Relazione progetto PharmaCharts - a.a 2021/2022

Anna Cisotto Bertocco
matricola: 1170706

1 Abstract

PharmaCharts è un'applicazione che permette di visualizzare grafici (*charts*) a partire da un dataset di farmaci venduti in provincia di Padova L'applicazione offre la possibilità di creare e modificare il dataset e di visualizzare quattro diverse tipologie di grafici (a barre, a torta, a linea e a dispersione) per poter confrontare diverse variabili contenute nel dataset. L'applicazione offre inoltre la possibilità di caricare e salvare il dataset da/su un file XML.
I grafici sono stati creati utilizzando il modulo Qt QChart. L'applicazione è stata sviluppata usando il framework Qt nella versione 5.9.5 per il sistema operativo macOS Monterey 12.0.1 e con compilatore clang 13.0.0.

1.1 Istruzioni di compilazione

Per compilare correttamente l'applicazione è necessario usare i comandi

```
qmake -project "QT += charts"
qmake
make
```

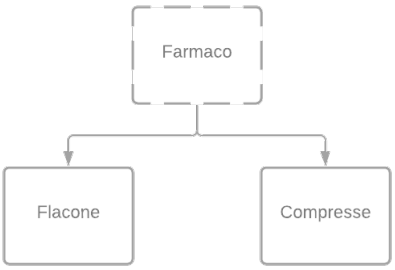
Successivamente, per poter eseguire l'applicazione, è sufficiente usare il comando

```
./PharmaCharts
```

2 Progettazione

2.1 Gerarchie di classi

L'applicazione PharmaCharts utilizza cinque diverse gerarchie di classi.

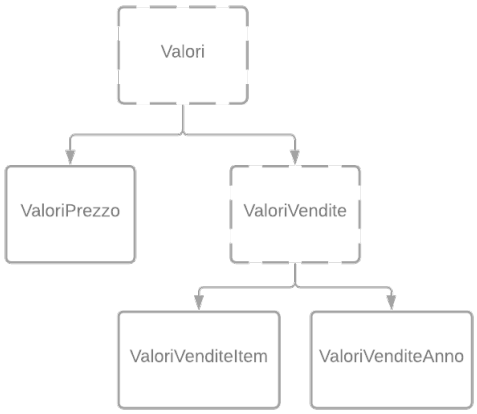


La prima gerarchia viene usata per rappresentare i tipi di farmaco usati nell'applicazione, ovvero Flacone o confezione di Compresse. La classe base astratta **Farmaco** rappresenta un generico farmaco caratterizzato da *nome*, *marca* e dalle *vendite* dal 2018 al 2021, gestite tramite un vettore. La classe inoltre definisce i seguenti metodi virtuali puri:

- **double calculatePrice()**, che calcola il prezzo finale del farmaco in base al suo tipo
- **std::string getType()**, che ritorna una stringa con il tipo di farmaco

- **Farmaco* clone()**, ossia un metodo di clonazione
- **bool operator==(const Farmaco& f)** e **bool operator!=(const Farmaco& f)**, ossia l'overloading degli operatori di uguaglianza e disuguaglianza

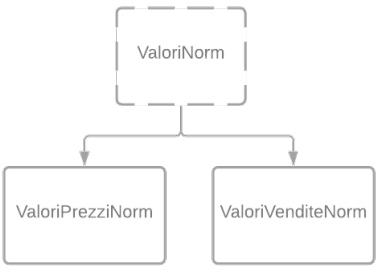
La classe **Flacone** deriva direttamente dalla classe Farmaco e rappresenta un farmaco prodotto in forma di flacone. Ogni flacone si caratterizza per la quantità di prodotto in ml e per il prezzo al ml; il prezzo finale viene calcolato moltiplicando gli ml totali per il prezzo al ml.
La classe **Compresse**, che deriva anch'essa direttamente dalla classe Farmaco, rappresenta invece un farmaco venduto in confezioni di compresse; ogni confezione di compresse è caratterizzata dal numero di compresse della confezione e dal prezzo a compressa, e il prezzo finale del farmaco viene calcolato moltiplicando il numero di compresse per il prezzo a compressa.
Per ciascuna classe della gerarchia sono stati inoltre implementati i metodi getter e setter per tutti i campi dati.



