Progetto di Programmazione ad Oggetti, a.a. 2020/2021

Dmitry Pluzhnikov, matricola 1169886

Sommario

L'applicazione "EASYCAR" è stata svilupata per il supporto di attività di un rivenditore moto, auto, camper e camion. Permette di conservare i dati di tutti veicoli disponibili, aggiunta di un nuovo veicolo, rimozione e ricerca per tutti parametri. Quindi possiede:

Manuale utente

1. Menubar con campi:
"**File**" con disponibile dentro un etichetta "*Chiudi app*" che permette di chiudere l'applicazione;
"**Inserimento**" che fa aprire la finestra di inserimento dei dati di un nuovo veicolo da aggiungere nella base di dati;
"**Ricerca**" che fa aprire finestra di inserimento dei dati di un nuovo veicolo;
"**Visualizza**" che contie dentro etichette "*Moto*", "*Auto*", "*Camper*", "*Camion*" quali permettono di aprire la finestra di resultato di ricerca con tutti veicoli disponibile della categoria scelta;
"**Pulisci**" che ha funzione di pulire e azzerare i campi della scheda aperta.

2. Finestra di ricerca che è interfaccia inziale. Contiene 4 tab (quali possono essere spostati su e giù alla preferenza) corrispondenti alle catogorie dei veicoli gestiti, con campi per inserire i valori entro quali si vuole fare ricerca. Tutti i campi sono opzionali per la compilazione (quindi la ricerca accetta veicolo con qualsiasi valore del parametro corrispondente), forche per "*Cambio*" e "*Carburante*" per Moto e Camion; "*Cambio*", "*Carburante*" e "*Numero passeggeri*" per Auto; *Cambio*", "*Carburante*", "*Numero passeggeri*" e "*Posti per dormire*" per Camper. Si prende anche in cosiderazione se i valori cercati massimi minori dei minimi (con il seguente avviso). Conclude la finestra tasto "*Cerca*" che fa avviare la ricerca e uscita di finestra del risultato oppure avviso quando non ci sono veicoli disponibili o parametri di ricerca non sono corretti.

3. Finestra di iserimento di un nuovo veicolo che contiene sempre 4 tab per scegliere la categoria del veicolo da inserire con seguente tasto "*Inserisci*" per tentare inserimento. In questo caso tutti campi (a parte checkbox) sono obbligatori per la compilazione. In mancanza viene corrispondente avviso.

4. Finestra del resultato di ricerca contiene elenco dei veicoli trovati, tasto "*Elimina*" per eliminare un veicolo selezionato (si disattiva quando elenco è vuoto) e tasto "*Chiudi*" per chidere la finestra.

Struttura progettuale

Il codice è stato sviluppato in lingua inglese per poter esser lavorato anche dalle persone straniere.L'applicazione è stata realizzata con design pattern Model-View-Presenter suggerito alle lezioni di tutorato. Quindi è composta dalla classe View che communica alla classe Controller quando c'è qualche evento da raccogliere, lui a sua volta lo elabora e conforme alla tipologia dell'evento communica conl a classe Model, View od entrambe. (Purtroppo quando tentavo di connettere i segnali in direzione dalla View al Controller programma abbortiva e nonsono riuscto di trovare la causa, quindi sono stato costretto di connettere i segnali della View con i propri slot e poi tramite puntatore communicare con Controller, però idea iniziale era come descritta sopra.

La classe View per poter realizzare le finestre descritte sopra usa classe QTabWidget, ed ogni scheda è realizzata con le classe definite da me che si trovano nelle cartelle *view/tabs_for_LayoutInsert* per la finestra di inserimento, e *view/tabs_for_LayoutSearch* per la finestra di ricerca.

Parte Controller è realizzata con omonima classe.

Parte Model realizzata con la classe Model, quale costituisce un'estensione dell'istanziazione del template di classe Container con template di classe DeepPtr, istanziato a sua volta con la classe Vehicle.

