Ivan Furlan Matricola: 1661622

Liberty - ConsumiLog

Introduzione

Ho voluto realizzare con questo progetto un'applicazione il cui scopo principale è tener traccia dei rifornimenti che si fanno dei veicoli in proprio possesso. I veicoli (a motore si intende) possono essere di qualunque tipo ed oltre ai rifornimenti, e quindi ai consumi, attraverso l'applicazione si possono gestire anche altri dati che caratterizzano i singoli veicoli. Ho cercare di implementare le classi della gerarchia in modo che in futuro si possano aggiungere facilmente nuove funzionalità.

Si presuppone che ogni rifornimento, di qualunque tipo, venga sempre fatto fino al livello massimo e che ogni volta venga azzerato il conta km parziale; così facendo tutti i dati riportati dal programma saranno il più attendibili possibili. Nel caso per esempio di veicoli ibridi si ipotizza che si tenga un parziale per ogni tipo di rifornimento (ES. il parziale per la benzina che a ogni pieno viene azzerato, e quello per le ricariche di corrente che allo stesso modo viene azzerato ad ogni ricarica).

Istruzioni di compilazione

Per compilare il progetto è necessario utilizzare il file Liberty.pro fornito insieme ai file sorgente, e quindi la compilazione può essere eseguita semplicemente attraverso il comando *˝qmake && make˝*.

Sviluppo

Il progetto è stato sviluppato su Ubuntu 20.04 LTS con Qt Creator 4.12.3, configurandolo per utilizzare Qt 5.9.5, quindi la stessa versione di Qt della VM che ci è stata fornita. Come compiler è stato utilizzato GCC 9.3.0.

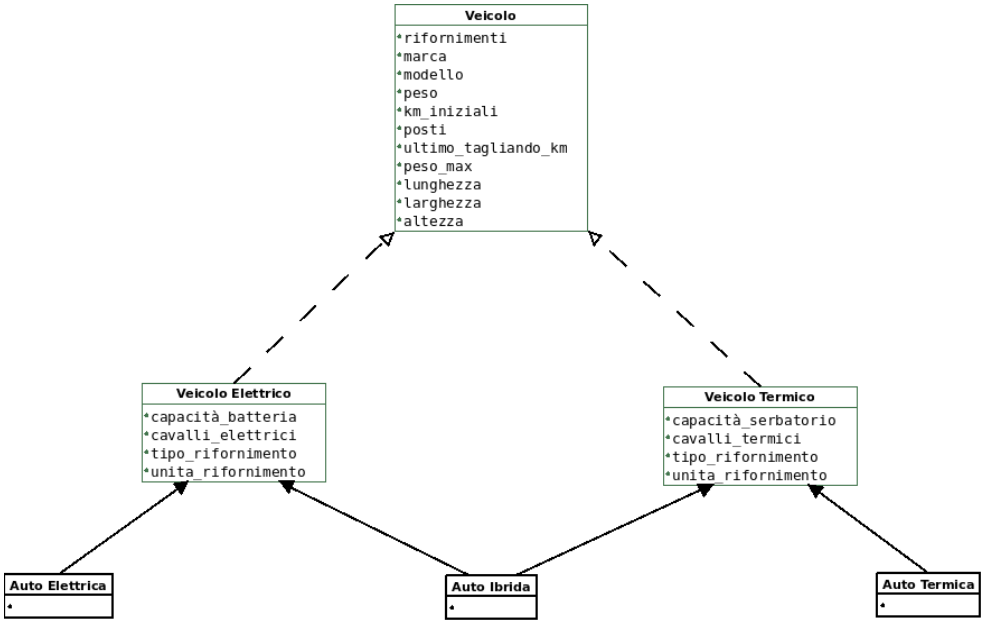
Nonostante l'abbia svolto individualmente, ho utilizzato il sistema di versionamento Git durante tutto lo sviluppo in modo da tenere lo storico del lavoro svolto, ed aiutarmi nel risolvere problemi dovuti a modifiche errate del codice.

Per l'implementazione del codice mi sono basato sul pattern Model-View-Controller, così da rendere possibile cambiare la vista senza dover modificare il codice del modello, o viceversa.

È stato sviluppato come da requisiti anche un container templetizzato cercando di fornire tutte le funzionalità che potessero servire per utilizzarlo in questo lavoro, come per esempio iteratori e funzioni di ricerca.

Non ho sviluppato un salvataggio e caricamento su file in quanto non essendo un gruppo di tre persone, ma singolo, non ne avevo l'obbligo, e probabilmente facendolo avrei sforato le 50 ore di lavoro richieste. Per rendere più semplice la valutazione e comprensione dell'applicativo ho creato una funzione nel file main.cpp per popolare il modello con alcuni dati esempio.

