Programmazione ad Oggetti, a.a. 2019/2020

Relazione Progetto

Odinotte Marco

Gruppo di lavoro:

Abdelwahad Kandoul 1120176

Marco Odinotte 1170564

Elia Vettoretto 1125262

Sommario

1 Abstract

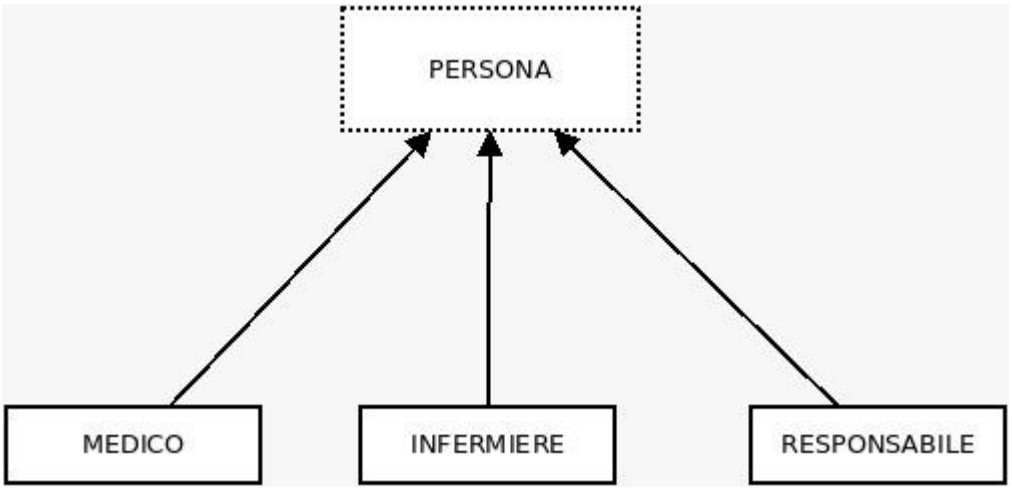
Considerando la realtà ipotetica di un piccolo ospedale, il progetto si pone lo scopo di gestire il personale di tale struttura e organizzarne i turni settimanali. I vari utenti, in base al ruolo professionale all'interno della struttura, hanno livelli di accesso diversi all'applicazione e alle informazioni che vi sono contenute. Esistono tre tipologie di utenti: medici, infermieri e responsabili. I responsabili sono gli unici utenti con il massimo livello di accesso; considerando il loro compito di aggiungere, eliminare e modificare utenti e turni; mentre medici ed infermieri potranno visualizzare tali informazioni. Essi, infatti, potranno solamente visualizzare i turni e le informazioni di tutti gli altri dipendenti, ad esclusione del salario settimanale che è accessibile unicamente ai responsabili e all'utente titolare. I turni creati sono esclusivamente giornalieri (non si hanno, infatti, più turni al giorno) e si differenziano in quattro tipologie in base alle proprie caratteristiche: turno intero, turno parziale, turno libero e turno straordinario. Mentre i primi tre sono facilmente intuibili, il turno straordinario è una particolare eccezione al turno libero; dato che viene creato solamente in seguito alla modifica di un giorno libero in un giorno lavorativo.

La funzione di ricerca permette, infine, di scorrere agevolmente la lista degli utenti e ricercare quello d' interesse; filtrando inoltre i risultati trovati, in base a criteri sulla tipologia di utente cercato.

2 Descrizione delle gerarchie utilizzate

Le gerarchie utilizzate sono quattro: due nel modello, che rappresentano gli utenti (Persona) e i turni (Turno), e due nella parte grafica che permettono di visualizzare e modificare turni e utenti.