Template di classe Container<T> è stato realizzato prendendo orientamento dalla classe Vector della linreria STL. Possiede di due claasi annidate iterator e const_iterator che permettono di iterare e nel caso iterator anche

modificare i dati. Queste classe possiedono anche del campo Container<T>* container per poter essere confrontate fra di loro se puntano allo stesso contenitore.

Template di classe DeepPtr<T> ha lo scopo di gestire gli oggetti in modo automatico, per evitare condivisione della memoria e memory leaks costruendo di copia, facendo assegnazione e distruzione in modo "profondo".

Gerarchie di classi

GERARCHIA 1:

Classe base astratta **Vehicle** che ha come sottoclassi:
- dirette concrete **Moto** e **Truck**;
- diretta astratta **Auto**.

Classe **Auto** a sua volta ha sottoclassi dirette concrete **Car** e **Camper**.

Questa gerarchia si usa per rappresentare i veicoli gestiti. Perciò:

Classe **Vehicle** contiene campi dati che rappresenta informazioni communi per tutti veicoli:
- string make (marca veicolo)
- string model (modello)
- unsigned int registrationYear (anno immatricolazione)
- unsigned int mileage (chilometraggio)
- double basePrice (prezzo base)
- unsigned short int fuelType (tipo alimentazione)
- unsigned short int transmissionType (tipo cambio)
- unsigned int power (potenza)
- unsigned int displacement (cilindrata)

Classe **Moto**:
- bool sidecar (presenza di carozzina)
- engineProtection (protezione motore)

Classe **Track**:
- unsigned int maxLoadability (carico massimo permesso)
- bool sleppingPlace (presenza posto per dormire)

Classe **Auto** contiene seguenti campi che sono comuni per **Car** e **Camper**:
- bool conditioner (presenza di climatizzatore)
- unsigned int passengerNum (numero posti per passegeri)
- bool pintleHook (presenza gancio traino)
- bool sensors (presenza sensori parcheggio)

Classe **Car**:
- bool stWagon (se è station wagon)
- bool sunroof (presenza lunotto)
- bool luggagerack (presenza portapacchi)

Classe **Camper**:
- unsigned int sleepingPlaces (numero posti per dormire)
- bool toilet (presenza bagno)
- bool kitchen (presenza cucina)

Tutte le classi dotate di corrispondenti getter che ritornano i valori dei campi.

Classe **Vehicle** contiene i seguenti metodi virtuali:
`virtual~ Vehicle();` - distruttore virtuale
`virtual Vehicle* clone() const=0;` - effettua clonazione dell'oggetto
`virtual bool isBetweenType(VehicleSearchResult*) const=0;` ritorna se l'oggetto su quale e stato chiamato corrisponde ai parametri di ricerca passati tramite `VehicleSearchResult*`.

Si usa dentro metodo `bool Vehicle::isBetween() const;` chiamato dentro metodi:

```
vector<pair<Moto, unsigned int>>* Model::searchMoto(MotoSearchResult*) const;
vector<pair<Car, unsigned int>>* Model::searchCar(CarSearchResult*) const;
vector<pair<Camper,unsigned int>>* Model::searchCamper(CamperSearchResult*) const;
vector<pair<Truck, unsigned int>>* Model::searchTruck(TruckSearchResult*) const;
```

A sua volta il metodo `bool Vehicle::isBetween() const;` chiama dentro se seguenti metodi virtuali:

`virtual unsigned int Vehicle::emissionClass() const=0;` ritorna valore della classe emmisione del veicolo.
`virtual bool Vehicle::isNoviceDriver() const=0;` ritorna se il veicolo può essere guidato da un neopatentato
`virtual double Vehicle::vehicleTax() const=0;` ritorna valore della tassa annuale del veicolo
`virtual double Vehicle::getFullPrice(bool) const=0;` ritorna il prezzo del veicolo se è stata aqcuistata anche garanzia
`virtual double Vehicle::loansMonthlyPayment(bool) const=0;` ritorna il valore della rata mensile se è stato deciso acquistare il veicolo tramite un finanziamento
Tutti questi metodi sono stati progettati perchè diverse catogorie dei veicoli hanno diverse formule del calcolo di valori rappresentati.