Gerarchia di tipi



La gerarchia è formata da 6 classi di cui 3 istanziabili (Auto elettrica, ibrida e termica). VeicoloElettrico e VeicoloTermico derivano virtualmente da Veicolo, e sono anch'esse classi astratte. VeicoloElettrico e VeicoloTermico sono astratte e quindi non istanziabili per favorire l'implementazione futura di altri tipi di veicoli o metodi; per esempio si pensi ad una moto, che potrebbe avere funzionalità o caratteristiche diverse da un auto termica, anche se deriverebbero entrambe da VeicoloTermico.

L'ereditarietà multipla è stata utilizzata nella classe AutoIbrida, e in particolare si tratta di una struttura a diamante con la classe Veicolo in comune.

In figura sono riportati solamente i campi dati delle varie classi.

Uso del polimorfismo

`virtual u_int getSommaKmRifornimenti() const;` restituisce la somma di tutti i km parziali di ogni rifornimento per sapere quanti km si hanno fatto. È segnato come virtuale perché, per esempio, nel caso dell'auto ibrida si hanno due tipi di rifornimenti con parziali separati, quindi il metodo necessita di una reimplementazione per poter fare la somma di solo parziali dello stesso tipo.

`virtual u_int getPesoTrasportabile(const u_short num_passeggeri = 0) const;` è un metodo non utilizzato dalla view, ma disponibile. Restituisce il peso massimo che si può caricare in veicolo considerando quanti passeggeri ci sono. È virtuale perché per esempio in un veicolo termico bisogna sottrarre dal peso massimo anche il peso del carburante nel caso il serbatoio fosse pieno.

`virtual u_int getKmAutonomia() const = 0;` è un metodo virtuale puro che ritorna il km l'autonomia massima del veicolo facendo il rifornimento al 100%. È puro perché non si hanno dati come capienza del serbatoio, delle batterie, o di entrambi in caso di veicolo ibrido.

`virtual u_short getCavalli() const = 0;` è un metodo virtuale puro che ritorna il totale dei cavalli del veicolo. È puro perché non si sa che tipo di veicolo sia, se elettrico, termico, oppure ibrido in cui i cavalli totali sarebbero la somma di quelli elettrici e termici.

`virtual u_short getKw() const;` effettua semplicemente una conversione da cavalli a Kw. È stato dichiarato virtuale perché si vuole dare la possibilità in caso di veicoli con ereditarietà multipla (tipo quelli ibridi), di avere il dato dei Kw di solo una parte del veicolo (ES. sapere i Kw solo elettrici). Se ci fosse stata un'implementazione che convertisse semplicemente il valore restituito da getCavalli(), l'utente per aver lo stesso dato (cioè, esempio, i Kw soltanto elettrici) avrebbe dovuto chiedere i cavalli del sottotipo desiderato mediante operatore di scope :: (*VeicoloElettrico::getCavalli()*), e successivamente sarebbe compito dell'utente fare la conversione in Kw.

`virtual bool fareTagliando() const = 0;` è un metodo virtuale puro che ritorna se è consigliato o meno fare un tagliando al veicolo. È puro perché il dato di ogni quanti km è consigliabile farlo dipende da veicolo a veicolo, per esempio anche il se è una moto o una macchina.

`virtual bool checkCorrettezzaRifornimento(const Rifornimento& ) const;` ritorna true o false in base a se il rifornimento che viene valutato è accettabile o meno per quel veicolo. C'è un'implementazione di base che controlla semplicemente che i valori siano positivi, poi per altri tipi di controlli servono dati che solo le classi derivate hanno. (Es. che non siano stati riforniti più della capienza del serbatoio o della batteria)

`virtual Veicolo * clone() const = 0;` è un metodo virtuale puro che ritorna un puntatore ad una copia dell'oggetto corrente. L'implementazione viene appunto fatto solo su classi istanziabili, quindi Auto elettrica, termica, o ibrida, le quali ritornano un puntatore a se stesse, ma che ovviamente tipa su Veicolo essendo classi derivate.