La seconda gerarchia è stata usata per gestire i possibili valori per generare un grafico. La classe base astratta **Valori** rappresenta i valori grezzi presi dal dataset e definisce i seguenti metodi virtuali puri:

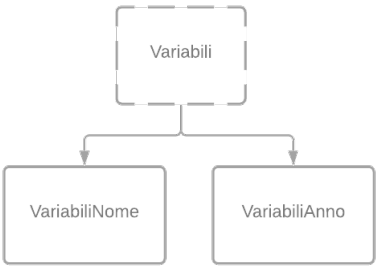
- **int getSize()**, che calcola la dimensione del vettore che gestisce i valori
- **bool isEmpty()**, che ritorna true sse la dimensione del vettore dei valori è 0

La classe **ValoriPrezzo**, che deriva direttamente da Valori, è caratterizzata da un vettore contenente i prezzi di tutti i farmaci presenti nel dataset.
Le classi **ValoriVenditeAnno** e **ValoriVenditeItem** derivano dalla classe astratta **ValoriVendite** e rappresentano rispettivamente i valori delle vendite di tutti i farmaci del dataset in un determinato anno e i valori delle vendite di un determinato farmaco dal 2018 al 2021. Per tutte queste classi concrete sono stati inoltre implementati l'overloading dell'operatore di indicizzazione e un metodo che ritorna il valore massimo assunto dalle varie classi di valori.

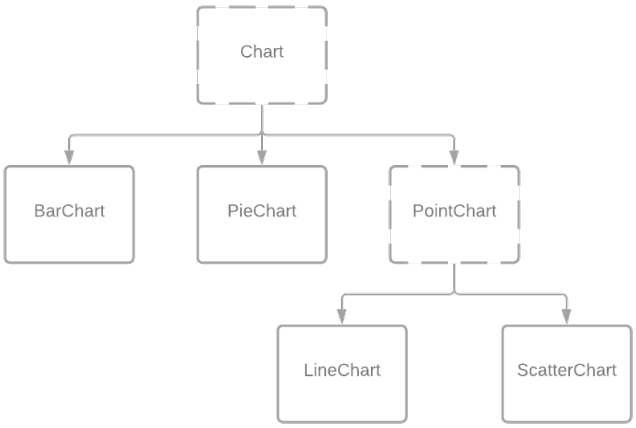


La terza gerarchia è stata utilizzata per trasformare i dati grezzi contenuti nella gerarchia dei Valori

in valori normalizzati per poterli visualizzare in maniera più compatta nei grafici. Le classi concrete **ValoriPrezziNorm** e **ValoriVenditeNorm**, che derivano dalla classe astratta **ValoriNorm**, rappresentano i valori dei prezzi e delle vendite normalizzati.



La quarta gerarchia è stata utilizzata per gestire i vari tipi di variabili da usare nei grafici. La classe base astratta **Variabili** è caratterizzata da un vettore contenente le variabili, mentre le classi concrete **VariabiliNome** e **VariabiliAnno** rappresentano rispettivamente i nomi di tutti i farmaci contenuti nel dataset e gli anni presi in considerazione per le vendite (2018, 2019,2020,2021).



La quinta ed ultima gerarchia è stata usata per gestire i vari tipi di grafico usando i valori normalizzati presi dal dataset. La classe base astratta **Chart** rappresenta un generico grafico caratterizzato da un *titolo*, dall'insieme di *variabili* e dall'insieme dei *valori* ed implementa i seguenti metodi virtuali puri:

- **std::string getChartType()**, che ritona una stringa con il tipo di grafico
- **QChart* createChart()**, che ritorna un QChart costruito in base al tipo di grafico

La classe concreta **BarChart** rappresenta un grafico a barre; i valori vengono gestiti tramite le QBarSeries mentre le variabili tramite una QStringList. La classe concreta **PieChart** rappresenta invece un grafico a torta, in cui i valori e le variabili vengono gestiti tramite le QPieSerires. La classe **PointChart** è una classe astratta che rappresenta una grafico che utilizza una serie di coordinate di punti (QPointF) per poter essere costruito; da questa classe derivano le classi concrete **LineChart** e **ScatterChart**, che rappresentano rispettivamente un grafico a linea e un grafico a dispersione e in cui i punti vengono gestiti tramite QLineSeries e QScatterSeries. I grafici derivati da PointChart inoltre non mostrano le variabili sull'asse X ma solo gli indici del vettore delle variabili.

2.2 Contenitori

L'applicazione utilizza una classe contenitore **Dataset** che contiene un vettore con puntatori ai farmaci: la scelta di utilizzare un vettore è stata dettata dal fatto che questo tipo di container

consente l'accesso di un elemento in posizione arbitraria in tempo costante, caratteristica che viene spesso usata nell'applicazione per modificare ed eliminare un determinato farmaco senza dover scorrere l'intero contenitore ogni volta. Nonostante anche l'inserimento in coda venga utilizzato altrettando spesso all'interno dell'applicazione, il che viene effettuato più efficacemente da una lista, l'accesso di un elemento in posizione arbitraria è stato ritenuto più importante ai fini dell'applicazione. La classe Dataset implementa i metodi di distruzione profonda, inserimento ed eliminazione di un farmaco ed è stata dotata di iteratori costanti iniziale e finale.

