Programmazione ad oggetti

Progetto "**gestione impianti a gas - GasEquipment**"

Relazione di **Ballarin Simone**

Matricola **1122286**

Informatica - Università di Padova

Anno 2016/2017

Contenuti

Utilizzare il file .pro consegnato

1 - Scopo del progetto

Il progetto si prefigge lo scopo di creare un programma al fine di facilitare la gestione di un'officina che si occupa di installazioni di impianti GPL, la quale deve trattare insieme di Veicoli e relativi impianti installati in essi (se presenti), l'accesso al database di veicoli è condizionato dal tipo di Utente che si è. Si può essere utenti di tipo Operaio o utenti di tipo Capo. I primi possono visualizzare i Veicoli presenti, filtrarli in base a criteri di ricerca, i secondi oltre a queste possibilità possono in aggiunta inserire nuovi veicoli, eliminarli, modificarli e gestire gli altri utenti (con possibilità di cambiarli nome, passworld, tipo, cancellarli o crearli).

I Veicoli possono eventualmente possedere un impianto in questo caso vengono chiamati Installazioni. Di un Veicolo è importante sapere i dettagli sul motore, sull'impianto di alimentazione, e sullo scarico. L'Impianto è un array di bombole. Gli impianti possono essere di vari tipo MaticGPL, DirectGPL, OmegaGPL e NoCatGPL (ognuno di essi richiede determinate caratteristiche al Veicolo per poter essere installato). Al momento sono tutti impianti a Gpl, ma non si esclude la possibilità di dover gestire impianti Metano in futuro. L'Officina inoltre ha la necessità di conoscere se ci sono Installazioni con bombole che necessitano di revisione (ogni 4 anni).

1.1 – Requisiti di impianto

Tutti gli impianti GPL possono essere installati solo su auto con cilindrata compresa in un certo intervallo:

Minima cilindrata per GPL motore aspirato=890;

Massima cilindrata per GPL motore aspirato=5500;

Minima cilindrata per GPL motore sovralimentato=1335;

Massima cilindrata per GPL motore sovralimentato=2250.

Inoltre sono previsti altri limiti in base al modello di impianto:

DirectGPL: -Iniezione diretta.

OmegaGPL: -iniezione di tipo elettronica

-presenza di catalizzatore

-presenza di sonda lambda

NoCatGPL: -iniezione elettronica

-no catalizzatore

MaticGPL: -carburatore

2 - Descrizione delle gerarchie di tipi usate

Il progetto presenta diverse gerarchie, cinque nella parte logica che rappresentano i componenti di un Veicolo, gli Utenti, una gerarchia per i Veicolo, una gerarchia di errori, infine una per gli Impianti. Nella parte grafica per ogni gerarchia del modello è presente una corrispondente gerarchia di Widget per creare i vari oggetti una per Visualizzarli, inoltre ci sono alcune gerarchie di finestre per implementare le differenze di possibilità tra i vari tipi di account. Per quanto riguardano le gerarchie nella parte logica si `e voluto utilizzare il più possibile la STL (tutto ad eccezione delle classi per la gestione dei file XML), in modo da non dover dipendere dal framework QT.

2.1 Gerarchia Componente

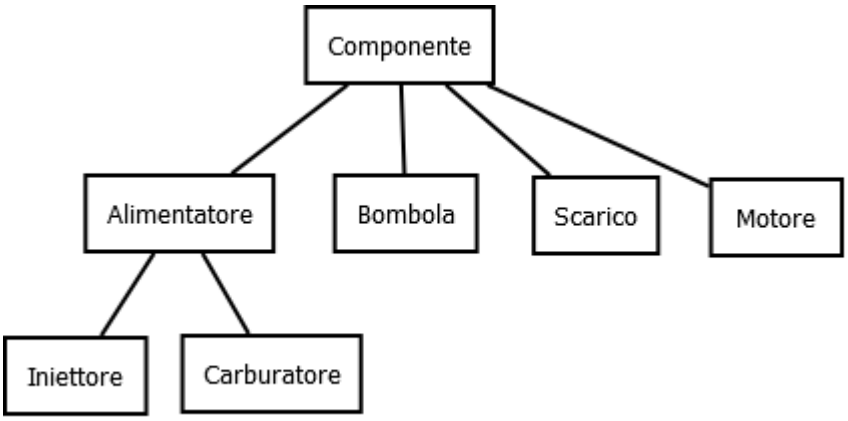

Figure 2: Gerarchia dei Componenti

La prima gerarchia costituisce l'insiemi di Componenti che caratterizzano un Veicolo, di cui ci interessa conoscerne i dettagli.

