

**Alberto Cherobin**

**Matricola: 1122201**

**Anno Scolastico: 2018/19**

# **Progetto Qontainer Programmazione Ad Oggetti “ItemStoreToys”**

## Sommario

Introduzione.....	3
Modalità di consegna.....	3
Descrizioni delle classi e della gerarchia.....	3
Contenitore/Qontainer.....	4
Avvio del Negozio.....	5
Parte gui.....	5
Model Control View.....	6
Modello.cpp e Modello.h.....	6
Controller.cpp e Controller.h.....	6
Estensibilità del codice.....	6
Metodi Polimorfi.....	7
Calcolo delle ore di lavoro.....	7

## Introduzione

Il progetto consiste nel creare un programma che permetta la gestione di un negozio di giochi. Sarà possibile aggiungere, modificare o rimuovere qualsiasi gioco tramite un'interfaccia grafica intuitiva, che comunicherà con il proprietario del negozio tramite appositi messaggi.

L'intero progetto è stato sviluppato su sistema operativo Windows10, utilizzando l'ide QtCreator nella versione 4.8.2 per poi essere successivamente testato nella macchina virtuale di cui è stata fornita l'immagine durante il corso tenutosi nell'anno accademico 2018/2019.

## Modalità di consegna

Il progetto è stato consegnato, come richiesto, tramite comando "consegna progetto-pao-2019". Al suo interno sono presenti tutti i file del progetto compreso il file **.pro** generato automaticamente da Qt all'avvio del progetto. Questo perché la compilazione del progetto necessita di un file (.pro) per qmake, diverso da quello ottenibile tramite l'invocazione di "qmake -project", che permetta la generazione automatica tramite qmake del Makefile.

## Descrizioni delle classi e della gerarchia

La gerarchia del progetto è costituita da una classe base astratta chiamata "ItemStoreToys" che conterrà tutte le caratteristiche che hanno in comune le classi derivate.

Le classi derivate istanziabili di primo livello sono due: "Videogioco" e "GiocoDaTavolo". È presente, nello stesso livello, anche la classe astratta "GiocoDiCarte", da cui derivano altre due classi istanziabili: "GiocoDaTavoloConCarte" e "CarteCollezionabili".

Mi sembra doveroso dare alcune precisazioni: per "gioco da tavolo" si intende un gioco che richiede una ben definita superficie di gioco, che viene detta di solito "tabellone" e sulla superficie vengono solitamente piazzati e/o spostati dei pezzi chiamati "segnalini" o "pedine"; mentre per "gioco da tavolo con carte" si intende un gioco da tavolo basato esclusivamente sull'utilizzo delle carte e in cui non è presente una superficie di gioco. Per esempio, il "Monopoli", è un classificabile come gioco da tavolo perché possiede una propria area di gioco con caselle lungo cui poi le pedine dovranno muoversi. Viceversa, il famoso gioco di carte "Uno" è classificabile, in questa gerarchia, come "gioco da tavolo con carte" in quanto i partecipanti al gioco utilizzano esclusivamente le carte e nient'altro.

La classe base astratta principale contiene le variabili private:

- **nome** del gioco;
- **casa produttrice** del gioco;
- **età** dalla quale è permessa la partecipazione/acquisto del gioco;
- **anno** in cui è stato pubblicato il gioco;
- **prezzo** di vendita;
- **pezzi in magazzino** disponibili;
- il fatto se quella categoria di pezzi di gioco è di tipo "**Usato**" o meno. Nel caso lo fossero il prezzo del lotto di prodotti verrà scalato del 50%. L'usato ha la precedenza sullo sconto;
- la **percentuale di sconto** da applicare sul prodotto. Nel caso il lotto dei prodotti risultasse usato lo sconto non verrà applicato.

La classe "Videogioco" rappresenta un videogioco fisico, per cui custodisce con disco/dischi al suo interno.

Contiene le variabili private:

- **Ps4**, ovvero se il videogioco è stato sviluppato per la piattaforma Playstation 4. È una variabile booleana, per cui il valore "True" significa che il gioco è stato sviluppato per la Playstation, "False" se altrimenti no;
- **XboxOne**, ovvero se il videogioco è stato sviluppato per la piattaforma Xbox One. È una variabile booleana, per cui il valore "True" significa che il gioco è stato sviluppato per la Xbox One, "False" se altrimenti no;
- **genere** di videogioco;
- **contenuto** invece descrive cos'è presente all'interno della custodia di gioco. (Numero di dischi, numero di guide ecc).

