Galtarossa Marco
Matricola: 1096393

Relazione del progetto di programmazione ad oggetti anno 2020/2021

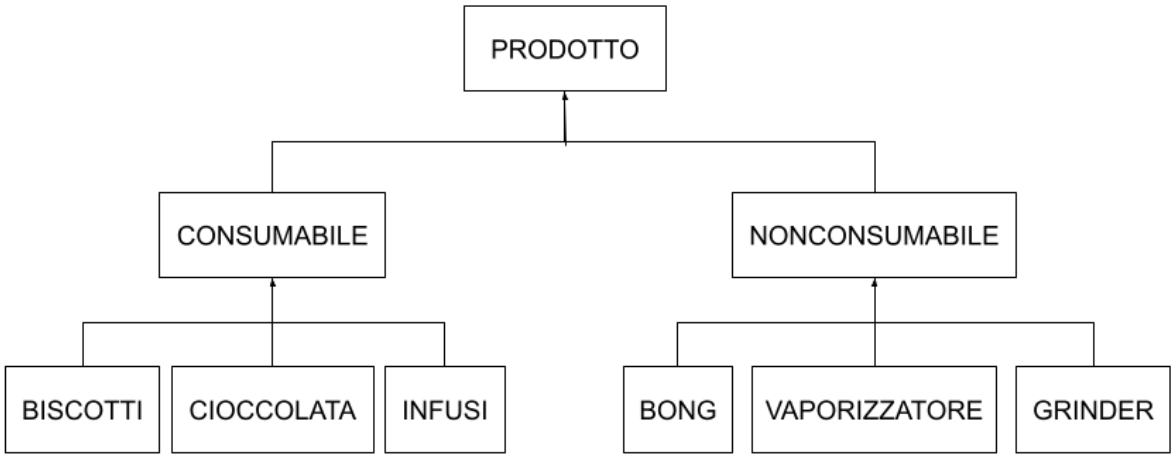
“ChartisLair”

Sommario

1 – Introduzione

Il progetto consiste nella realizzazione di un programma per la gestione di un distributore di prodotti a base di CBD e di accessori per fumatori. Tramite l'applicazione sarà possibile visualizzare la lista dei prodotti presenti nel distributore con relativo prezzo e guadagno, effettuare una ricerca, modificare, eliminare ed inserire un prodotto, ricevere informazioni importanti riguardanti il catalogo dei prodotti e lo sviluppatore. Il progetto è stato sviluppato su sistema operativo Windows 7 utilizzando l'ide Qt Creator versione 4.11.0 basata sulla versione di Qt 5.14.2. Successivamente il lavoro è stato testato nella macchina virtuale fornita dal professore.
Nella cartella zip da consegnare per la valutazione sono presenti tutti i file del progetto compreso quello della relazione e il file *.pro* generato da Qt all'avvio del progetto in quanto la compilazione richiede un file di questo tipo per qmake differente da quello ottenibile tramite l'invocazione di "qmake -project" da terminale, che permetta la generazione automatica tramite qmake del Makefile. Ho notato un bug nell' apertura del progetto sulla VM: se non si sceglie di usare come cartella di destinazione della build quella del progetto, Qt creerà in automatico una cartella di build esterna. Questo comporta che le immagini e il file xml, dove sono salvati i prodotti, non vengono letti da Qt, Il programma è comunque operativo nelle sua funzioni. Se si vuole visualizzare il programma nella maniera appropriata bisogna cambiare, nelle impostazioni del progetto di Qt, la cartella di destinazione della build, facendola coincidere con quella del progetto.

2 – Descrizione della gerarchia



La gerarchia è costituita da una classe base astratta **prodotto**. Le classi figlie dirette sono **consumabile** e **nonConsumabile**. In fondo alla gerarchia sono presenti le sei classi concrete **biscotti**, **cioccolata**, **infusi**, **bong**, **vaporizzatore**, **grinder**: le prime tre derivano da **consumabile**, le seconde tre da **nonConsumabile**.
La classe base astratta prodotto contiene i seguenti campi dati privati:

- **nome,** variabile di tipo *string* che indica il nome del prodotto;
- **confezioneRegalo**, variabile di tipo *boolean* che indica con *true* se il prodotto è in una confezione regalo e con false l'opposto.