2.3 Polimorfismo

Nell'applicazione vengono spesso usate chiamate polimorfe per diverse gerarchie grazie all'implementazione di metodi virtuali:

- il metodo di clonazione *clone()*
- il metodo *calculatePrice()*, usato per calcolare il prezzo finale di un farmaco
- i metodi *getType()* e *getChartType()*, usati al posto del dynamic cast
- l'overloading degli operatori di uguaglianza e disuguaglianza
- il metodo *getSize*
- i distruttori virtuali standard implementati in tutte le gerarchie
- il metodo *createChart()*, usato per creare un QChart

L'utilizzo del polimorfismo è stato ritenuto fondamentale per l'applicazione anche perchè rende più semplice la futura estensibilità delle classi utilizzate, soprattutto per le gerarchie dei farmaci e dei grafici.

2.4 Formati di I/O

Per l'applicazione si è scelto per l'archiviazione e il caricamento dei dati il formato di file XML; questo formato si presenta infatti particolarmente comprensibile e facilmente utilizzabile tramite le apposite classi e relativi metodi forniti proprio da Qt (**QXmlStreamReader** e **QXmlStreamWriter**). I metodi di lettura e scrittura di/su file XML sono stati implementati nella classe **XmlUtil** tramite stream reader e stream writer.
Un esempio di come viene strutturato un file XML usato nell'applicazione è il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Non modificare!-->
<Farmaci>
    <Compresse>
        <Nome>Tachipirina 500Mg</Nome>
        <Marca>Angelini</Marca>
        <Vendite>
            <Vendite2018>53400</Vendite2018>
            <Vendite2019>52950</Vendite2019>
            <Vendite2020>51900</Vendite2020>
            <Vendite2021>53990</Vendite2021>
        </Vendite>
        <Comp>30</Comp>
        <PrezzoComp>0.100000</PrezzoComp>
    </Compresse>
    <Flacone>
        <Nome>Vicks Sinex</Nome>
        <Marca>Vicks</Marca>
        <Vendite>
            <Vendite2018>43758</Vendite2018>
            <Vendite2019>44809</Vendite2019>
            <Vendite2020>42915</Vendite2020>
            <Vendite2021>43955</Vendite2021>
        </Vendite>
        <Ml>15.000000</Ml>
        <PrezzoMl>0.680000</PrezzoMl>
```