Si parte da una generica classe Componente astratta, con campi nomeProduttore e dataProduzione, e una serie di metodi utile alla gestione di questo oggetto. Da Componete di derivano poi le varie classi Alimentatore, Bombola, Motore, Scarico. Ad eccezione di Alimentatore tutte le classi sono concrete.

Alimentatore non introduce né campi dati, né metodi, serve solo al fine della gerarchia, questa classe rappresenta un generico sistema di alimentazione di un classico motore. In seguito la classe Alimentatore si concretizza con le classi Iniettore e Carburatore. Il primo rappresenta un iniettore, esso può essere diretto o indiretto e meccanico, misto o elettrico. Queste caratteristiche vengono esplicitate tramite campi dati. Il Carburatore invece non introduce nuove informazioni a parte quella di essere un carburatore (cosa che per gli scopi dell'officina è più che sufficiente, ma in caso la classe è pronta per future espansioni).

La Classe Motore ha il compito di descrivere un motore a combustione infatti ne rappresenta le caratteristiche più interessanti quali : la cilindrata, n di cilindri, rapporto di compressione, coppia massima e se è turbo aspirato o meno. La classe Scarico descrive uno scarico di un veicolo esso può opzionalmente avere una sonda lambda o un catalizzatore.

Bombola rappresenta una bombola di un impianto di essa è importante conoscerne la capacità e la data di ultima revisione (in quanto ogni 4 anni è necessaria una revisione).

2.2 Gerarchia Utente

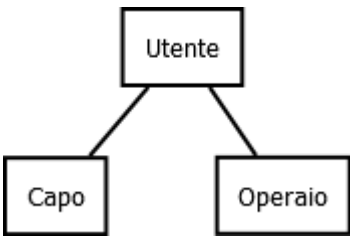

Figure 3: Gerarchia degli utenti

La gerarchia utenti consiste in una classe astratta Utente, resa astratta dai metodi di

clonazione , dal distruttore, dalla quale derivano due classi concrete. E' presente un metodo che consentono l'esportazione delle informazioni su file sono SalvaInfoUtente() virtuale e getTipo(), la prima salva le informazioni su file mentre la seconda, virtuale, fornisce per ogni classe l'informazione sul tipo. La classe astratta Utente rappresenta un utente del

programma, di esso ci interessa conoscere solamente username e password per gestire l'accesso al database. Le due derivazioni servono solamente per discernere il tipo di utente e quindi capire le relative possibilità di azione all'interno del programma.
Le classi concrete inoltre hanno un metodo di nome LeggiUtente per estrapolare i dati degli utenti da file questi metodi sono statici.