La classe consumabile, derivata da prodotto, contiene i seguenti campi dati privati:

- **ingredienti,** variabile di tipo *vector di string* che indica l'insieme degli ingredienti che possiede il prodotto. L'utente non può modificarli;
- **peso,** variabile di tipo *double* che indica il peso del prodotto;
- **erba,** variabile di tipo *boolean* che indica con *true* se l'inflorescenza è indica, altrimenti sarà sativa;
- **produzione,** variabile di tipo *boolean* che indica con *true* se le inflorescenze sono coltivate in serra (indoor), con *false* se sono coltivate su campi all'esterno (outdoor).

La classe nonConsumabile, derivata da prodotto, contiene i seguenti campi dati privati:

- **colori,** variabile di tipo *string* che indica il colore del prodotto.

La classe biscotti, derivata da consumabile, contiene i seguenti campi dati privati:

- **tipoFarina,** variabile di tipo *enum* che indica la tipologia di farina utilizzata per la produzione di biscotti e può assumere valore: 1 se di grano, 2 se di riso, 3 se di mandorle, 4 se di castagne, 5 se di amaranto;
- **gocceCioccolata,** variabile di tipo *enum* che indica la tipologia di gocce di cioccolato utilizzata per la produzione di biscotti e può assumere valore: 1 se di latte, 2 se fondente, 3 se bianco, 4 se è senza gocce.

La classe cioccolata, derivata da consumabile, contiene i seguenti campi dati privati:

- **tipoGranella,** variabile di tipo *enum* che indica la tipologia di granella della cioccolata prodotta e può assumere valore: 1 se di cocco, 2 se di noce, 3 se di mandorla, 4 se di nocciola;
- **livelloFodenza,** variabile di tipo *enum* che indica la percentuale di cacao presente nella cioccolata prodotta e può assumere valore: 1 se 0% (niente, ovvero al latte), 2 se 50% (bassa), 3 se 75% (media), 4 se 90% (alta);
- **stecca,** variabile di tipo *boolean* che indica con *true* se la forma del cioccolato e a stecche, e con *false* se è a pralina.

La classe infusi, derivata da consumabile, contiene i seguenti campi dati privati:

- **aroma,** variabile di tipo *vector di string* che indica il gusto, o gusti (massimo due) presenti nell' infuso;
- **sfuso,** variabile di tipo *boolean* che indica con *true* se l'infuso è venduto in forma sfusa, con *false* se venduto in bustine.

La classe bong, derivata da nonConsumabile, contiene i seguenti campi dati privati:

- **backer,** variabile di tipo *boolean* che indica con *true* se la forma del bong è a backer, con *false* se la forma è dritta;
- **altezza,** variabile di tipo *enum* che indica le varie altezze delle varie tipologie di bong. In base alla forma ci sono altezze diverse. Per ogni altezza corrisponde una larghezza del braciere. La variabile altezza può assumere valore: 1 se alto 24cm, 2 se alto 32cm, 3 se alto 37cm, 4 se alto 46cm, 5 se alto 33cm, 6 se alto 45cm;
- **larghezza,** variabile di tipo *double* che indica la larghezza corrispondente all' altezza.

La classe vaporizzatore, derivata da nonConsumabile, contiene i seguenti campi dati privati:

- **velocitaEvaporazione,** variabile di tipo *enum* che indica la velocita di evaporazione e può assumere valore: 1 se di livello1, 2 se di livello2, 3 se di livello3;
- **capienza,** variabile di tipo *int* che indica la capienza massima dello sportello vaporizzatore;
- **schermo,** variabile di tipo *boolean* che indica con *true se* presente lo schermo, *false* altrimenti.

La classe grinder, derivata da nonConsumabile, contiene i seguenti campi dati privati:

- **ndenti,** variabile di tipo *int* che indica il numero di denti del trita tabacco;
- **raccoglipolline,** variabile di tipo *boolean* che indica con *true* se presente il raccogli polline, *false* altrimenti.