La classe "GiocoDaTavolo" contiene le seguenti variabili private:

- **numGiocatori** indica il numero massimo di giocatori che possono partecipare ad una partita;
- **tipologia** del gioco da tavolo;
- **regolamento** del gioco da tavolo;
- **contenuto della scatola di gioco**, inteso come tabellone, numero di pedine presenti ecc.

La seconda classe astratta è la classe “GiocoDiCarte”. Sta a rappresentare i giochi in cui si utilizzano principalmente le carte.

Ha come unico campo privato:

- **edizioneLimitata** che sta ad indicare se il gioco di carte fa parte di una edizione limitata o meno. Essendo anch'essa una booleana il valore “True” starà ad indicare che il gioco è una edizione limitata, altrimenti le verrà assegnato il valore “False”.

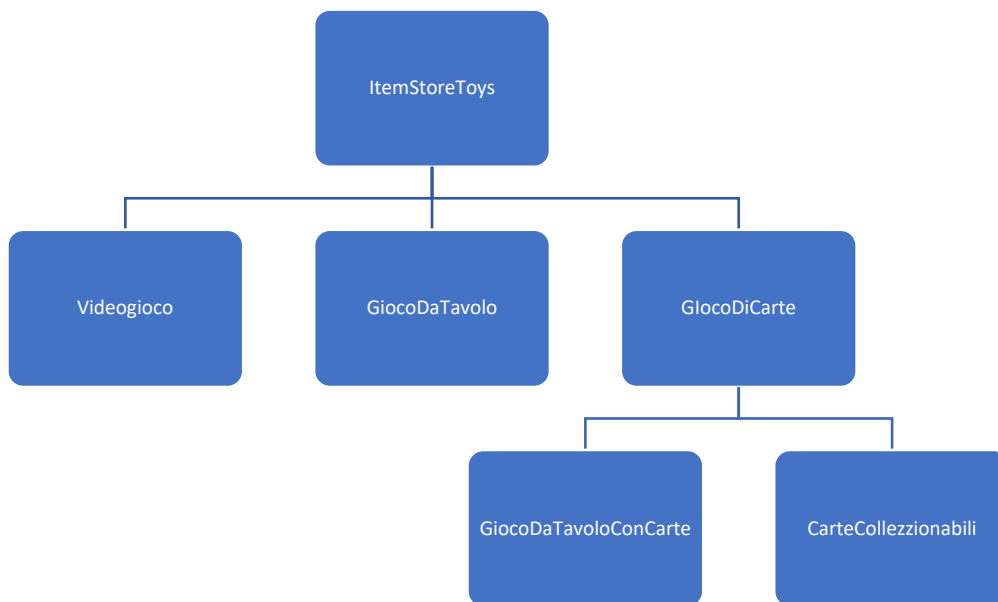
La prima classe istanziabile derivata dalla classe “GiocoDiCarte” è la classe “GiocoDaTavoloConCarte”, la quale possiede i seguenti campi privati:

- **regolamento** del gioco da tavolo;
- **numero di giocatori massimi** che possono partecipare al gioco;
- **contenuto** della scatola di gioco.

La seconda classe istanziabile derivata dalla classe “GiocoDiCarte” è la classe “CarteCollezionabili”, intese come pacchi/deck oppure bustine di carte.

Le sue variabili private sono:

- **numero di carte** contenute all'interno del mazzo/bustina;
- **il nome dell'edizione** a cui appartengono.



## Contentitore/Container

È stato scritto un template di classe che rappresenta un Contentitore che gestisce gli oggetti tramite un'architettura a lista doppiamente collegata. Per cui ogni nodo ha 3 campi, un puntatore al nodo precedente, un puntatore al nodo successivo e il campo che contiene l'informazione del nodo. I puntatori all'interno del nodo non hanno dei puntatori veri e propri ma sono rappresentati da una classe chiamata “SmartP” il cui obbiettivo è quello di prevenire la cancellazione automatica della memoria.

Nella parte privata della classe Container sono presenti due campi di tipo SmartP con nome “primo” ed “ultimo”, essi punteranno rispettivamente al primo SmartP che contiene il primo elemento della lista e all'ultimo smartP che contiene l'ultimo elemento della lista. Ovviamente nel caso in cui ci fosse solo un elemento in lista essi punteranno al medesimo SmartP.

Le funzioni pubbliche principali contenute nella classe Qontainer sono “togliOggetto” e “Ricerca”.