```
    </Flacone>
    ...
</Farmaci>
```

3 GUI

3.1 Model

La GUI dell'applicazione PharmaChart utilizza due componenti principali: una QTableView per presentare e gestire il dataset in forma tabulare e una QChartView per visualizzare i grafici.
Per gestire la TableView è stato usato un design patter **Model-View** (MV) per consentire maggior separazione possibile tra parte logica e interfaccia grafica. Sono state quindi create una classe **Model** con il compito di gestire tutte le funzionalità del dataset, una classe **TableModelAdapter** derivata da QAbstractTableModel con il compito di far comunicare il modello con la tabella e gestire inserimento, eliminazione e modifica dei dati di quest'ultima, e infine una classe **ProxyModelAdapter** derivata da QSortFilterProxyModel con il compito di collegare le azione effettuate dall'utente con il TableModelAdapter. Quest'ultima classe in particolare consente una maggiore estensibilità dell'applicazione, in quanto tramite Qt preve già dei metodi per un eventuale ordinamento dei dati della tabella.
Per quanto riguarda la gestione dei grafici invece, si è scelto di collegare direttamente la QChartView con la gerarchia di chart tramite il metodo *createChart()*, che ritorna direttamente un QChart, dato che quest'ultimo è già un oggetto grafico strettamente legato alla libreria Qt; la gestione dei dati usati nel chart resta invece gestita tramite il modello collegato con il dataset, il che consente di tenere il grafico aggiornato anche quanto viene eliminato o inserito un elemento nella tabella. L'unico caso in cui è necessario aggiornare manualmente il grafico, tramite un apposito bottone, rimane quando viene modificata una cella nella tabella.

3.2 View

Per quanto riguarda le view, si è scelto di dividere il tutto in molteplici classi ciascuno con un compito specifico per ottenere una maggiore modularità del codice:

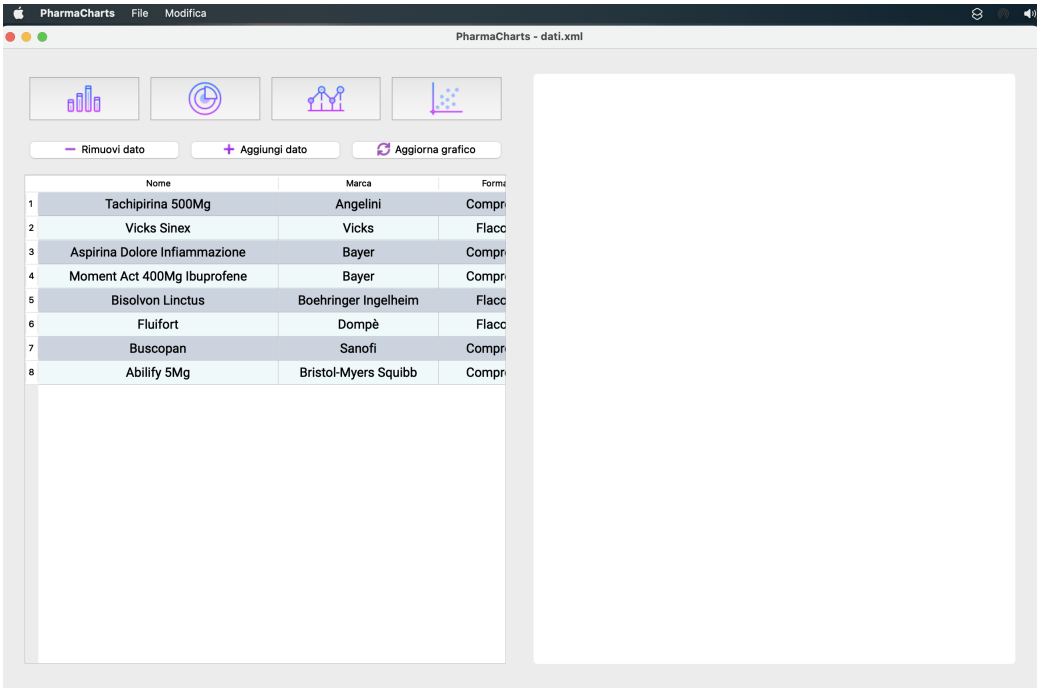
- la classe **FormatComboBox** si occupa di creare una QComboBox con il possibili tipi di farmaco da inserire
- la classe **InsertView** si occupa di inserire un nuovo farmaco nella tabella, abilitando o meno i diversi campi dati a seconda del tipo di farmaco selezionato
- la classe **Menu** si occupa di creare la barra dei menu, da dove è possibile effettuare le principali azioni (inserimento, caricamente e salvataggio su file e creazione dei vari grafici)
- la classe **MainContent** si occupa di visualizzare la tabella con il dataset e la serie di bottoni per effettuare le azione sulla tabella e sui grafici
- la classe **VariablesDialog** si occupa di eseguire il QDialog per la creazione di un grafico, in cui vengono chiesti il titolo e le variabili da considerare per la creazione del chart

Le azioni di *inserimento* e *rimozione* di un dato dalla tabella e di *creazione* di un grafico, così come le azioni di *salvataggio* e *caricamento* su/da file vengono gestite nella **MainWindow**, che si occupa quindi di raggruppare e gestire insieme tutte le varie componenti view dell'applicazione.