2.3 Gerarchia Impianto

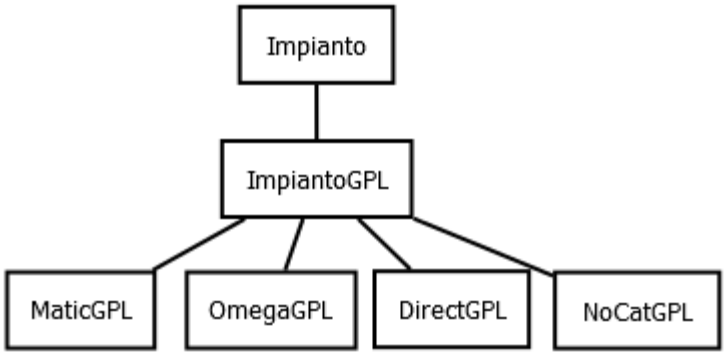

Figure 4: Gerarchia Impianto

Alla base della gerarchia è presente la classe astratta Impianto si tratta di una classe contenitore. Rappresenta un insieme di bombole, per fare ciò si è deciso di utilizzare un contenitore std::vector di puntatori a Bombola. La gestione della memoria è profondo a tal fine si è definito dei nuovi costruttore di copia, distruttore e operatore di uguaglianza. Inoltre sono presenti dei metodi virtuali getInfo(), getNome(), Salva() e idoneita() (non implementato). Il primo restituisce le info dell'impianto in base al tipo dinamico che presenta comunque una parziale implementazione nella classe astratta. Il secondo restituisce il Nome dell'impianto cioè il nome commerciale dell'impianto che sono le classi concrete della gerarchia.
L'ultimo è un metodo per la scrittura sul file, esso è parzialmente definito nella classe astratta.
Inoltre la classe fornisce i metodi statici Leggi e LeggiImpiantoInstanziabile. La prima legge i dati dell'impianto e siccome non può creare un oggetto Impianto (in quanto la classe è virtuale) restituisce un vector di puntatori a Bombola creati nello heap. La seconda invece serve a capire dai tag nel file xml di che tipo di impianto si tratta e chiamare la funzione Leggi() ,che è presente in ogni classe concreta di impianto, adatta. Inoltre la classe fornisce una serie di metodi che restituiscono informazioni sull'insieme di bombole che sono NBombole, capacita_impianto, AlertBombola (scorre il vector per controllare se c'è almeno una bombola che necessità di revisione) e un metodi per aggiungere una bombola all'impianto che è inserisciBombola. È stato anche inserito un operatore di subscripting che ritorna per riferimento le bombole.
Da Impianto deriva la classe ImpiantoGPL che rimane comunque astratta e che non apporta particolari aggiunte alla classe padre tranne per il fatto che introduce il metodo idoneità che ha il compito di verifica se l'impianto un veicolo è ha i requisiti legali (compito delegato a un metodo della classe veicolo) per l'installazione di un impianto qualsiasi GPL. Inoltre la classe fornisce implementazioni dei metodi Salva e Leggi. L'obbiettivo di questo classe è quello di garantire un elevato grado di espandibilità in previsione di dover gestire anche impianti a Metano.
La Gerarchia si conclude con Le classi concrete OmegaGPL, MaticGPL, NoCatGPL, DirectGPL. Le classi sono praticamente identiche differiscono solamente per la diversa implementazione di idoneita.

2.4 Gerarchi Err

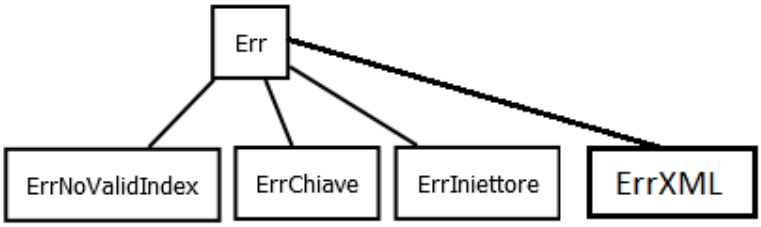

Figure 5: Gerarchia Err

E' presente anche una semplice gerarchia di classe di Errore che ha come inizio la classe Err da cui poi derivano le classi errnoiniettore, errnovalidindex, errchiave che rappresentano diversi tipi di errore.

Tutte le classi sono concrete. La classe Err ha come campo dati una stringa allo scopo di contenere info sull'errore occorso. Il campo è privato, si può scrivere solo attraverso il costruttore Err(std::string) . Inoltre è presente una getInfo che ritorna per valore questa stringa. Le classi che vengono derivate non introducono niente servono solo a scopi di usarle nelle throw e catch.

2.5 Gerarchia Veicolo

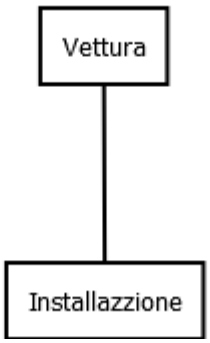

Figure 6: Gerarchia Veicolo

La gerarchia di Veicolo è composta da sole due classi, entrambe instanziabili. La prima è Veicolo

Che rappresenta il cuore dell'attivà dell'officina. Un Veicolo è sostanzialmente un Motore, un Alimentatore e uno Scarico con l'aggiunti di altri dati che sono proprietario, targa, data ultima revisione e data di immatricolazione. Per fare ciò si è scelto di usare tre puntatori verso lo heap di Motore, Alimentatore e Scarico (relazione has-a). Si è deciso di Fornire alla classe metodi di get e di set non solo per i propri campi dati ma anche per i dati del Motore, Alimentatore e Scarico. Siccome non si sà il tipo dinamico di Alimentatore in caso di richieste non coerenti vengono lanciati degli errori che dovranno essere gestiti dal chiamante. Si è fatta questa scelta in quanto risultava concettualmente più naturale. Inoltre sono presenti e pubblici metodi che mi permettono di fare la get dei tre componenti che sono getAlimentatore, getScarico e getMotore si è scelto di fare ciò in quanto si è scelto di considerare i tre componenti comunque indipendenti dal veicolo anche se facenti parte di esso.