2.1 Gerarchia delle persone

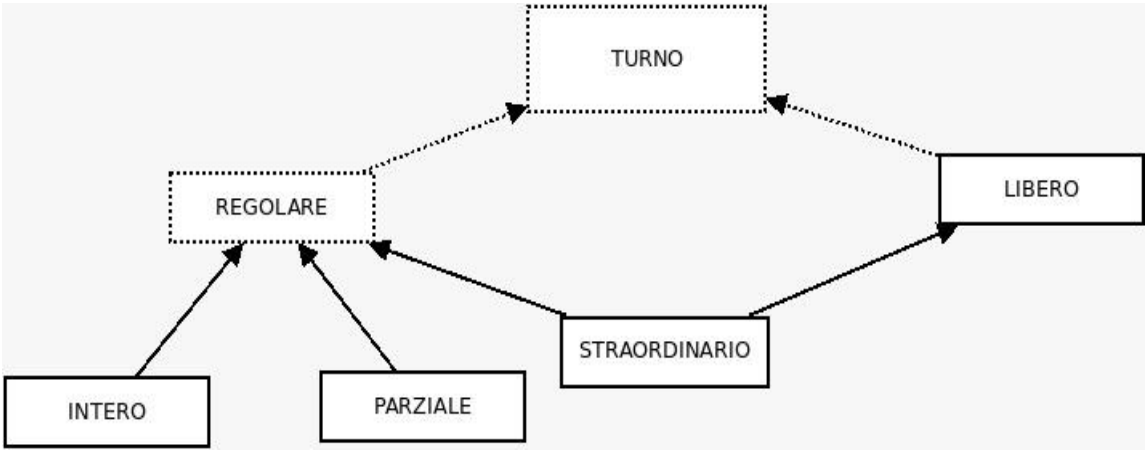


Tale gerarchia rappresenta gli utenti che avranno accesso all'applicazione, definendone i dati personali e i relativi turni, che verranno poi inseriti nel contenitore Queue richiesto dalle specifiche. La gerarchia nasce dalla classe base astratta Persona contenente i dati anagrafici principali di tutti gli utenti (username, password, nome, cognome, data di nascita, genere e la Queue dei turni). Da Persona derivano le tre categorie principali di utenti. Medico eredita i campi privati di Persona aggiungendo ad essi la specializzazione del medico e un booleano che indica se il medico in questione è un chirurgo o meno. Infermiere e Responsabile, invece, mantengono semplicemente i campi ereditati dalla base.

In ogni classe di tale gerarchia sono presenti due metodi virtuali:

- *infoPersona()* : restituisce una stringa contenente tutte le informazioni riguardanti l'utente in questione
- *stipendio()* : ritorna il valore del salario settimanale calcolato sulla base dei turni indicati nella Queue "turni".

2.2 Gerarchia dei turni



Questa seconda gerarchia rappresenta le varie tipologie di turni che possiamo rappresentare ed utilizzare all'interno dell'applicazione. La classe base astratta è Turno; che conterrà una variabile di tipo *giorno* (enum appositamente creato per rappresentare più agevolmente i giorni della settimana) e due variabili di tipo QTime che rappresenteranno gli orari di inizio e di fine del turno. L'orario di fine non va ad inficiare la scelta di mantenere un turno al giorno, dato che viene preso il giorno di inizio come riferimento.

Anche questa gerarchia contiene dei metodi virtuali:

- *infoTurno()* : come in Persona si occupa di ritornare le informazioni principali del turno
- *getColor()* : ritorna una classe che esprime il colore simbolico associato a tale tipo di turno

- *TotOre()* : calcola l'ammontare delle ore per il turno in esame; calcolate in base alle ore di inizio e fine. Tale calcolo è volutamente approssimato a quarti di ora, in modo da rendere più schematica ed organizzata la gestione dei turni.
- *paga()* : ritorna il salario calcolato sulla base del singolo turno in esame e ad alcuni parametri inseriti da un Responsabile

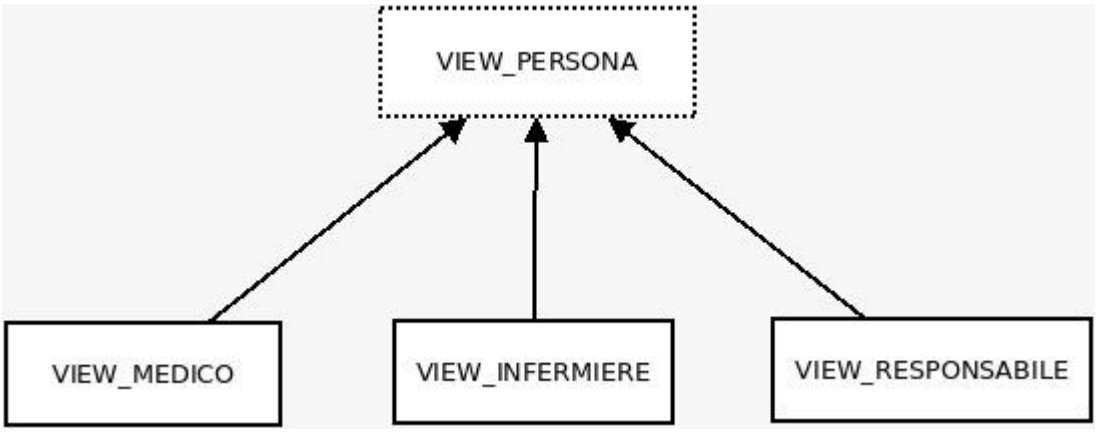
Da turno deriva direttamente Turno_libero; variante di Turno che si occupa di rappresentare i giorni liberi spettanti ai dipendenti. Essa è caratterizzata da un booleano *permesso* per indicare se tale giorno libero è previsto a cadenza settimanale oppure è solamente un giorno libero extra. Qui *oraInizio* e *oraFine* sono impostati in modo fisso dato che la giornata libera non ha vincoli di orario.

