Alberto Cocco

Matricola 1167891

QDrink

Progetto di programmazione ad oggetti A.A. 2018/2019

Ore complessive impiegate:

- Analisi preliminare del problema: 4 ore

- Progettazione modello: 5 ore

-Progettazione GUI: 7 ore

-Apprendimento libreria qt: 8 ore

-Codifica modello e gui: 18 ore

- debugging e testing: 9 ore

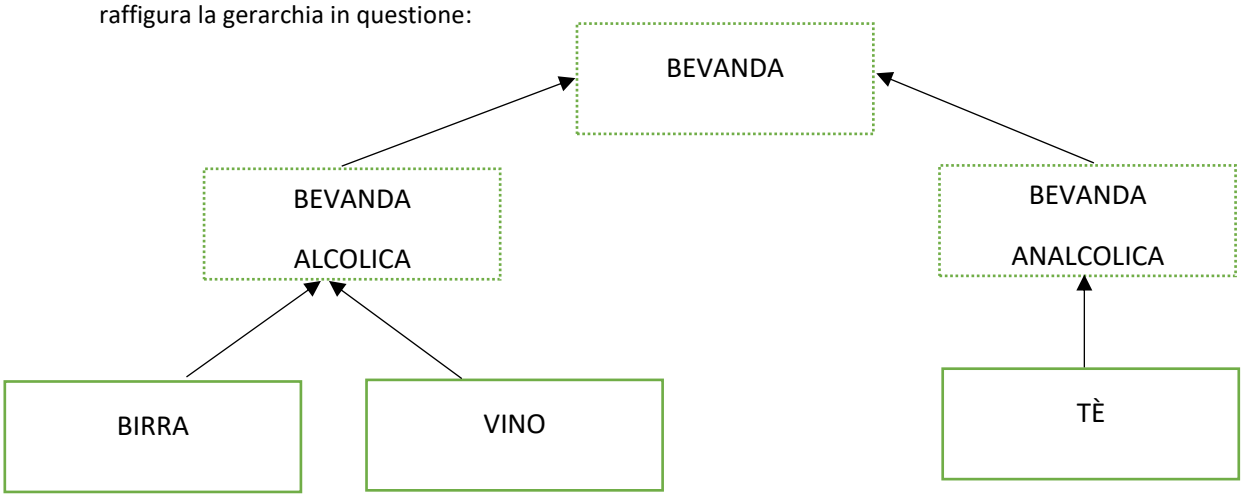
- Totale : 51 ore

Scopo del progetto:

Il progetto si propone come scopo quello di realizzare un'applicazione per la gestione di bevande di qualsiasi tipo. Esso fornisce alcune funzionalità essenziali quali: inserimento di una bevanda, rimozione di una bevanda, ricerca filtrata di elementi, salvataggio su file e caricamento da file.

Descrizione della gerarchia e dei principali metodi virtuali:

La gerarchia di QDrink è composta da una classe base virtuale astratta chiamata bevanda da cui derivano pubblicamente altre due classi virtuali astratte denominate bevanda alcolica e bevanda analcolica. Da bevanda alcolica derivano pubblicamente le classi concrete birra e vino, mentre da bevanda analcolica deriva pubblicamente la classe concreta tè. Sarà possibile estendere la gerarchia aggiungendo delle nuove classi derivate concrete delle quali si dovrà dare una corretta implementazione dei metodi virtuali. Nella figura sottostante è rappresentano un diagramma che raffigura la gerarchia in questione:



In tutta la gerarchia sono presenti metodi virtuali puri e metodi virtuali che rendono le classi polimorfe. Segue una breve descrizione dei principali metodi virtuali:

Nella classe base astratta bevanda sono presenti i seguenti metodi virtuali puri:

- *virtual std::string tipo() const=0*
- *virtual bool isAlcoholic() const=0*
- *virtual ~bevanda()= default*. Il distruttore virtuale permette la corretta distruzione degli oggetti delle classi polimorfe e nelle sue sottoclassi.