3 - Descrizione del container

Per la realizzazione del progetto sono stati scritti due template di classe: il primo di nome deepPtr, che rappresenta l'equivalente dei puntatori polimorfi al tipo T che implementa una gestione automatica della memoria in modalità "profonda", mentre il secondo di nome lista, che rappresenta un contenitore che gestisce gli oggetti con un'architettura a lista doppiamente linkata. Il template deepPtr è formato da una variabie di tipo T* e implementa un costruttore, il costruttore di copia profonda, implementato con il metodo clone, l'assegnazione profonda, e il distruttore e infine la ridefinizione dell' operatore ==. Nel template lista i puntatori contengono una classe nodo che ha tre campi dati privati: **info,** che contiene l'oggetto della gerarchia, **prev** , puntatore di tipo *nodo** che punta al nodo precedente se presente oppure a *nullptr* , e **next** , puntatore di tipo *nodo** che punta al nodo successivo se presente oppure a *nullptr* . Di conseguenza i nodi della lista del primo template sono gestiti tramite dei puntatori di tipo *nodo,* per ottenere una gestione controllata della memoria. Inoltre, il template lista contiene due campi dati privati di tipo *nodo:* **primo,** che indica il primo nodo della lista, ed **ultimo,** ossia l'ultimo nodo della lista. Nel caso ci sia un solo elemento nella lista questi due campi privati coincideranno, e nel momento in cui la lista viene riempita questi campi verranno aggiornati di conseguenza. Per scorrere la lista sono state definite due classi di iteratori: iteratore e iteratoreConst, la prima permette lo scorrimento della lista con eventuali possibili modifiche mentre la seconda non ammette modifiche. I metodi rilevanti del template lista sono in totale quattro:

- **aggiungiCoda,** metodo che crea un nuovo nodo con campo info pari all'oggetto passato come parametro e lo aggiunge alla lista ponendolo come ultimo nodo;
- **aggiungiTesta,** metodo che crea un nuovo nodo con campo info pari all'oggetto passato come parametro e lo aggiunge alla lista ponendolo come primo nodo;
- **togliUno,** metodo che rimuove dalla lista il nodo contenente l'oggetto passato come parametro. Per scrivere questo ho preso spunto dal relativo metodo **Togli_Telefonata** del libro di Programmazione ad Oggetti;
- **vuoto,** metodo che ritorna un valore di tipo *boolean* che indica con true se la lista in questione è vuota, mentre con false l'opposto.

La lista è stata inizializzata a **lista<deepPtr<prodotto>>.**

4 - Polimorfismo ed estensibilità del codice

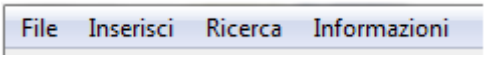
I metodi polimorfi utilizzati nella gerarchia sono i seguenti:

- **distruttore** della classe astratta;
- **operator==,** ridefinizione dell'operatore di uguaglianza utilizzato per controllare la presenza di doppioni all'inserimento di un nuovo oggetto nella lista;
- **prezzo,** ritorna una variabile di tipo *double* contente il prezzo calcolato in base a determinate caratteristiche del prodotto;
- **ricavo,** ritorna una variabile di tipo *double* il ricavo del venditore calcolato come una percentuale sul prezzo;
- **tipoElemento,** ritorna una variabile di tipo *string* che rappresenta la tipologia di prodotto;
- **visualizzaInfo,** ritorna una variabile di tipo *string* che rappresenta tutte le informazioni e le caratteristiche di un determinato prodotto;
- **clone,** ritorna l'oggetto copiato.

Per assicurare l'estensibilità del codice per tutti i metodi, i costruttori e gli operatori sono state separate la dichiarazione e la definizione come richiesto dalle specifiche. Questo meccanismo è stato utilizzato anche per la parte grafica del progetto, inoltre per quest'ultima sono state creati diversi file per la creazione della GUI in modo da ottenere il codice più chiaro possibile.