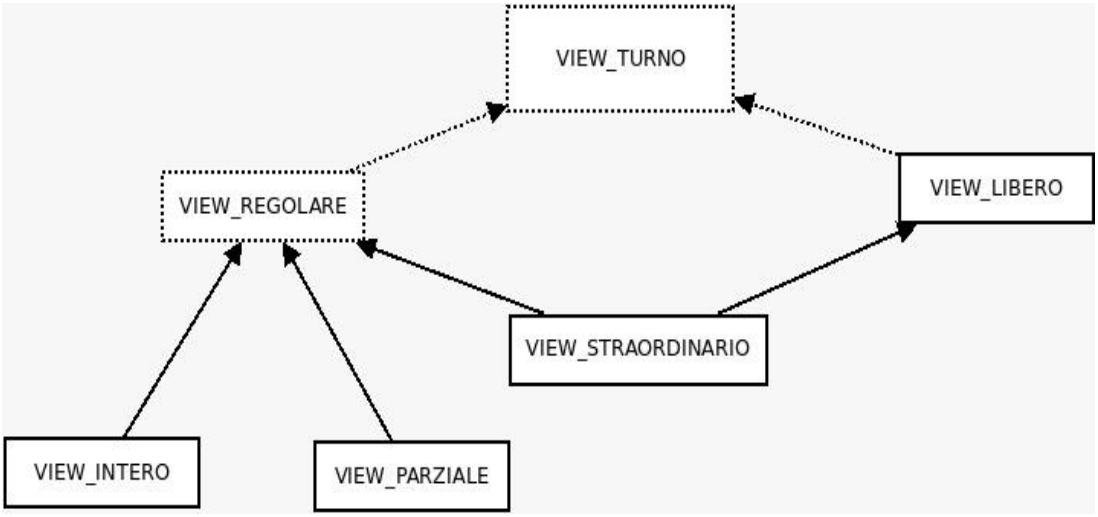
Sempre da Turno deriva anche Turno_regolare, una classe astratta che rappresenta i turni lavorativi di ogni dipendente o responsabile. Essa possiede il campo *reparto per* esplicitare il reparto dove tale turno si svolgerà.

Turno_regolare funge da classe base astratta per Turno_intero e Turno_parziale, due classi derivate che permettono di esprimere i turni interi o parziali che un utente potrebbe avere settimanalmente. Esse ereditano i campi privati da Turno e Turno_regolare, aggiungendovi i campi *paga_intero* e *paga_parziale* che andranno a contribuire al calcolo della paga giornaliera e, successivamente, dello stipendio settimanale.

Turno_straordinario, come accennato prima, è una variante di Turno_libero in cui un giorno considerato "libero" viene reso giorno lavorativo. Per tale motivo Turno_straordinario deriva sia da Turno_regolare che da Turno_libero, ed è, di conseguenza, generabile unicamente da un Turno_libero già preesistente.

2.3 Gerarchie delle viste





Le gerarchie delle viste, direttamente derivate da QWidget, sono la rappresentazione grafica del modello appena descritto. Esse si occupano di modificare e generare nuovi utenti o turni tramite la GUI adatta. Esse sono anche responsabili del controllo dei dati inseriti nei campi in modo da assicurare coerenza e integrità agli oggetti creati.

2.4 Classe contenitore

I contenitore utilizzato Queue costituisce una piccola gerarchia per, appunto, contenere i dati utili all'applicazione e gestirli tramiti appositi, metodi, iteratori (Base_iterator) ed operatori. I contenitore è costituito da una lista doppiamente linkata utile per rappresentare sia la lista degli utenti, sia la sequenza di turni settimanali che ogni utente possiede. Queue è, infatti, una classe templatizzata da cui derivano QueuePersone e QueueTurni, i due contenitori adatti a svolgere le operazioni appena elencate: mentre QueuePersone memorizza dei puntatori Persona* , QueueTurni immagazzina dei puntatori Turno* e, come da progetto, ha una dimensione di sette nodi come i giorni da rappresentare.

3 Uso del polimorfismo

3.1 Polimorfismo nella gerarchia persone

In tale gerarchia il polimorfismo è utilizzato innanzitutto dal distruttore, reso virtuale in modo da evitare qualsiasi memory leak. Successivamente è stato introdotto il metodo super polimorfo *clone()* per rafforzare ulteriormente il concetto di polimorfismo (utilizzato sia in QueuePersone.cpp che in QueueTurni.cpp). Altri metodi resi super polimorfi sono *isResponsabile()*, *canEditTurni()*, *canAddTurni()* che ritornano le informazioni di accesso per ogni utente.