E i metodi virtuali con il seguente contratto:

- *virtual double CalcPrezzoTot() const*. CalcPrezzoTot restituisce il prezzo totale della bevanda su cui viene invocato, moltiplicando il prezzo unitario della bevanda per la quantità presente.
- *virtual std::string getFields() const.* getFields restituisce una stringa contenente tutti i campi dati della classe accompagnati dal rispettivo valore (nel caso in cui i campi dati siano int o qualsiasi altro tipo diverso da string vengono convertiti a string). È utilizzata, inoltre, una variabile di tipo stringstream per configurare l'approssimazione del valore del campo dati prezzo al centesimo, servendosi del metodo *setprecision()* della libreria std::iomanip. Tale metodo è risultato necessario per l'inserimento degli elementi all'interno del QListWidget della view.

La classe derivata astratta bevanda alcolica fornisce un'implementazione dei seguenti metodi virtuali:

- *virtual bool IsAlcoholic() const.* Il metodo overridato restituisce true se l'oggetto sul quale viene invocato è una bevanda alcolica.
- *virtual std::string getFields() const override* che restituisce una stringa contenente tutti i campi dati della classe accompagnati dal rispettivo valore.

La classe derivata astratta bevanda analcolica ha il seguente metodo virtuale puro:

- virtual bool isCarbonated() const=0

E fornisce un'implementazione dei seguenti metodi virtuali:

- *virtual bool IsAlcoholic() const.* Il metodo overridato restituisce false se l'oggetto sul quale viene invocato è una bevanda analcolica.
- *virtual std::string getFields() const override* che restituisce una stringa contenente tutti i campi dati della classe accompagnati dal rispettivo valore.

La classe concreta birra derivata da bevanda alcolica possiede un campo dati statico che rappresenta una tassa applicata sulle bevande di tipo birra. Essa implementa i metodi virtuali:

- *virtual double CalcPrezzoTot() const override*. CalcPrezzoTot restituisce il prezzo totale della bevanda su cui viene invocato, moltiplicando il prezzo unitario della bevanda per la quantità presente. Se la bevanda è di tipo birra tiene conto della tassa applicata.
- *virtual std::string tipo() const override. I*l metodo overridato restituisce il tipo della bevanda.
- *virtual std::string getFields() const override* che restituisce una stringa contenente tutti i campi dati della classe accompagnati dal rispettivo valore.

La classe concreta vino derivata da bevanda alcolica implementa i metodi virtuali nel seguente modo:

- *virtual std::string tipo() const override*. Il metodo overridato restituisce il tipo della bevanda.
- *virtual std::string getFields() const override* che restituisce una stringa contenente tutti i campi dati della classe accompagnati dal rispettivo valore.

La classe concreta tè derivata da bevanda analcolica implementa i metodi virtuali:

- *virtual std::string tipo() const override.* Il metodo overridato *restituisce il tipo della bevanda.*

- *virtual std::string getFields() const override* che restituisce una stringa contenente tutti i campi dati della classe accompagnati dal rispettivo valore.
- virtual bool isCarbonated() const override che restituisce true se la bevanda analcolica sul quale viene invocato il metodo è gassata.(Al momento questo campo non ha alcuna utilità, ma in futuro se si vorranno aggiungere nuovi tipi di bevanda analcoliche (es. bibite) se ne potrà fare un buon uso).

Classe data

È stata implementata una classe di nome data per definire la data di scadenza delle bevande. Essa è dotata di un convertitore al tipo stringa che è stato utilizzato per realizzare la funzione di save.