Nel caso in cui il tipo T del parametro formale all'interno della funzione “Ricerca” coincida con un oggetto all'interno della lista, essa ritornerà true, altrimenti false.

Per quanto riguarda la funzione “togliOggetto” prende spunto dalla funzione “togliBolletta” presente nel libro del corso: l'intera lista viene ricopiata nodo per nodo, nel momento in cui viene individuato il nodo con l'oggetto che si desidera togliere, viene effettuato un'erase, e viene restituita la lista senza il nodo che si desiderava togliere.

Vi sono inoltre due funzioni che permettono l'inserimento: “insertFront” e “insertBack” che permettono rispettivamente di inserire un oggetto nella parte iniziale e nella parte finale della lista.

Per quanto riguarda lo scorrimento della lista sono state definite due classi apposite: “iterator” e “constiterator”. Mentre il primo consente lo scorrimento della lista ed eventualmente effettuare modifiche sugli oggetti contenuti dai nodi puntati, il secondo no.

## Avvio del Negozio

All'avvio del progetto viene richiesto di selezionare il file .xml che contiene tutti i dati dell'oggetti contenuti nel negozio. L'xml non è altro che un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

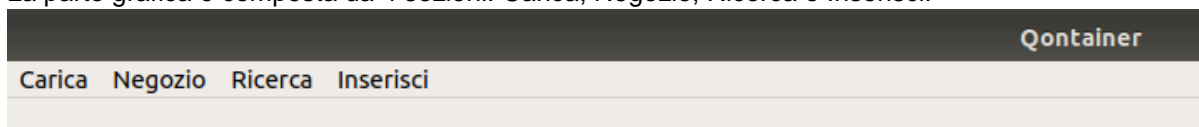
Una volta che il file .xml selezionato dall'utente viene modificato tramite il progetto, con processi di lettura e scrittura, non sarà più possibile andare a modificare direttamente il file, se non tramite l'utilizzo del progetto. Nel caso ci fosse una modifica diretta sul file da parte dell'utente il progetto non sarà più in grado di effettuare la lettura del file, rendendolo inutile. Nell'immagine sottostante ho riportato come ciascun oggetto del negozio venga salvato sul file xml.

```
<Videogioco Nome="Fifa19" CasaProduttrice="Eletronic Arts" Eta="3" AnnoPubblicazione="2018" Prezzo="60" PezziInMagazzino="56" Usato="false" pathImm=":/SalvataggioDati/Immagini/fifa19.jpg" Sconto="0" Ps4="true" XboxOne="true" Genere="Simulatore" Contenuto="1 disco"/>
<GiocoDaTavolo Nome="Cluedo" CasaProduttrice="Hasbro" Eta="3" AnnoPubblicazione="2007" Prezzo="24.9" PezziInMagazzino="90" Usato="false" pathImm=":/SalvataggioDati/Immagini/cluedo.jpg" Sconto="0" NumGiocatori="6" Tipologia="Giallo deduttivo" Regolamento="Indovina l'assassino" Contenuto="1 tabellone, pedine, carte gioco"/>
<CarteCollezionabili Nome="DragonBall" CasaProduttrice="Panini Editore" Eta="3" AnnoPubblicazione="2009" Prezzo="2" PezziInMagazzino="40" Usato="false" pathImm=":/SalvataggioDati/Immagini/dragonball.jpg" Sconto="0" edizioneLimitata="false" NumCarte="13" Edizione="Edizione Oro"/>
<GiocoDaTavoloConCarte Nome="Munchkin" CasaProduttrice="Steve Jackson Games" Eta="14" AnnoPubblicazione="2011" Prezzo="16.5" PezziInMagazzino="100" Usato="false" pathImm=":/SalvataggioDati/Immagini/Munchking.jpg" Sconto="0" edizioneLimitata="false" Regolamento="Sconfiggi tutti i mostri" NumGiocatori="6" Contenuto="2 mazzi di carte"/>
```

Tra i file del progetto è presente anche un **.QRC**. È possibile andarlo a definire direttamente tramite Qt Creator e grazie ad esso il progetto sarà in grado di avere un punto di riferimento per l'inizio dei path/cammini per andare a prendere le immagini dei prodotti.