Da Veicolo deriva Installazione che introduce la presenza di un impianto che si è scelto di rappresentare inserendo come campo dati un puntatore verso lo heap di un Impianto.

2.6 Altri classi del Modello

Sono inoltre presenti altre classi non facenti parti di gerarchia che sono Officina, DB_utenti e Data.

Data fornisce un tipo che ha lo scopo di rappresentare una valida data del calendario gregoriana e predispone tutti i metodi necessari per gestire tale oggetto.

Officina e DB_utenti sono invece entrambi classi contenitore. La prima rappresenta l'insieme dei veicoli gestiti ed è implementata tramite un std::vector di puntatori a Veicoli per sfruttare questa gerarchia e quindi poter avere sia Veicoli che Installazioni. La classe fornisci operatori di subscriting sia per posizione, sia per targa ognuno di questi e presente sia constante sia che permetta side effect (quelli che causano side-effect sono privati). L'unico modo per modificare un oggetto del contenitore è farlo tramite i metodi che esso fornisce . Come nelle altre classi oltre a metodi pubblici per ottenere semplici info sull'insieme gestito sono presenti metodi per l'input output su/da file.

L'altra classe contenitore è DB_Utenti , implementata tramite un std::vector di puntatori ad heap per sfruttare la gerarchia Utente. La classe offre metodi per la gestione degli utenti come rimozione, aggiunta get dei dati e modifica dei dati dei singoli utenti. Queste operazioni si possono attuare solo tramite la classe DB_Utenti in quanto è stato scelto di non fornire metodi per accedere direttamente agli oggetti Utente (E' presente operatore di subscripting in accessibilità privata).

2.7 Gerarchia Grafica

La gerarchia grafica cerca di rispettare la gerarchia degli oggetti su cui lavora. Ciò per rendere più facili la rappresentazione grafica di possibili eventuali estensioni del modello. Ogni oggetto delle gerarchie di Utente, Componente e Veicolo presente due gerarchie corrispondenti una che permette di creare un Widget al fine di rappresentarlo l'altro che permette la creazione di un oggetto di quel tipo.

Per la gerarchia di impianto si è scelto di creare solo due classi al fine di mostrare e creare un qualsiasi tipo impianto.

Inoltre nella parte grafica sono presenti ulteriori gerarchie di Finestre allo scopo di discernere le possibilità che ha un utente normale rispetto ad un utente Capo. È il caso di W_Inizio e W_InizioCapo. Altre classi grafiche sono presenti e derivano da QWidget (gestioneUtenti, W_registra, W_Consulta, W_accedi).

L'apertura e la chiusura di tutte le finestre è gestita dalla classe GestioneGui che si occupa anche di creare gli oggetti Officina e DB_utenti quando servono e deallocarli quando non servono.

3 - Descrizione dell'uso di codice polimorfo

3.1 Utilizzo polimorfico della gerarchia degli utenti

Innanzitutto l'uso polimorfico della classe Utente, per non creare memory leak il distruttore `e stato reso virtuale, in modo che quando si distrugge un puntatore a user viene richiamato il distruttore corretto. Oltre a ciò Utente presenta anche la funzione super polimorfa Utente* clona(), che crea una copia dell'oggetto e ne restituisce un puntatore, spesso utilizzata per creare condivisione profonda della memoria in DB_Utenti. Altre funzioni che fanno uso del polimorfismo nella gerarchia delle classi Utente sono bool getTipo(), SalvaInfoUtente.