La classe Container<T>

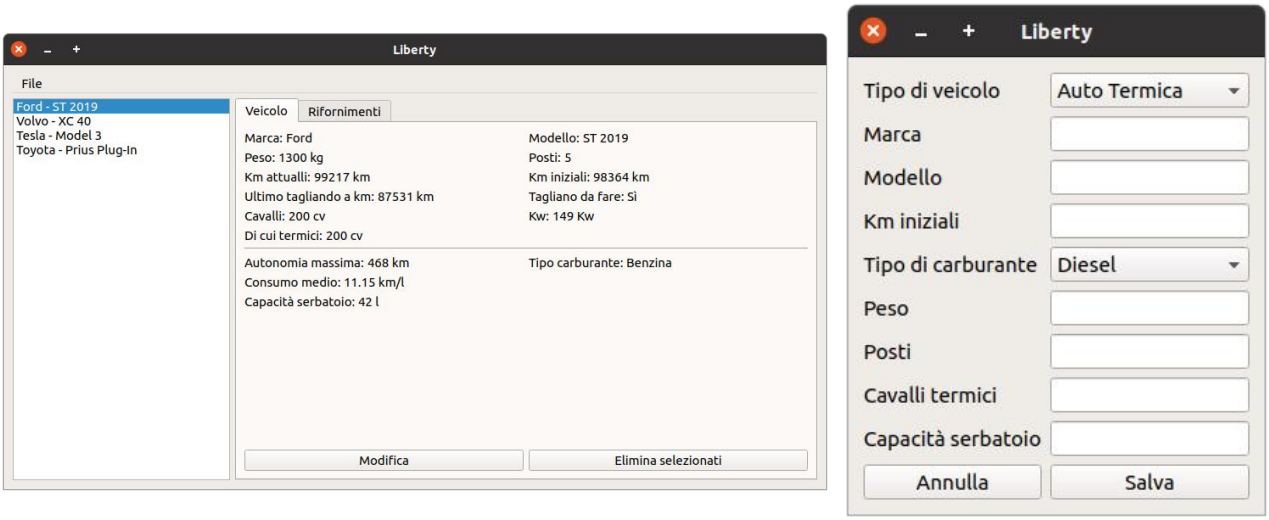
Come contenitore ho deciso di creare una struttura che prendesse ispirazione dal contenitore std::vector<T>, che contenesse al suo interno un array. Una struttura del genere credo sia appropriata per il progetto che ho realizzato in quanto le procedure di modifica, aggiunta o eliminazione di veicoli sono eventi che teoricamente accadono raramente, e una volta inseriti restano finché non si cambia a distanza anche di anni. Questa struttura è pensata quindi per essere usata quasi unicamente in lettura.

Tempo dedicato al progetto

Ho dedicato all'intero progetto circa 52 ore così suddivise:

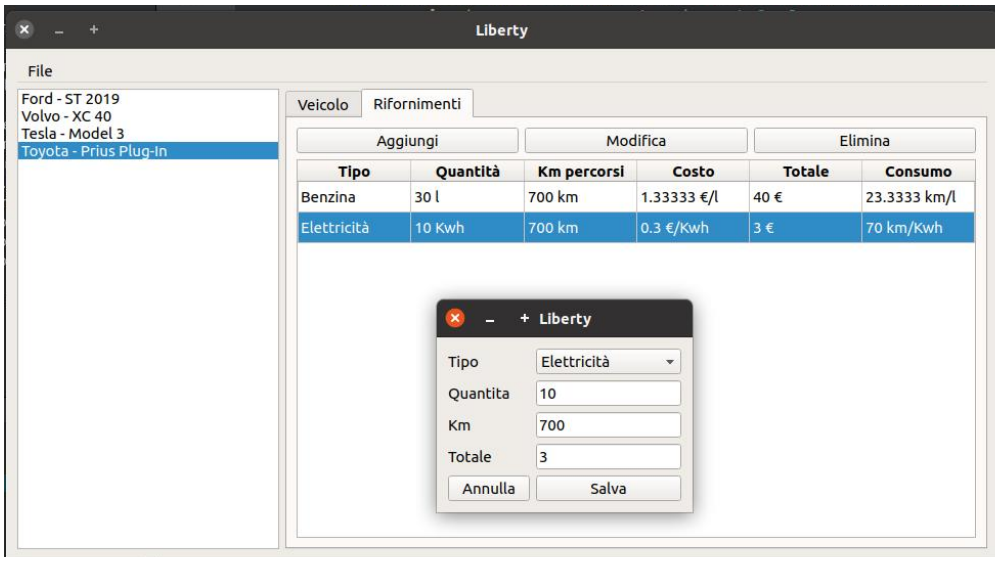
- Analisi del problema e valutazione delle possibili alternative: 4h
- Analisi e studio della libreria Qt, facendo anche delle prove per capirne le funzionalità: 7h
- Progettazione e implementazione contenitore: 8h
- Progettazione e implementazione della gerarchia e modello: 10h
- Progettazione e implementazione della GUI: 15h
- Debug e test: 6h
- Stesura relazione 2h

Presentazione dell'applicazione



La schermata principale si presenta come in figura e selezionando con il mouse un veicolo dalla lista di sinistra verranno mostrati nella parte inferiore alla linea dati tra cui i consumi medi e l'autonomia massima, invece nella parte superiore, oltre a dei dati che caratterizzano il veicolo stesso, ci sono i km attuali che sono calcolati come la somma ai km iniziali di tutti i parziali dei rifornimenti, e in base ai km attuali se ci sarebbe o meno il tagliando da fare.

Andando sul menu *"File"* è possibile aggiungere un nuovo veicolo, eliminare i selezionati, oppure uscire dell'applicazione.



Cliccando invece sul tab *"Rifornimenti"* si possono appunto gestire tutti i pieni o le ricariche in base al tipo di veicolo, e facendo doppio click su una riga, o un click sul tasto modifica, si possono andare a modificare quelli già inseriti.