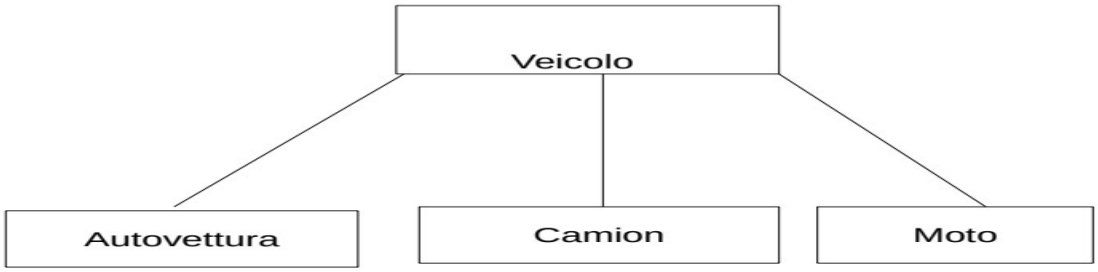
RELAZIONE

Progetto di PROGRAMMAZIONE AD OGGETTI 18-19

QONTAINER

MARCEL JUNIOR WANDJI 1171044

Descrizione della gerarchia dei tipi e dell'uso di chiamate polimorfe



L'insieme delle classi realizzate serve per la gestione di un parco di veicoli a motore. In particolare le classi sono realizzate come segue:

una classe polimorfa astratta Veicolo i cui oggetti rappresentano un veicolo a motore memorizzato in un parco. Ogni veicolo è caratterizzato dal telaio , dal modello , dall'alimentazione e dalla targa. La classe è astratta in quanto prevede i seguenti metodi virtuali puri:
- un metodo bool controllaTarga(), che ritorna true se la targa corrisponde al tipo di veicolo;
-un metodo string tipoVeicolo() , che ritorna il tipo di veicolo;
Inoltre la classe Veicolo ha un metodo virtuale non puro string veicoloString() ,che ritorna una descrizione del veicolo.

Una classe concreta AutoVettura derivata da Veicolo i cui oggetti rappresentano un veicolo di tipo auto. Ogni oggetto AutoVettura è caratterizzato dal numero di posti. La classe AutoVettura implementa i metodi virtuali puri di Veicolo come segue:
-per ogni oggetto AutoVettura, controllaTarga() ritorna true se la lunghezza della stringa della targa è uguale a 7 e i primi e gli ultimi due caratteri sono lettere mentre i restanti 3 sono numeri.
-per ogni oggetto AutoVettura, tipoVeicolo() ritorna "Autovettua"
- per ogni oggetto AutoVettura, veicoloString() ritorna la sua descrizione
Una classe concreta Camion derivata da Veicolo , i cui oggetti rappresentano un veicolo di tipo camion. Ogni oggetto Camion è caratterizzato dal numero di assi. La classe Camion implementa i metodi virtuali puri come segue:
-per ogni oggetto Camion, controllaTarga() ritorna true se la lunghezza della stringa della targa è uguale a 7 e i primi e gli ultimi due caratteri sono lettere mentre i restanti 3 sono numeri e la prima lettera deve essere un 'x'
-per ogni oggetto Camion, tipoVeicolo() ritorna "Camion"
-per ogni oggetto Camion, veicoloString() ritorna la sua descrizione
Una classe concreta Moto derivata sempre da Veicolo, i cui oggetti rappresentano un veicolo di tipo moto. Ogni oggetto Moto è caratterizzato da guida libera o no. La classe Moto implementa i metodi virtuali puri come segue:
-per ogni oggetto Moto, controllaTarga() ritorna true se la lunghezza della stringa della targa è uguale a 7 e i primi caratteri due sono lettere mentre i restanti sono numeri.
-per ogni oggetto Moto, tipoVeicolo() ritorna "Moto"
-per ogni oggetto Moto, veicoloString() ritorna la sua descrizione

-La classe Veicolo è una classe astratta e quindi non si può istanziare oggetti di questo tipo. Tutti i suoi metodi sono implementati (ridefiniti) nelle sue classi derivate (AutoVettura, Camion, Moto).
-La classe Container è una classe template contenitore.

