo *search* invece presenta la funzionalità di ricerca di un elemento passato come parametro, ritornando come risultato il puntatore a tale oggetto nel caso di ricerca positiva, altrimenti restituendo il valore *nullptr*. Questa capacità del contenitore viene utilizzata in più punti per verificare la presenza o meno di un particolare oggetto.

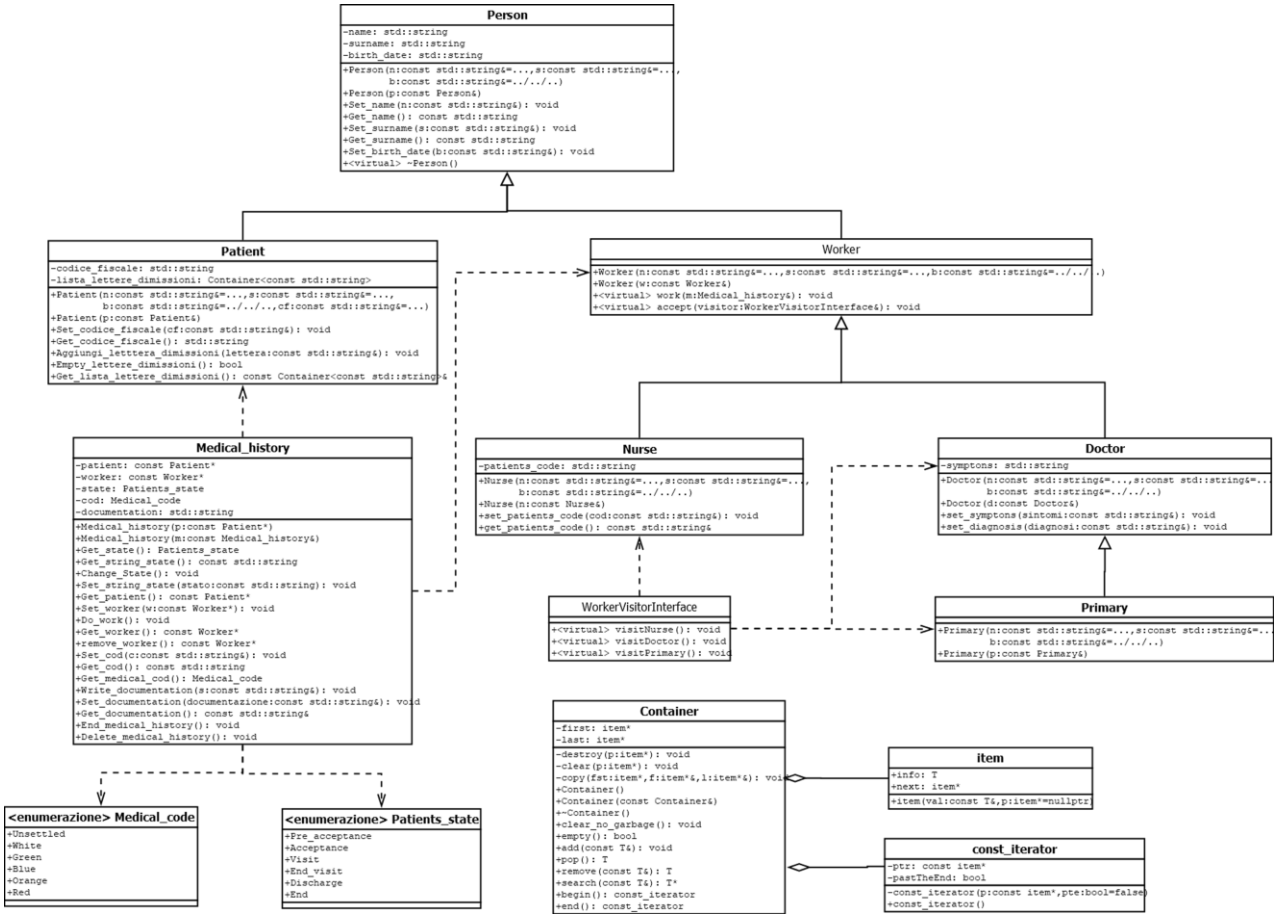

Figura 1: Diagramma delle classi del modello

Il progetto presenta infine una interfaccia *WorkerVisitorInterface* che fornisce i metodi astratti che verranno in seguito implementati dalla classe *WorkerInfoVisitor*: questo tipo di visitor presenta la funzione di restituire una *QLabel* contenente il tipo dell'operatore che lo accetta.

Polimorfismo

In questo progetto il polimorfismo viene utilizzato per svolgere lavori differenti sulla cartella medica in base al tipo di operatore a lei associata.
Per eseguire il lavoro, sulla *Medical_history* viene invocato il metodo *Do_work* che come precedentemente spiegato invoca il metodo virtuale *work* attraverso il puntatore alla classe astratta *Worker*.

