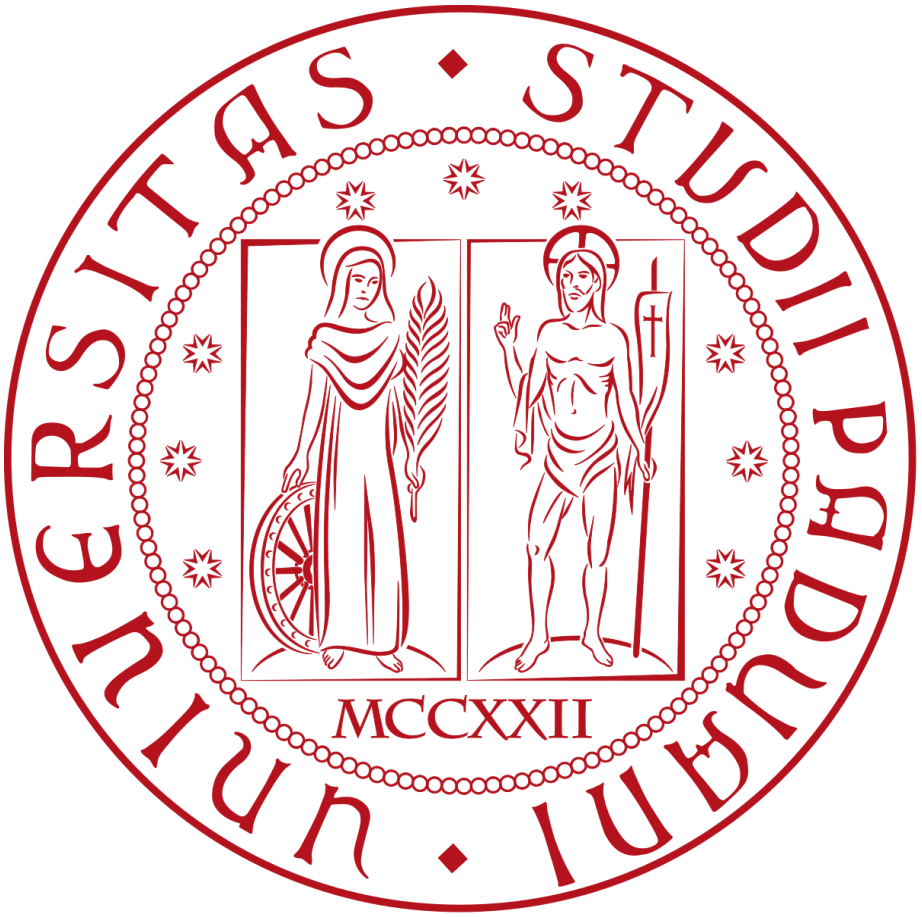
PROGETTO DI PROGRAMMAZIONE AD OGGETTI

A.A 2021/22

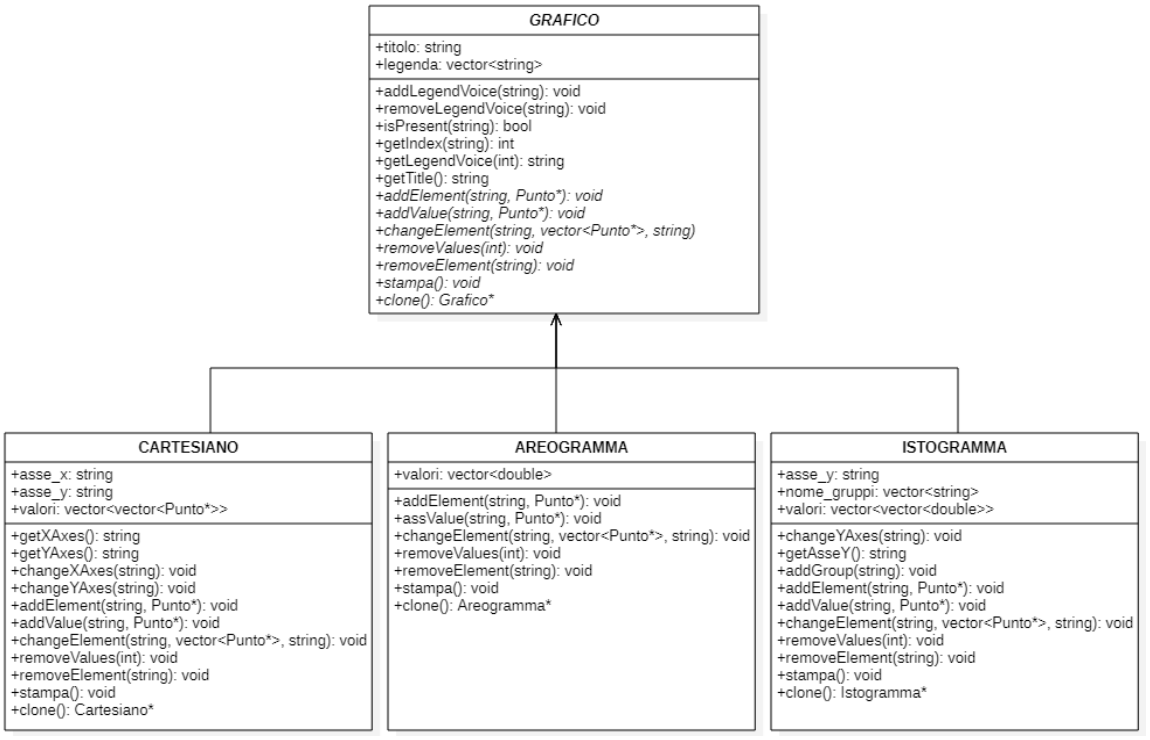


Fincato Alessandro 1201264

1. INTRODUZIONE

ChartU permette la creazione di grafici tramite la definizione di dati tramite una tabella. Il software rende disponibile la creazione di tre tipi di grafici: diagramma a Torta (o Areogramma), Istogramma oppure in grafico Cartesiano (lineare). Inoltre, oltre alla sola creazione, permette il salvataggio del dataset definito per creare il grafico su un file XML. Gli stessi file potranno quindi essere caricati sul software per ottenere subito il grafico che era stato definito.

2. LA GERARCHIA



La gerarchia alla base di ChartU si basa sulla classe base astratta **Grafico** dove vengono definiti i membri base presenti in un qualsiasi grafico, ovvero il *titolo* e la *legenda* contenente il nome degli elementi presenti nel grafico.
La classe base astratta viene poi estesa da 3 classi concrete che definiscono ognuna un tipo di grafico:

- **Cartesiano**: definisce i membri presenti in un qualsiasi grafico cartesiano, ovvero il nome dell' *asse x* e dell'*asse* y ed i punti di ogni elemento. Questi ultimi sono stati definiti in un vettore di vettore di punti, dove ognuno di questi ultimi definisce i punti appartenenti al rispettivo elemento definito nella legenda (l'indice dell'elemento nella legenda corrisponde all'indice del suo vettore di punti all'interno del vettore *valori*);
- **Areogramma**: definisce i membri presenti in un qualsiasi diagramma a torta, ovvero i soli *valori* degli elementi (visto l'assenza di assi del grafico). Questi sono definiti in un vettore di double con lo stesso concetto del vettore dei valori di *Cartesiano*, ovvero il valore all'indice *i* corrisponde all'elemento all'indice *i* presente in *legenda*.
- **Istogramma**: definisce i membri presenti in un qualsiasi istogramma, ovvero il nome dell'*asse y*, un vettore di string contenente i diversi *nomi dei gruppi* in cui si divideranno i valori di ciascun elemento e un vettore di vettori di double contenente appunto i *valori* di ciascun elemento appartenente al grafico. L'associatività tra vettore dei valori ed elementi è la stessa vista nei precedenti tipi, inoltre salviamo solamente double (ovvero il valore y del punto) perchè il valore x è dato da ciascun gruppo (il primo punto di un elemento appartiene al primo gruppo ed ha valore y).