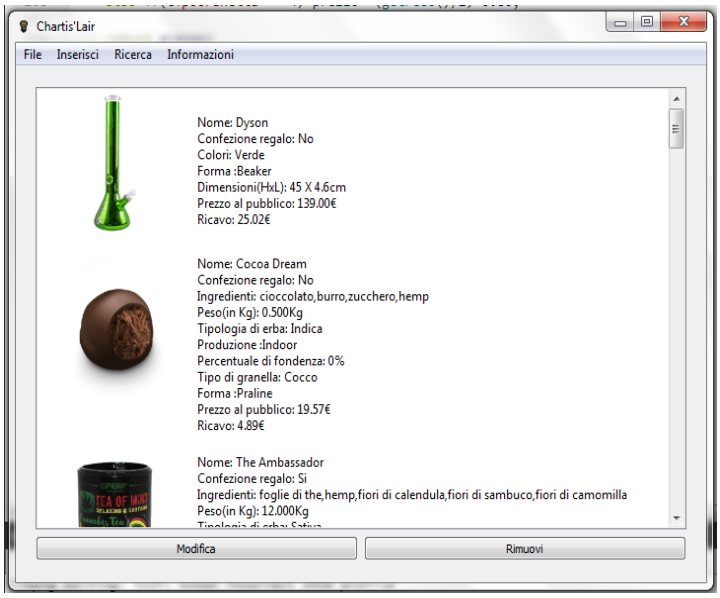
5 - Descrizione della GUI

La GUI si presenta con una dimensione minima impostata, questo perché se ridotta le parole scomparirebbero. Il menù è diviso del seguente modo:

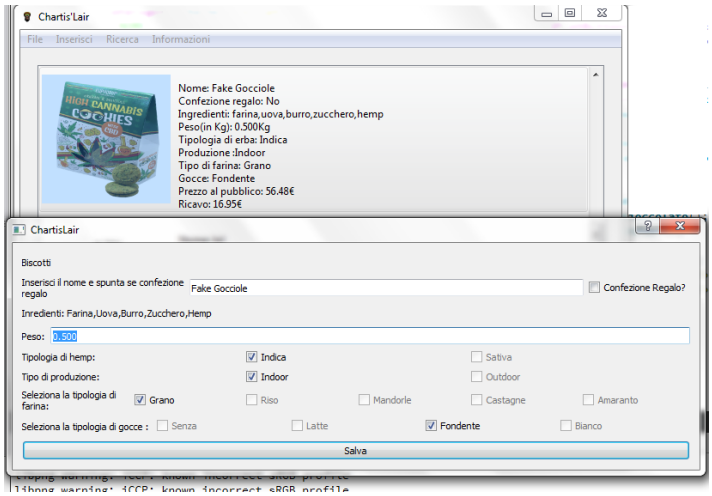


- **File:** dove si può tornare alla home quando si è in altre sezioni, caricare e salvare un file xml,e uscire dall'applicativo;
- **Inserisci:** pagina di inserimento di un nuovo prodotto in catalogo;
- **Ricerca:** pagina di ricerca di un prodotto nel catalogo;
- **Informazioni:** Si può scegliere se visualizzare le informazioni sul catalogo, cioè la quantità di ogni prodotto disponibile nel catalogo. Oppure si può scegliere di visualizzare le informazioni sullo sviluppatore dell'applicativo.

5.1 - Home

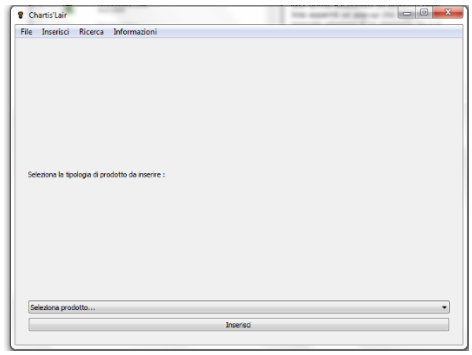


Il catalogo viene rappresentato sotto forma di lista, dove si potrà vedere l'immagine ed un elenco ordinato di tutte le informazioni presenti per ogni prodotto. Sotto la lista dei prodotti ci sono due pulsanti: *Modifica* e *Rimuovi*. Se si clicca su uno dei due senza aver prima selezionato un prodotto dalla lista apparirà un pop-up che segnalerà la mancata selezione di un elemento. Se si è selezionato un elemento e si preme il tasto *Rimuovi* il prodotto verrà rimosso dal catalogo, apparirà un pop up che ti mostrerà lo stato delle scorte del prodotto e successivamente un altro pop up che confermerà l'avvenuta eliminazione.