-La classe Modello rappresenta il modello ed è caratterizzata da un oggetto di tipo Contaner<Veicolo*> che contiene tutti i veicoli presenti nella lista. I suoi metodi rendono possible l'inserimento di un nuovo veicolo nella lista, la cancellazione di un veicolo nella lista ,la modifica ,il numero di veicoli presenti nella lista, scrittura sul file e lettura dal file. Contiene anche un metodo polimorfo getVeicolo che ritorna un puntatore polimorfo ad un veicolo di indice i.
-La classe ListView è una classe derivata da QListView ed è utile per dimenzionare la view con sizeHint().
-La classe QListModelAdapter è derivata da QAbstractListModel e permette alla vista di visualizzare elementi del modello dei dati.
-La classe QFilterProxyModel derivata da QSortFilterProxyModel, funziona da intermediario fra QListModelAdapter e ListView per permettere alla vista di visualizzare solo certi elementi (veicolo) sulla base dell'input di ricerca inserito pur non dovendo effettuare alcuna operazione sul reale modello dei dati sottostante e senza duplicare dati.
-Le classi EditAuto, EditCamion e EditMoto sono derivate da QDialog e servono per costruire un autovettura, camion e moto rispettivamente.
-La classe MainWindow è derivata da QWidget ed è la finestra principale dell'applicazione. Contiene la vista che permette di visualizzare gli elementi del modello ,pulsanti per aggiungere , modificare e rimuovere elementi, salvare i dati e chuidere la finestra (l'applicazione), la search bar per cercare un elemento nella lista e anche un menu.

Descrizione del formato del file di caricamento/salvataggio del contenitore

-Il tipo di file scelto per il caricamento e il salvataggio del contenitore è xml ed è scritto e letto da QXmlStreamWriter e QXmlStreamReader rispettivamente. Il tag iniziale (root) marca l'inizio della scrittura e della lettura del file mentre la fine del tag è indicato con </root>.

-Ogni oggetto nel contenitore è salvato nel file scrivendo I suoi campi dati, cominciando dal suo tipo nel tag veicolo , il telaio nel tag telaio, il modello del veicolo nel tag modelo, l'alimentazione nel tag alimentazione, la targa nel tag targa e il tag text contiene il dato corrispondente al carattere particolare di ogni tipo di veicolo. Tutti i tag dopo quello del veicolo viene chiuso primo dell' apertura del successivo. Quando si chuide il tag text ,viene chiuso il tag veicolo che indica la fine di scrittura di un oggetto di tipo veicolo. Il meccanismo viene ripettuto per tutti I veicoli nel contenitore. Quando tutti gli oggetti nel contenitore sono scritti, il writter marca la fine del file scrivendo $</root>$.
-Analogamente con la lettura, il reader legge il root come l'inizio del file. Legge ogni tag e quando legge $</veicolo>$ che indica la fine di un veicolo, viene costruitto un veicolo corrispondente al tipo nell'attributo type. Il meccanismo viene ripettuto finché il reader non legge la fine del file.

Manuale utente della GUI

La gui è abbastanza facile da usare;
-Per aggiungere un veicolo basta cliccare su "Aggiungi" in alto a sinistra poi cliccare sul tipo di veicolo che si vuole aggiungere
-Per rimuovere ,modificare o avere dettagli di un veicolo, prima si seleziona il veicolo poi si preme sul pulsante rimuovi ,modifica o dettagli
-Con il pulsante "clear errate" si cancella tutti veicoli con targhe errate
-Per salvare nel file le modifiche , cliccare su "File" menu in alto a sinistra e poi su salva
-La ricerca viene effettuata sulla barra di ricerca, come segue:

*per avere solo veicoli di un certo tipo si digita il nome del tipo.Per esempio per avere una lista di veicoli di tipo AutoVettura si digita autovettura
*per avere tutti i veicoli di una certa alimentazione per esempio benzina si digita “A. benzina “ oppure semplicemente benzina
* per avere tutti i veicoli di un certo modello per esempio fiat500 si digita “m. fiat500” oppure semplicemente fiat500
*per trovare una targa (se c’e’) per esempio AA000AA si digita “t. AA000AA” o semplicemente AA000AA
-Il numero in basso a destra indica il numero di elementi (veicoli) presenti nella lista
-Ogni veicolo è visualizzato in maniera opportuna solo quando la targa corrispondente è corretta

Istruzioni di compilazione ed esecuzione

-compila ed esegue correttamente sulla macchina *ssh.studenti.math.unipd.it*
-La compilazione è possibile invocando la sequenza di comandi : qmake ⇒ make
-La compilazione non necessita l’invocazione del comando qmake -project (nella consegna c’è il file Qontainer.pro)

Numero di ore richieste per la realizzazione del progetto

-2 ore per l’analisi totale del problema
-5 ore per la progettazione del modello e GUI
-20 ore per la codifica del modello,GUI
-4 ore di tuttorato
-2 ore di debbugging e testing
-17 ore apprendimento della libreria Qt