Nella InsertView inoltre sono stati posti dei valori massimi per gli ml totali (massimo 1000 ml), per le compresse totali (massimo 100), per il prezzo unitario (massimo 10 euro) e per le vendite (massimo 60000 per evitare di superare il massimo intero rappresentabile in c++ durante la normalizzazione dei dati).

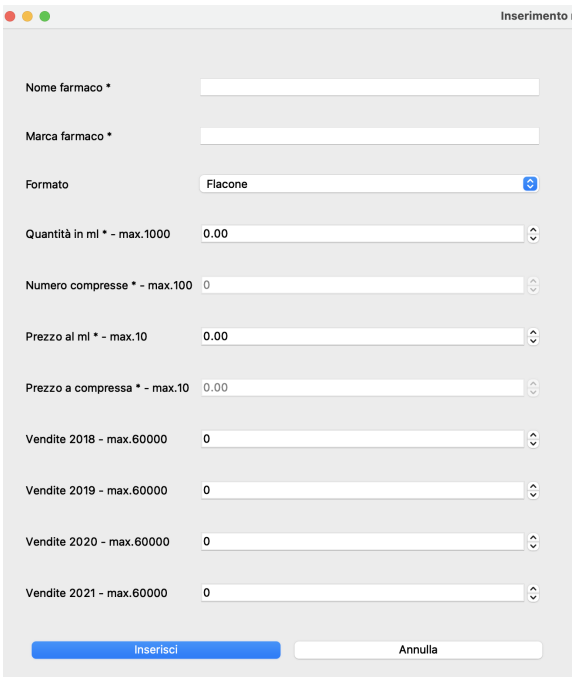
3.3 Esempio di utilizzo

All'apertura dell'applicazione viene richiesto di caricare il dataset da un file XML: se viene scelto un file valido si apre la schermata principale con la tabella popolata dal dataset e la chartview vuota, altrimenti viene creato un nuovo file vuoto e si aprirà la schermata principale con la tabella e la chartviee entrambe vuote.

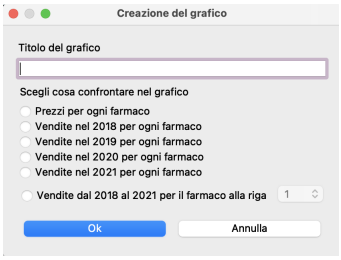


I bottoni di creazione dei vari tipi di grafico vengono tutti abilitati solo se il dataset ha al suo interno almeno due farmaci. Se il dataset risulta vuoto (come nel caso di creazione di un nuovo file) i bottoni sono tutti disabilitati, mentre se vi è solo un farmaco nel dataset vengono abilitati solo i bottoni per la creazione di BarChart e PieChart; infatti sia LineChart che ScatterChart, essendo costruiti a partire da coppie di punti, necessitano di almeno due coppie di punti distinti per venire generati. Lo stesso vale per le voci del menu corrispondenti.

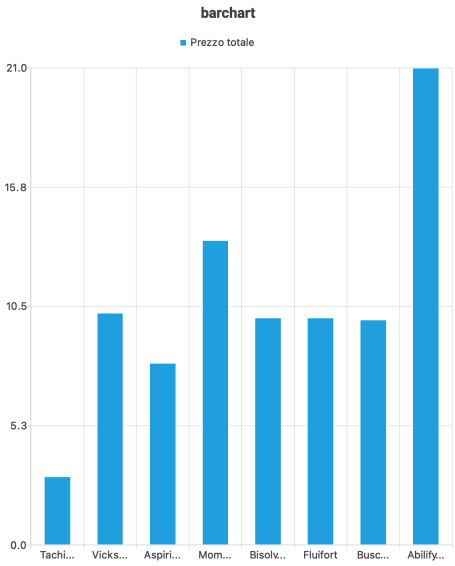
Dalla schermata principale, tramite il bottone "aggiungi dato", e dal menu è possibile inserire un nuovo farmaco nel dataset con la comparsa della view di inserimento. I campi dati obbligatori sono stati contrassegnati da un asterisco e se non compilati correttamente non rendono possibile l'inserimento del farmaco.
Inoltre il bottone "rimuovi dato" della schermata principale consente la rimozione della riga selezionata dalla tabella.



Tramite i quattro bottoni in alto, e dal menu, è possibile creare i quattro diversi tipi di grafico: BarChart, PieChart, LineChart e ScatterChart. Cliccando uno di questi bottoni viene caricato un Dialog che richiede di dare un titolo al grafico e di scegliere cosa confrontare nel grafico, scegliendo tra i prezzi di tutti i farmaci, le vendite in un determinato anno di tutti i farmaci o le vendite dal 2018 al 2021 del farmaco posto alla riga selezionata. Nel caso che tutte le vendite in un determinato anno di tutti i farmaci o le vendite



(a)



(b)

di un determinato farmaco siano tutte uguali a 0, appare un messaggio di errore e il grafico non viene creato per mancanza di dati.
Se i dati sono corretti invece nella schermata principale viene popolata la chartview con il grafico scelto.

Il grafico si aggiorna automaticamente quando viene inserito o eliminato un dato dalla tabella, mentre quando si effettua la modifica di una casella della tabella è necessario cliccare il bottone "aggiorna grafico".

Alla chiusura dell'applicazione viene sempre chiesto se salvare o meno il dataset.

4 Conteggio ore di lavoro

- **Analisi preliminare del problema:** 2 h
- **Progettazione modello:** 10 h
- **Progettazione GUI:** 3 h
- **Apprendimento libreria Qt:** 3h
- **Codifica modello:** 13 h
- **Codifica GUI:** 10 h
- **Debugging:** 2 h
- **Testing:** 5 h
- **Stesura relazione:** 2 h