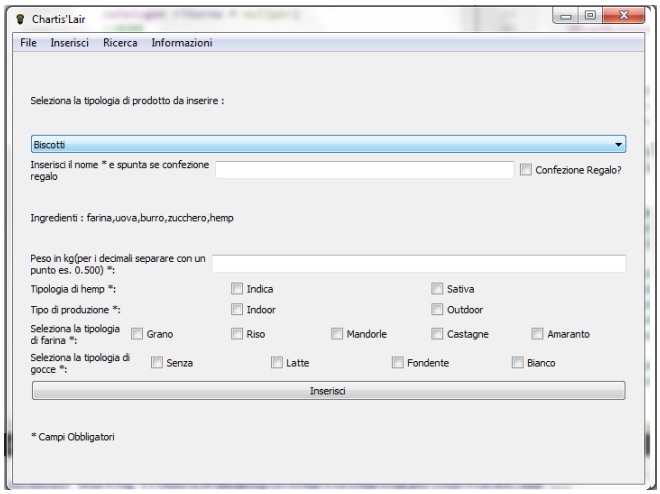


Se si è selezionato un elemento e si preme il tasto *Modifica*, apparirà un pop up con elencate le proprietà del prodotto selezionato e si darà possibilità di modificare alcune caratteristiche. La schermata è un esempio se si scegli di modificare un prodotto biscotto

5.2 - Inserisci

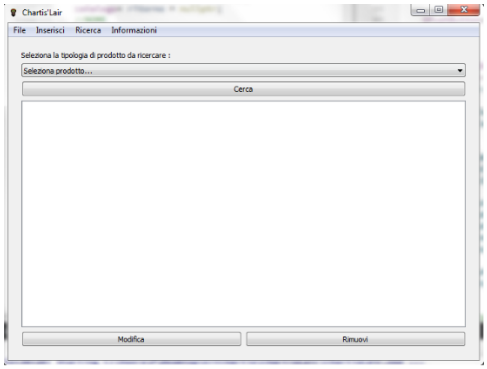


Se l'utente vuole aggiungere al catalogo un nuovo prodotto, selezionerà il pulsante Inserisci e si troverà in questa pagina. Da questa pagina è possibile selezionare la tipologia di prodotto da inserire e si può scegliere tra: Biscotti, Cioccolata, Infusi, Bong, Vaporizzatore, Grinder.
Una volta selezionato la tipologia del prodotto la finestra si espanderà consentendo di scegliere le caratteristiche del nuovo prodotto da inserire.
Se si preme *Inserisci* senza aver selezionato nessun prodotto, apparirà un pop up di errore.

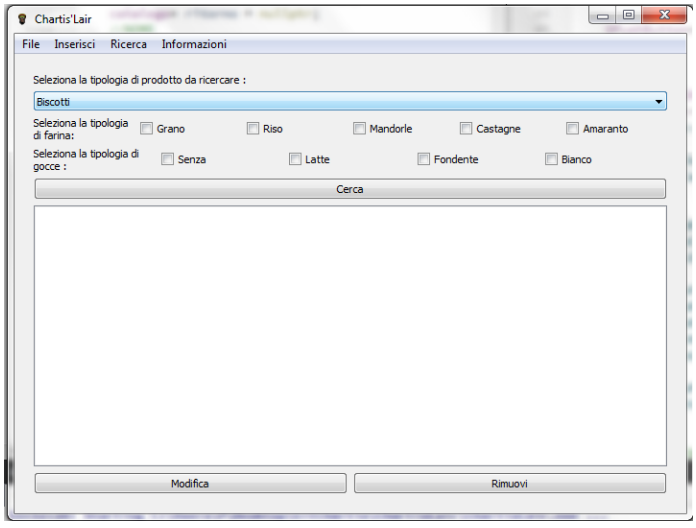


Questa è una schermata dove per esempio scelgo di aggiungere dei biscotti.
Una volta compilati e selezionati tutti i campi, si preme il bottone *Inserisci* posto in fondo alla pagina. Se non si inseriscono tutti i dati contrassegnati con l'asterisco, apparirà un pop up di errore. Ogni volta che viene selezionata una scelta nella checkbox, l'altra o le altre, appartenenti alla stessa categoria, vengono disattivate. Per riattivarle basta deselezionare la checkbox scelta. Ciò si applica a tutte le checkbox presenti nell' applicativo.