Qontainer

Come container del progetto è stato scelto di usare un template di vector. Esso è un array dinamico avente come campi dati un puntatore al tipo T, la size del vector e la capacity. È prevista un'operazione di ridimensionamento dell'intero array nel caso di inserimento di un numero di elementi maggiori della capacity fissata. Le operazioni di accesso casuale e di inserimento o rimozione di elementi alla fine vengono quindi eseguite in tempo O(1) costante. Il container è dotato di costruttore di copia profonda, assegnazione profonda e distruttore profondo. Per accedere e iterare sugli elementi dell'array sono presenti iterator e const iterator dotati di opportuni overloading dei principali operatori. Le più importanti funzionalità del container sono:

- void push_back(const T& t) : aggiunge l'elemento di tipo T come l'ultimo elemento del container. Nel caso in cui size>= capacity raddoppia la capacity dell'array.
- bool is empty() const: restituisce true se il container è vuoto.
- T&(unsigned int i) const: restituisce un riferimento al tipo T in posizione i nell'array.
- void erase(iterator it): rimuove l'elemento puntato dall'iteratore it. (è disponibile anche una versione che rimuove l'elemento di indice intero indicato).
- T& front(): restituisce un riferimento al primo elemento del container.
- T& back(): restituisce un riferimento all'ultimo elemento del container.
- I metodi get per size e capacity.

Model

Il pattern utilizzato per la realizzazione dell'applicazione è model view controller. Si è scelto di usare tale pattern per favorire la totale separazione di parte logica e GUI al fine di poter sfruttare la prima anche in applicazioni diverse da qt. Il modello è una classe che rappresenta la parte logica: esso ha un campo dati *container <bevanda*> obj* che istanzia il template del container al tipo bevanda*. Nel container saranno quindi contenuti puntatori polimorfi alla classe base astratta della gerarchia. Oltre al campo dati *container<bevanda*> obj* è presente un altro campo dati chiamato *container <bevanda*> search* che contiene gli elementi filtrati dalle operazioni di ricerca.

Il modello istanzia i metodi del container e offre la possibilità di salvare i dati su file XML o caricare dati da file XML. Sono presenti ben 3 metodi che producono chiamate polimorfe e sono:

- *bool b_isAlcoholic(unsigned int) const:* restituisce true se l'oggetto su cui viene invocato è una bevanda alcolica.
- *std::string b_getTipo(unsigned int) const:* restituisce il tipo dell'oggetto su cui viene invocato
- *double b_calcPrezzoTot(unsigned int) const:* restituisce il prezzo totale dell'oggetto sul quale viene invocato

I metodi per effettuare caricamento da file e salvataggio su file sono i seguenti:

- *void save(QString&) const :* fa utilizzo della classe QXmlStreamWriter per effettuare la scrittura su file.
- *void load(std::string&):* fa utilizzo della classe QXmlStreamReader per effettuare la lettura da file.

Sono presenti poi dei metodi wrapper che servono a istanziare e richiamare i metodi del container elencati nel paragrafo precedente.

Gui

La gui dell'applicazione si compone di una schermata principale per l'inserimento degli elementi e di due QDialog per la ricerca e la modifica. La mainwindow contiene i QPushButton per l'inserimento, modifica degli elementi e la QMenubar con cui interagire per caricare dati o salvare dati oppure aprire il dialog per iniziare una ricerca filtrata. I QLineEdit, QSpinBox, QComboBox, QCheckBox e QTextEdit della mainwindow sono dichiarati in una classe separata chiamata mywidgets di cui mainwindow ha un puntatore. La classe mywidgets si occupa inoltre di inserire i QWidgets all'interno di formLayout personalizzati nella classe form. Nelle immagini sottostanti sono raffigurate la mainwindow principale e la schermata di ricerca. È stata inoltre creata una classe mylistitem derivata da QListWidgetItem per personalizzare gli elementi da inserire nel QListWidget.