## Parte gui

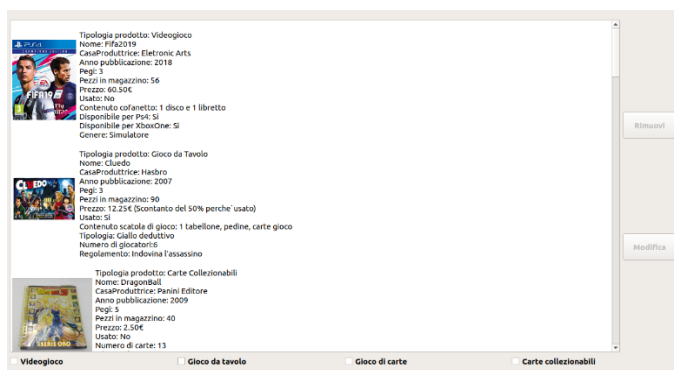
La parte grafica è composta da 4 sezioni: Carica, Negozio, Ricerca e Inserisci.



### Carica

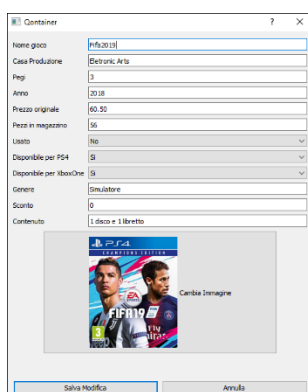
Questa opzione del menù permette di inserire un nuovo file (.xml) all'interno del negozio. Nel caso si volesse cambiare il file xml oppure ci si fosse dimenticati di caricarlo all'inizio.

### Negozio



È la pagina principale del gestore, in cui è possibile visualizzare tutti i prodotti presenti all'interno del negozio. Tramite l'utilizzo dei due tasti posti sul lato destro si può rimuovere e modificare gli oggetti dopo averli accuratamente selezionati.

È stato aggiunto un filtro nella parte inferiore dell'interfaccia in modo tale che si possa visualizzare solo la categoria di oggetti che si preferiscono.

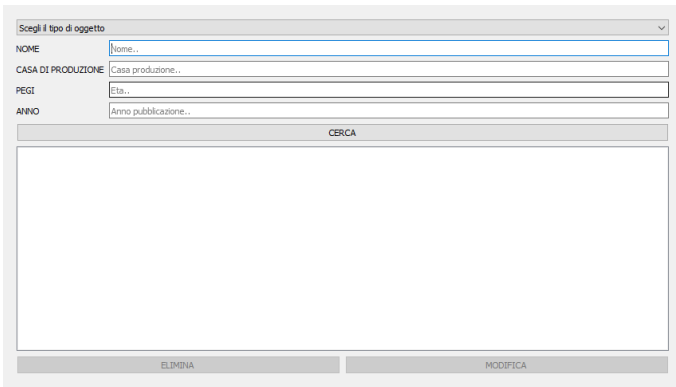


### Modifica

La modifica dei prodotti è affidata a delle classi di tipo QFileDialog. Ognuna di esse non è altro che una finestra di livello superiore utilizzata principalmente per attività a breve termine e comunicazioni brevi con l'utente.

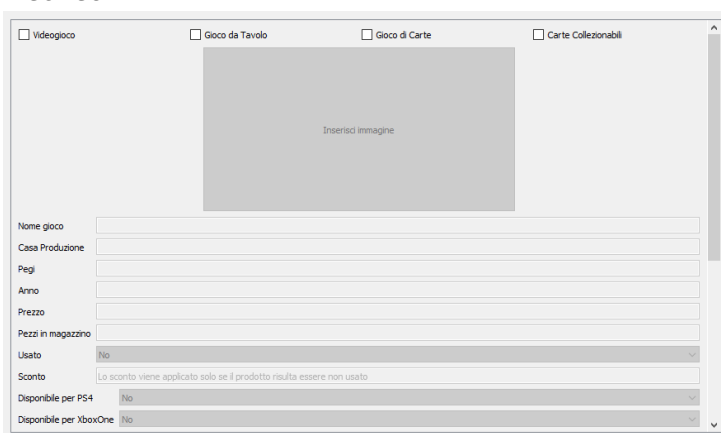
In questo caso quest'ultimo può modificare praticamente qualsiasi campo dato a proprio piacimento. Ovviamente alcuni campi sono limitati ad un determinato numero di valori che possono assumere, per cui l'utente dovrà scegliere l'opzione di suo piacimento tramite un comodo menu a tendina.

## Ricerca



parte inferiore della schermata. La compilazione dei 4 campi è obbligatoria, per cui se l'utente dimentica di compilarne uno gli verrà comunicato tramite messaggio e ovviamente è altrettanto importante selezionare il tipo di oggetto che si vuole cercare tramite l'utilizzo del menù a tendina posto nella parte superiore dell'interfaccia. Anche in questo caso, se l'utente dimentica di selezionarlo, gli viene immediatamente comunicata la cosa.