5.3 - Ricerca



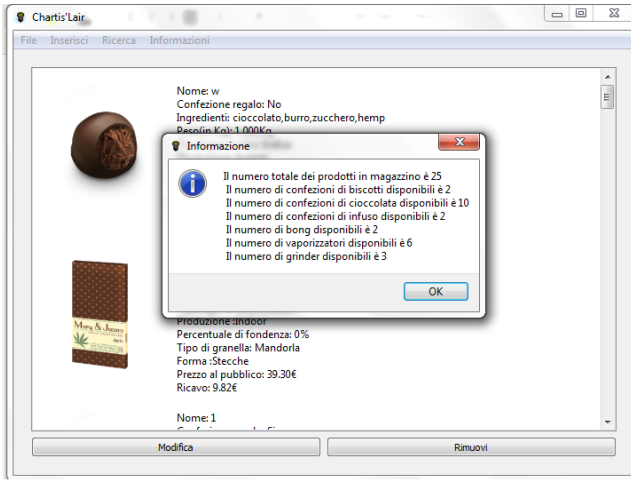
Se l'utente vuole ricercare un prodotto nel catalogo premerà il pulsante *Ricerca*, e si troverà in questa pagina.
Dalla pagina, potrà selezionare una categoria di prodotto da ricercare, si può scegliere tra: Biscotti, Cioccolata, Infusi, Bong, Vaporizzatore, Grinder. Se si preme *Cerca* senza aver selezionato nessun prodotto, apparirà un pop up di errore.
Per cercare un prodotto bisogna perciò selezionare la tipologia e appena viene effettuata la selezione, la finestra si espanderà permettendo di filtrare la ricerca con le caratteristiche di un determinato prodotto.



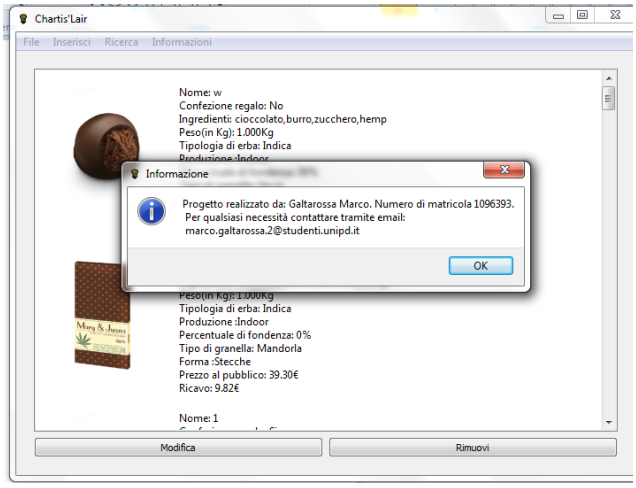
In questa schermata per esempio ho selezionato i biscotti come prodotto da ricercare, e si sono aggiunti due filtri tramite i quali è possibile fare una ricerca più precisa. Per effettuare una ricerca non occorre inserire o contrassegnare ogni campo. Prendendo come esempio i biscotti, si può: visualizzare tutti i biscotti nel catalogo, visualizzare i biscotti filtrandoli per una determinata farina oppure per determinate gocce di cioccolato, visualizzare i biscotti filtrandoli per farina o gocce. In fondo alla schermata ci sono due pulsanti: *Modifica* e *Rimuovi*. Se si clicca su uno dei due senza aver prima selezionato un prodotto dalla lista apparirà un pop-up che segnalerà la mancata selezione di un elemento.
Se si è selezionato un elemento e si preme il tasto *Rimuovi* il prodotto verrà rimosso dal catalogo, apparirà un pop up che ti mostrerà lo stato delle scorte del prodotto e successivamente un altro pop up che confermerà l'avvenuta eliminazione.
Se si è selezionato un elemento e si preme il tasto *Modifica*, apparirà un pop up con elencate le proprietà del prodotto selezionato e si darà possibilità di modificare alcune caratteristiche. Questi due pulsanti funzionano analogamente a quelli presenti nella Home.