GERARCHIA 2:

Classe base astratta **VehicleSearchResult** ha quattro sottoclassi dirette concrete:
- **MotoSearchResult**
- **CarSearchResult**
- **CamperSearchResult**
- **TruckSearchResult**

Questa gerarchia e stata progettata per passare parametri di ricerca a Model. Classe **VehicleSearchResult** a come campi:

- bool make: è un flag che ha valore false quando il campo “Marca” della finestra di richerca non è stato compilato, quindi questo parametro va ignorato
- bool model: stesso funzionamento del flag precednte per il campo “Modello”
- bool warranty: suo valore true significa che nel range dei valori di prezzo minimo e massimo deve entrare prezzo del veicolo calcolato considerando supplemento di garanzia
- bool loan: questo flag significa che nel range dei valori di prezzo minimo e massimo deve entrare valore di pagamento mensile del finanziamento
- bool isNoviceDriver: fa selezionare solo i veicoli, quali possono essere guidati anche da neopatentati
- unsigned short int emissonClass: riporta il valore di classe emmissione dei veicoli cercati
pertutti sudetti campi sono disponibili dei getter corrispondenti.

Classe **VehicleSearchResult** possiede di due metodi
- `virtual Vehicle* getMinValues()=0;`
- `virtual Vehicle* getMaxValues()=0;`

che hanno funzione di ritornare puntatori a due oggetti della stessa classe per quale si effettua ricerca e che contengono valori minimi e massimi di ricerca. Questi metodi si invocano sempre nel metodo
`bool Vehicle::isBetween() const;`

GERARCHIA 3:

Questa gerarchia è composta da classe base astratta **VehicleSearchResultDialog** e quattro sottoclassi dirette concrete:
- **MotoSearchResultDialog**
- **CarSearchResultDialog**
- **CamperSearchResultDialog**
- **TruckSearchResultDialog**

che implementano
```
virtual void updateSearchResultDialog() const=0;
virtual unsigned int returnVehicleForRemoveIndex() const=0;
```

Lo scopo di questa gerarchia è di realizzare la finestra di resultato di richerca per ogni tipologia del veicolo.
Il metodo

```
virtual unsigned int returnVehicleForRemoveIndex() const=0;
```

ha funzione di ritornare l'indice dell'elemento da eliminare nel Modlello. Si invoca da

```
unsigned int View::returnVehicleFor RemoveIndex() const;
```

Il metodo

```
virtual void updateSearchResultDialog() const=0;
```

ha invece la funzione di aggiornare gli indici di elementi nel Model quali si trovano dopo elemento eliminato. Va invocato da

```
void View::updateSearchResultDialog() const;
```

Note

Per il progetto non è stato usato nessun file supplementare per input/output, la sua funfione prende file loadData.h che fa riempire il container per comodita di analisi. Questo file non fa parte del progetto quindi non va preso in considerazione.

Visto che il progetto contiene file ***progetto.pro*** la compilazione si effetua invcando i comandi **qmake** e **make**.

Per realizzare l'applicazione sono state impiegate approssimativamente 60 ore di lavoro effettivo:

- analisi preliminare del problema: 3 ore
- progettazione modello e GUI: 8 ore
- apprendimento libreria Qt: 15 ore
- codifica modello e GUI: 30 ore
- debugging e testing: 4 ore

Il superamento della soglia di 50 ore è dovuta allo studio della libreria Qt per trovare le soluzioni ottimali della realizziazione GUI.

Il progetto è stato sviluppato con sistama operativo Linux Ubuntu 18.04.3 LTS , compilatore gcc 7.5.0, libreria Qt 5.9.5