3.2 Utilizzo polimorfico della gerarchia dei Componenti

Per quanto riguarda la gerarchia dei Componenti, esattamente come per la classe Utenti, il distruttore è virtuale e presenta il metodo Componente * clona() utilizzato sempre nel contenitore Officina per creare la condivisione di memoria profonda. Un altro metodo polimorfico presente è string getInfo(), che restituisce una stringa contenente tutte le informazioni relative

al tipo dinamico utilizzato nella in Veicolo per creare la propria funzione getInfo. Un altro metodo che usa il polimorfismo è Salva, metodo ampiamente utilizzato nella funzione di Salvataggio su file di Officina nel suo carattere polimorfico. Salva è anche richiamata dalle derivazioni successive in maniera statica.

3.3 Utilizzo polimorfico della gerarchia degli Impianti

La gerarchia degli Impianti presenta innanzitutto il distruttore virtuale in modo da evitare memory leak e anche metodo di clonazione virtuale, esattamente come le altre gerarchie precedentemente analizzate. Presenta i metodi virtuali ,gia descritti, getInfo (usato nella getInfo di installazzione e anche richiamato staticamente negli overriding successivi), getNome e idoneita (usato nel metodo Idoneita della classe veicolo.). Il polimorfismo è anche sfruttato tramite l'uso del puntatori a Impianto nella classe Installazione che mi permette così di poter inserire un qualsiasi tipo concreto.
Come nelle altre gerarchie con la stessa logica è presente il metodo virtuale Salva.

3.4 Utilizzo polimorfico della gerarchia Veicolo

Questa gerarchia fa un uso del polimorfismo che si limita alla funzione getInfo, clonazione, metodo di salvataggio su file e distruttore. Tutte queste funzioni sono implementate e utilizzate con la stessa logica degli analoghi nelle altre gerarchie. Si sfrutta questo polimorfismo nel vettore presente nella classe Officina.

3.5 Utilizzo polimorfico della gerarchia delle viste

La gerarchia delle viste presenta innanzitutto il distruttore virtuale in modo da evitare memory leak, esattamente come le altre gerarchie precedentemente analizzate.

4 - Manuale utente

L'applicazione fa uso di due file xml presenti nella directory principali eseguibile per salvare le informazioni, listaUtenti.xml per gli utenti e officina.xml per il database di Veicoli. Non è presente la necessità che questi file siano presenti, in tal caso verrà creato un utente admin di default con nome admin e password pass. In caso di eliminazione di admin questo verrà ricreato al prossimo tentativo di Login. Durante la navigazione tra le finestre, se si volesse tornare alla schermata precedente quando questa si è chiusa basta premete il pulsante di chiusura (la X). La prima schermata è d'accesso, da essa è presente la possibilità di inserire un nuovo utente di qualunque tipo. Dopo aver effettuato il login si aprirà una schermata in base al tipo di account in uso in cui sono presenti le varie funzionalità del gestionale.
La Schermata Consulta permette di visualizzare il contenuto dell'Officina tramite una lista di targhe. Facendo doppio click sulla targa verrà proposta nel Box in alto a sinistra le informazioni del Veicolo/Installazione selezionato, stesse informazioni posso anche essere visualizzate in una Finestra dedicata tramite il pulsante Mostra (se si è un utente di tipo capo la schermata proposta permettere anche la modifica di alcuni dati del veicolo e l'aggiunta o rimozione di bombole in caso sia una installazzione). È possibile ottenere un sottoinsieme della lista tramite le funzioni di ricerca presenti in alto a destra. La ricerca del proprietario è effettuata tramite prefisso. Il pulsante elimina togli dall'officina il Veicolo correntemente in selezione. In maniera analoga funziona la schermata GestisciUtenti.
Inserisci Veicolo/Installazione è la schermata che permette di inserire un Veicolo o una Installazione in Officina. Basta inserire i campi dati e poi procedere all'operazione con il pulsante inserisci, per discernere tra un Installazione o un Veicolo basta cliccare l'apposito pulsante. In caso di Installazione bisognerà inserire pure i dati riguardanti l'impianto che dovrà essere idoneo alle caratteristiche del veicolo.

5 - Ore utilizzate

Progettazione concettuale: 7h

scrittura codice: 12h

---

progettazione grafica (inclusa la scelta dei layout): 8h

scrittura gui: 46h

---

test: 3h

debug: 4h

---

totale: 80h

6 - Ambiente di sviluppo

Sistema operativo: Microsoft Windows 10 Pro 64-bit 10.0.10586

Compilatore: mingw32

QT: 5.7.0 Qmake: 3.0