3.2 Polimorfismo nella gerarchia turni

Come nella gerarchia delle persone, anche qui il distruttore e il metodo *clone()* sono resi virtuali, in modo da agevolare la distruzione dei turni nella Queue e la clonazione degli oggetti di tipo Turno. Anche *paga()* è resa polimorfa in modo da comprendere nel calcolo qualsiasi derivato di Turno, oltre a tutti i parametri compresi nelle classi. Infine *exportXmlData(QXmlStreamWriter&)* è un altro metodo polimorfo che permette di facilitare l'export dei dati dell'applicazione in un file di tipo XML.

3.3 Polimorfismo nella gerarchia delle viste

Oltre al distruttore, reso virtuale come nelle altre gerarchie, le viste comprendono dei metodi particolari come check(), edit() e build_field(), resi virtuali in modo che possa costruire la vista per modificare i singoli turni oppure gli utenti. Viene eseguito, inoltre un controllo dei campi inseriti nella creazione e modifica dei dati.

4 Utilizzo I/O

Come già accennato, l'applicazione permette di scrivere e leggere dati da file di tipo XML. Tale scelta è stata applicata per poter gestire in modo facile le dinamiche di I/O e utilizzare i meccanismi di parsing già esistenti all'interno delle libreria Qt. QXmlReader e QXmlWriter sono le classi utilizzate in tale processo e, in particolar modo permettono di rappresentare i dati attraverso dei Tag che rendono l'organizzazione molto più salda. Se non è presente un file dei dati, il programma, all'avvio, segnalerà tale mancanza e ne genererà uno apposito.

5 Manuale utente

L'applicazione farà uso del file dei dati presente nella cartella per accedere alle informazioni necessarie al suo funzionamento. Se tale file non fosse presente, verrà cerato un file contenente un utente Responsabile (Username: admin, Password: admin) di default per permettere il primo accesso. Una volta acceduti troviamo a sinistra l'elenco del personale registrato nell'applicazione, compreso l'utente che ha effettuato il login, mentre a destra compaiono le informazioni dell'utente selezionato nell'elenco. Nella parte sottostante troviamo i pulsanti per l'aggiunta, modifica ed eliminazione di utenti, il calcolo dello stipendio ed il riordinamento degli utenti secondo l'ordine alfabetico. I primi tre pulsanti sono disponibili sono agli utenti di tipo Responsabile, ad eccezione del tasto *Elimina utente* che è utilizzabile da un responsabile verso tutti gli altri utenti (compresi altri responsabili) al di fuori di sé stesso. *Calcola stipendio* invece è attivo solo per i responsabili o per gli utenti che controllano il proprio stipendio. Infine troviamo la serie di pulsanti dei turni, uno per ogni giorno della settimana, con le informazioni relative al singolo turno. Cliccando un turno si accederà ad una finestra per selezionare il tipo

di turno: se il tipo di turno selezionato coincide con il tipo di turno preesistente, verrà innescata una modifica del turno che importerà nella vista i dati del turno preesistente. Altrimenti, se il tipo non coincide, tale turno verrà cambiato di tipo in base a quello selezionato ed i dati presenti saranno dati di default.

6 Suddivisione dei ruoli e tempistiche

I ruoli nel gruppo sono stati suddivisi nel seguente modo:

- Abdelwahad Kandoul : finestra e gestione del login, GUI, debugging e testing
- Marco Odinotte: progettazione e sviluppo gerarchie, GUI, debugging e testing
- Elia Vettoretto: realizzazione contenitore Queue, GUI, debugging e testing

Mentre le tempistiche impiegate rispettano le seguenti cadenze:

- Progettazione modello: 2 ore
- Progettazione parte grafica: 5 ore
- Implementazione modello: 10 ore
- Implementazione parte grafica: 20 ore
- Apprendimento libreria Qt: 8 ore
- Debugging e testing: 5 ore

7 Ambiente di sviluppo

Lo sviluppo è stato effettuato principalmente su:

- Sistema operativo: Windows 10 Version 2004 [versione 10.0.19041]
- Compilatore: gcc 7.3.0
- Versione: Qt 5.14.1

Il programma compila correttamente anche nella macchina virtuale fornita. I passaggi per la compilazione corretta sono stati:

- sudo apt get-install qt5-default
- riavvio pc

E per la corretta compilazione:

- qmake –project “QT +=widgets”
- qmake
- make

All’interno della cartella esempi è fornito un file XML di esempio (persone data.xml) che va inserito tra i file oggetto del programma.