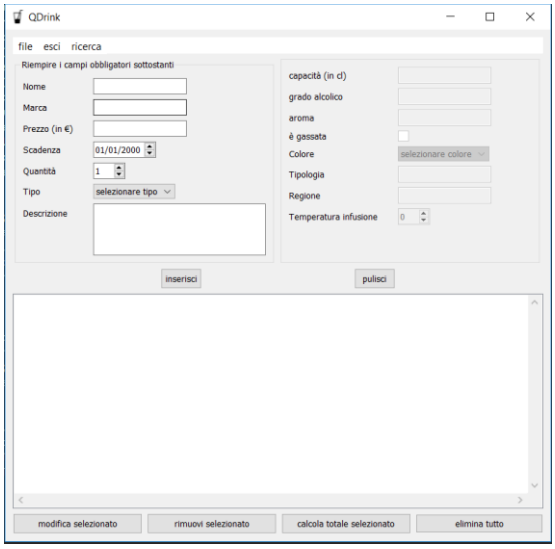


Figura 1: schermata di inserimento

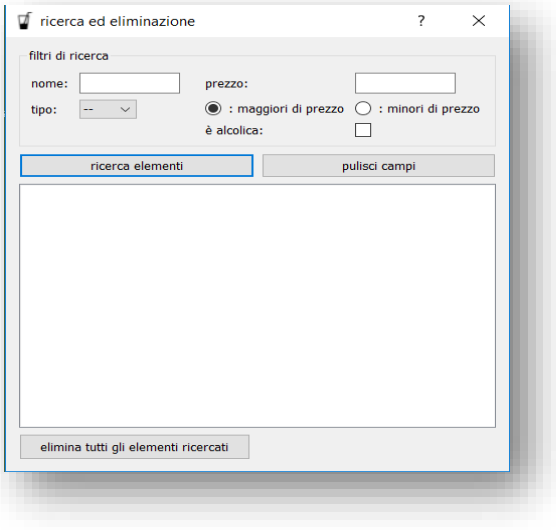


Figura 2: schermata di ricerca

Controller

Il controller permette la comunicazione tra modello e view. Ogni operazione effettuata dall'utente provoca l'attivazione di signal che vengono catturate dagli slot del controller. Per effettuare tali modifiche il controller ha come campi dati un puntatore al modello e ad ogni schermata della view.

Breve Guida all'utilizzo

<u>Per l'inserimento:</u>

All'apertura dell'applicazione si apre la mainwindow di inserimento. Viene richiesto prima di riempire obbligatoriamente tutti i campi presenti nel QGroupBox sulla sinistra. La scelta del campo tipo provocherà l'attivazione di alcuni campi sul QGroupBox di destra che dovranno essere riempiti a loro volta. Il bottone "inserisci" provoca l'inserimento della bevanda nel QListWidget mentre il bottone pulisci riporta alla configurazione di default tutti i QWidget di inserimento.

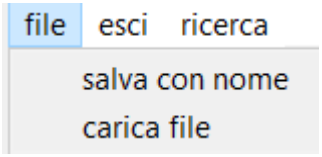
<u>Per le altre interazioni con mainwindow:</u>

I pulsanti nella parte inferiore di mainwindow permettono l'interazione dell'utente con gli elementi presenti nel QListWidget. Premendo con il tasto sinistro su uno degli elementi del QListWidget lo si seleziona. Premendo il pulsante "modifica selezionato" si apre un QDialog che permette all'utente di modificare la bevanda selezionata. Finchè non si chiude il dialog di modifica o si apporta una minima modifica all'elemento selezionato non si potrà in nessun modo interagire nuovamente con mainwindow. Premendo il pulsante "rimuovi elemento" si rimuove l'elemento selezionato. Premendo "calcola totale selezionato" , invece, apparirà un QMessageBox che avvertirà l'utente del costo totale della bevanda selezionata. "Elimina tutto", banalmente, svuota il QListWidget.

Per save e load:

Si deve premere file nella menubar in alta destra

dove si può scegliere se salvare su file o caricare da file.



Per la ricerca:

Premere su ricerca nella menubar. Si aprirà la schermata di ricerca da cui si possono scegliere i filtri su cui basare la ricerca. L'apertura della schermata di ricerca non permette nessuna interazione con la mainwindow con la quale si potrà interagire nuovamente solo nel momento in cui si chiude la schermata di ricerca. È possibile effettuare un'eliminazione di tutti gli elementi ricercati.

Note sulla compilazione

Per la corretta compilazione del progetto è necessario **non eseguire** qmake -project, ma utilizzare la sequenza di comandi qmake => make con il .pro già fornito tra i file consegnati.

Compilatore: Microsoft visual C++ 2015, 32 bit

Sistema operativo: Microsoft Windows 10 Home

Qt creator : versione 4.8.1