5.4 - Informazioni

I pop up di informazione sono due: *Informazioni Catalogo* o *Informazioni Sviluppatore.* Nel primo si trovano le informazioni riguardo la quantità dei singoli prodotti presenti nel catalogo, nel secondo troviamo le informazioni sullo sviluppatore dell'applicativo.

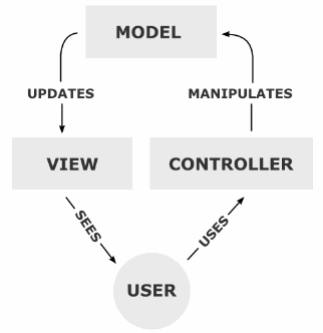


Informazioni Catalogo



Informazioni Sviluppatore

6 - Model View Controller



Per la progettazione della GUI ho aderito al pattern Model-View-Controller (MVC), pattern che separa la logica di presentazione e la logica applicativa. MVC è basato sulla separazione dei compiti fra i componenti software che hanno tre ruoli principali:

1. *model* che fornisce i metodi per accedere ai dati dell'applicazione;
2. *view,* visualizza i dati presenti nel model e si occupa dell'interazione;
3. *controller,* riceve i comandi dall'utente e li concretizza modificando lo stato dei primi due componenti.

7 - Input/Output

All'avvio del progetto si aprirà una finestra di sistema che chiede di selezionare un file di tipo *xml* il quale verrà letto e in questo modo verrà caricata la lista dei prodotti nel catalogo. Se non si seleziona alcun file verrà avviata l'applicazione con i comandi legati alla lista dei prodotti disabilitati in quanti non è presente alcuna lista su cui lavorare, per poter utilizzare appieno l'applicazione basterà caricare un file *xml* tramite il comando **Carica file** presente come sottovoce della voce **File** del menu del programma. Nel momento in cui si salvano le modifiche di un prodotto, si rimuove un prodotto oppure si sceglie di salvare dal menu principale dell'applicazione, verrà salvato un file di tipo *xml* contenente le informazioni aggiornate dei prodotti.

8 - Conteggio delle ore

Per la realizzazione dell'intero progetto e la stesura della relazione sono state impiegate in totale 60 ore, valore superiore al dovuto a causa della realizzazione della parte grafica e al debugging. Inoltre, ulteriori 15 ore (segnate in rosso nello riepilogo sottostante) per la correzione degli errori riscontrati nella consegna precedente. Di seguito la suddivisione delle ore in base alle aree tematiche prese in considerazione:

- Analisi delle specifiche del progetto: 1 ora;
- Panificazione e Scrittura della gerarchia e del container: 15 ore; 10 ore;
- Apprendimento parte grafica ide Qt: 15 ore;

- Scrittura della parte grafica (GUI): 22 ore;
- Debugging: 5 ore; 4 ore;
- Stesura della relazione: 2 ore; 1 ora;

9 – Elementi revisionati

Rispetto alla precedente consegna il progetto è cambiato radicalmente nella gestione dei deepPtr.
Prima non era stata fatta una copia profonda come richiesto dalle specifiche e inoltre la lista era inizializzata in maniera errata. Ovviamente questa modifica ne ha prodotte altre di piccole a cascata, relative principalmente al passaggio degli oggetti. Un'altra modifica apportata è stata la divisione della gerarchia, mentre prima era definita tutta in un unico file. La classe base astratta precedentemente si chiamava catalogo, ma effettivamente, il suo nome non era calzante. Perciò il nome è stato cambiato in prodotto. Altro nome cambiato è stata **forma** della classe Bong e Cioccolata, anche in questo caso il nome non indicava la funzionalità, perciò il nome è stato cambiato in **stecca** e **backer**.