## Inserisci



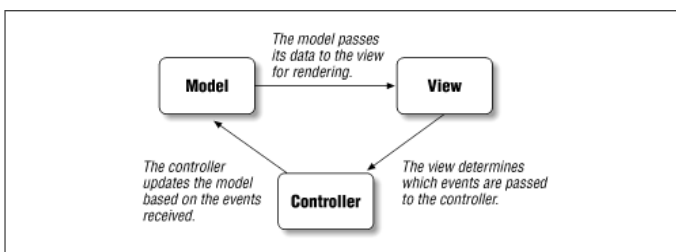
La terza e ultima sezione permette di inserire nuovi prodotti all'interno del gestore del negozio. I campi di ogni oggetto dovranno essere tutti compilati, nel caso ciò non venga fatto si verrà sempre avvisati da un QMessageBox che inviterà a compilare tutti i campi.

Il pulsante "Inserisci" è stato inserito alla fine della scrollArea per obbligare l'utente a scrollare verso il basso in modo tale da visualizzare tutti i campi per poi andare a compilarli.

I campi da compilare si sbloccheranno dopo aver spuntato una tra le tipologie di oggetto presenti nella QComboBox posta nella parte superiore della pagina.

## Model Control View

Per la progettazione architetturale della gui ho aderito al design pattern "Model-View-Controller":



- il model fornisce i metodi per accedere ai dati utili all'applicazione;
- il view visualizza i dati contenuti nel model e si occupa dell'interazione con l'utente;
- il controller riceve i comandi dell'utente e li attua modificando lo stato degli altri due componenti.

## Modello.cpp e Modello.h

Le funzioni principali utilizzate nel modello sono Salvataggio e Caricamento, le quali sono le principali responsabili della lettura e della scrittura del file .xml, all'interno dei quali vengono salvati i dati del gestore. Esse vengono richiamate sempre in caso di modifica, rimozione o aggiunta di un prodotto.

## Controller.cpp e Controller.h

Il flusso delle informazioni principali passa attraverso di esso. È il responsabile del prelievo di tutti i dati dei nuovi prodotti e delle modifiche che vengono apportate all'interno del gestore.

## Estensibilità del codice

Tutto il progetto è stato separato tra dichiarazione e definizione di metodi e funzioni come richiesto dalle specifiche.

Anche la parte grafica è stata separata su più file: le sezioni Negozio, Ricerca e Inserimento sono state dichiarate e definite su 3 file diversi in modo tale da permettere in futuro modifiche rapide e pulite.

L'ereditarietà utilizzata nel progetto può essere usata come meccanismo per ottenere l'estensibilità e il riuso del codice, e risulterà particolarmente vantaggiosa quando verrà usata per definire altri sottotipi, sfruttando le relazioni is-a. Oltre all'evidente riuso del codice della superclasse, l'ereditarietà permette la definizione di codice generico attraverso il meccanismo del polimorfismo.

## Metodi Polimorfi

I metodi virtuali utilizzati nella mia gerarchia sono:

- "getContenuto" : serve a restituire il contenuto della custodia di gioco per la classe "Videogioco" e il contenuto della scatola di gioco per le classi "GiocoDaTavolo" e "GiocoDaTavoloConCarte". Mentre per la classe "CarteCollezionabili" restituisce il numero di carte contenuto nella bustina o mazzo di carte;
- "getTipo" : per ogni oggetto restituisce tramite stringa a quale tipologia di classe appartiene;
- "infoOggetto" : viene utilizzato per la stampa a video di tutti i campi degli oggetti all'interno della Gui;
- overloading dell'operatore di uguaglianza: utilizzato principalmente nella sezione ricerca per effettuare il confronto tra gli oggetti. Il confronto tra gli oggetti si focalizza solamente su 4 campi dati: Nome, Casa produttrice, Pegi ed Anno di pubblicazione;
- distruttore della principale classe astratta.

## Calcolo delle ore di lavoro

- Analisi specifiche del progetto: 1 ora
- Pianificazione gerarchia e Container : 2 ore
- Scrittura Container e Gerarchia: 6 ore
- Apprendimento parte grafica ide Qt: 15 ore
- Scrittura parte grafica (GUI): 23 ore
- Debugging: 4 ore
- Stesura relazione: 2 ore

Totale ore sviluppo progetto: 53 ore