A seconda del tipo dell'operatore associato i lavori che vengono svolti con la chiamata al metodo polimorfo *work* si differenziano in:

- nel caso in cui venga puntato un oggetto della classe *Nurse*, verrà inserito il codice di gravità della cartella medica precedentemente associato all'infermiere/a solamente nel caso in cui la *Medical_history* si trovi nel *Patients_state Pre_acceptance*. Una volta inserito tale codice, l'operatore eseguirà il metodo *Change_State* sulla cartella medica, portando così il nuovo stato a *Acceptance.* Il tutto viene infine scritto all'interno della *documentation*;
- nel caso in cui invece l'oggetto puntato sia della classe *Doctor* e la *Medical_history* si trovi nello stato *Visit*, l'operatore svolgerà una visita scrivendo nella *documentation* i sintomi con la relativa diagnosi, cambiando infine lo *state* in *End_visit* una volta terminata appunto la visita;
- nel caso in cui infine l'operatore puntato sia di tipo *Primary* e la cartella medica si collochi nello *state End_visit*, il metodo *work* provvede a terminare la scrittura della *documentation* e a settare lo stato al valore *Discharge.*

L'esecuzione del metodo *Do_work* appartenente alla classe *Medical_history* prevede un caso speciale in cui la chiamata polimorfa al metodo *work* viene a meno: nel caso in cui infatti l'operatore puntato sia di tipo *Primary* e la cartella si trovi nello *state Visit*, viene effettuata la chiamata al metodo preceduta dall'operatore di scope della classe *Doctor* (ovvero *Doctor::work*), permettendo così ad un/a primario/a di fare una visita, in quanto anche loro sono pur sempre dei dottori in grado di svolgere tale mansione.

Un altro uso di polimorfismo viene presentato nel metodo *accept* all'interno della sotto-gerarchia degli operatori: a seconda del tipo di operatore su cui viene invocato, il metodo permette di accettare un *WorkerInfoVisitor* (derivante pubblicamente dalla classe *WorkerVisitorInterface*) invocando il metodo rispettivo tra *visitNurse, visitDoctor* e *visitPrimary,* permettendo di settare il *QWidget** proprio della classe con la *QLabel** puntante alla label contenente il tipo di operatore.

Persistenza dei dati

Per la persistenza dei dati viene utilizzata la classe *File_manager* che lavora su un unico file per catalogare tutti i pazienti, gli operatori e le cartelle mediche aperte presenti all'interno del gestionale dello studio medico.

Gli oggetti *Patient* e tutti i tipi di operatori vengono scritti all'interno del file *SaveData.txt* in zone delimitate da opportuni indicatori quali ad esempio "*Patient:"* e "*END_Patient",* zone che appunto contengono l'elenco di tutti gli oggetti creati, allocando i valori dei loro attributi all'interno parentesi tonde '(' ')' e separandoli attraverso l'uso del carattere '|', motivo per il quale tali caratteri non vengono resi disponibili nelle fasi di creazione degli oggetti *Patient* e tutti gli oggetti della sotto-gerarchia *Worker*: per questi ultimi inoltre non è concesso un nome iniziante col carattere '0' (in quanto tale valore viene utilizzato per identificare il valore *nullptr* in una cartella medica alla quale non è ancora stato associato un operatore) mentre per definire il tipo di ciascun operatore vengono utilizzati i caratteri N per *Nurse*, *D* per *Doctor* e *P* per *Primary,* stabiliti attraverso l'uso del *dynamic_cast* applicato ad un puntatore di tipo *Worker*.

Anche le *Medical_hisotry* vengono memorizzate all'interno di una zona delimitata allo stesso modo delle classi precedenti ma la persistenza di uno di questi oggetti viene realizzato in modo più

complesso, presentando comunque l'elenco dei suoi attributi a cui viene posta in anticipo la lettera iniziale maiuscola del nome dell'attributo (ad esempio per l'attributo *patient* viene utilizzato il carattere '*P*'). Per gli attributi di tipo puntatore, ovvero per il *patient* e per il *worker,* vengono memorizzati gli oggetti a cui il puntatore punta con la tecnica medesima precedentemente menzionata.

La classe *File_manager* presenta:

- un metodo di scrittura del file che permette il salvataggio delle modifiche sugli oggetti presenti all'interno del gestionale (a cui si aggiunge un metodo specifico per la scrittura delle cartelle mediche) che viene invocato attraverso l'uso dei *signal* ogni qual volta risulta necessario
- Un metodo di lettura dal file che permette di restaurare lo stato del gestionale al momento dell'ultima chiusura dell'applicazione: tale metodo viene invocato sugli oggetti del *main.cpp* prima di mandare in esecuzione la *mainwindow*.

Funzionalità implementate

Le funzionalità implementate vengono presentate nella prima schermata di avvio dell'applicazione: queste si suddividono in creazione di oggetti resi disponibili e visione di liste di tali oggetti creati.

Gli oggetti che possono essere creati sono:

- *Patient*: attraverso il pulsante '*inserisci paziente*' si aprirà una schermata di creazione di un paziente. Questo paziente dovrà differire dai pazienti già precedentemente creati e dovremmo fornire suo nome, cognome, data di nascita e codice fiscale;
- *Worker*: attraverso il pulsante '*inserisci operatore'* si aprirà una schermata di creazione di un operatore che, a seconda del tipo selezionato tra i tre pulsanti resi disponibili, potrà essere un *Nurse* (tipo che viene inizialmente selezionato per default) o un *Doctor* o un *Primary*. Anche in questo caso l'operatore dovrà differire da tutti gli altri operatori precedentemente creati, solamente in base al nome, cognome e data di nascita (senza considerare dunque il tipo di operatore, ovvero non potremmo avere due operatori omonimi con ruoli però differenti);
- *Medical_history*: attraverso il pulsante '*Creazione Cartella Medica'* si aprirà una schermata che ci permetterà di cercare un paziente tra quelli inseriti all'interno del gestionale dello studio medico e che, in caso di riscontro positivo, porta alla creazione di una cartella medica associata a tale paziente, con codice *Unsettled* in stato *Pre_acceptance.*

Le liste di visualizzazione degli oggetti creati invece si suddividono in:

- *Elenco pazienti:* l'elenco dei pazienti creati, associando ad ognuno i valori dei propri attributi, presentano infine le funzionalità di modifica, di eliminazione e un pulsante che permette di visualizzare l'*Elenco delle lettere di dimissione* che a sua volta permette di visualizzare la documentazione vera e propria;
- *Elenco operatori disponibili*: questo elenco non costituisce l'elenco di tutti gli operatori presenti nello studio medico bensì l'elenco di tutti quelli operatori che non risultano occupati con nessun paziente in quel momento (gli operatori occupati sono visibili all'interno del prossimo elenco, ovvero l'elenco delle cartelle mediche). Anche in questo elenco, come nel precedente, sono presenti gli attributi di ciascun operatore assieme ai

pulsanti che permettono la loro modifica ed eventuale eliminazione, preceduti dalla etichetta ottenuta tramite il *WorkerInfoVisitor* che ci indica il tipo di ciascun operatore (infermieri, dottori e primari vengono infatti visualizzati all'interno dello stesso elenco);

- *Elenco cartelle mediche*: l'ultimo elenco è costituito dall'elenco di tutte le cartelle mediche attualmente attive all'interno dello studio medico. Questa costituisce una schermata particolarmente importante in quanto permette di gestire tutte le attività che vengono svolte all'interno dello studio, attraverso un elenco completo e distinto di tutte le cartelle mediche in base al codice di gravità che gli è stato associato. Per ciascuna cartella viene indicato il codice fiscale del paziente associato, il codice di gravità, lo stato attuale della cartella medica, il tipo di operatore associato e il cognome di quest'ultimo. Le cartelle sotto la voce '*Unsettled'* vengono prese in gestione in base all'ordine di creazione, seguendo una politica 'first-in, first-out', attraverso il pulsante '*Esegui accettazione'* che permette di far sì, sempre che vi sia un operatore di tipo *Nurse* libero, di procedere con l'inserimento della cartella medica all'interno dell'elenco apposito in base al codice di gravità selezionato in una schermata che vi si aprirà non appena premuto il pulsante precedentemente menzionato (di default viene selezionato il codice "*White")*. Ciascuna cartella che presenta un codice di gravità valido (ovvero differente da quello speciale '*Unsettled'*) possiederà al termine del suo spazio di rappresentazione un pulsante con la relativa azione che si deve svolgere per far sì che la cartella medica proceda nel suo ciclo di vita. Tali azioni si suddividono in:
    - *Visita*: porta all'apertura di una schermata (fermo restando che vi sia un *Doctor* disponibile, *Primary* compreso) all'interno della quale dobbiamo inserire i sintomi e la diagnosi associata a tale paziente
    - *Dimissioni:* sempre considerando che vi sia *Primary* disponibile, conclude la stesura della documentazione della cartella medica, portandola al suo stato finale.
    - *Terminazione:* rimuove la cartella medica dall'elenco a cui era associata, eliminando la cartella medica ed inserendo la lettera delle dimissioni nell'elenco del paziente correlato.

Rendicontazione ore

Attività	Ore Previste	Ore Effettive
Studio e progettazione	10	9
Sviluppo del codice del modello	10	10
Studio del framework Qt	10	9
Sviluppo del codice della GUI	10	14
Test e debug	5	8
Stesura della relazione	5	5
totale	50	**55**

Il monte ore è stato leggermente superato in quanto sviluppo del codice di modello e GUI ha richiesto più tempo di quanto previsto, in quanto questo progetto costituisce la mia prima esperienza in un lavoro che prevedesse un'interfaccia grafica, in particolare col framework Qt.

Differenze rispetto alla consegna precedente