2.1. CHIAMATE POLIMORFE

Nella classe base astratta grafico abbiamo definito dei metodi virtuali puri in modo da sfruttare il polimorfismo necessario per l'uso nel *Model* di solo un puntatore di tipo *Grafico*. Tali metodi virtuali puri sono:

- `void addElement(string nome, Punto* punto = nullptr)`
  permette l'aggiunta nell'oggetto di un nuovo elemento. Il *nome* del nuovo elemento sarà inserito all'interno della legenda, mentre verranno estratti i valori x e y dal punto -se definito- per poi essere usati a seconda del tipo (*Areogramma* e *Istogramma* useranno solamente il valore x, mentre *Cartesiano* userà l'intero punto). punto inoltre non deve essere per forza definito, perché ci importa solo della creazione dell'elemento, non anche di un suo eventuale "popolamento";

- `void addValue(string nome , Punto* punto)`
  permette l'aggiunta nell'oggetto di un valore all'elemento *nome*. Il concetto dell'utilizzo di *punto* è lo stesso del metodo precedente, con la differenza che in questo caso necessario passarlo come parametro per ovvie ragioni;

- `void changeElement(string Vnome,vector<Punto*> punti,string Nnome="")`
  permette il cambiamento per intero di un elemento appartenente all'oggetto. I valori dell'elemento *Vnome* verranno sostituiti con quelli definiti nel vettore *punti* ed il nome può essere modificato in *Nnome*;

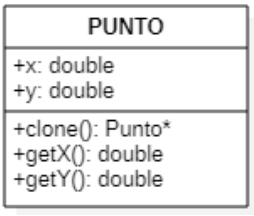
- `void removeValues(int i)`
  permette la rimozione del valore *i-esimo* di ogni elemento appartenente all'oggetto (sempre ammesso che ce l'abbia). Nel caso di *Areogramma*, pone tutti gli elemento a 0;

- `void removeElement(string)`
  permette la rimozione dell'elemento appartenente all'oggetto presente all'indice *i*. Come abbiamo fatto notare precedentemente, tale indice associa l'elemento in posizione *i* nella *legenda* con i rispettivi valori all'indice i del vettore dei *valori*;

- `void stampa()`
  metodo opzionale utile alla verifica che tutti i dati inseriti nella vista siano stati passati correttamente all'oggetto. La chiamata del metodo prova una stampa sul terminale;

- `T* clone()`
  classico metodo virtuale di clonazione visto anche a lezione;

2.2. CLASSE PUNTO



La classe *Punto* è stata definita con lo scopo di rappresentare le informazioni all'interno dell'oggetto in modo simile a quanto fanno i punti cartesiani nella vita reale. Dato l'utilizzo puramente "di lettura", ovvero senza la necessità di eseguire delle operazioni tra punti oppure confronti, sono stati definiti solamente dei metodi di clonazione e di lettura dei valori x e y (oltre che alla ridefinizione del costruttore, distruttore, costruttore di copia e operatore di assegnazione come quelli di default).

Nota importante: nel Model avviene uno scambio effettivo tra valore della x e della y utilizzati nei grafici e quelli utilizzati nel punto. Tale scambio avviene puramente per motivi pratici in quanto mi veniva più spontaneo, ad esempio in un Istogramma assegnare i valori x anziché quelli y. Spiego meglio, in un istogramma rappresentato dalla vista il primo elemento ha coordinate (gruppo di appartenenza, valore) che avrei quindi dovuto passare in modo equivalente agli oggetti. Quello che invece succede è la definizione di punti come (valore, gruppo di appartenenza), appunto per rendere la lettura più "pratica".

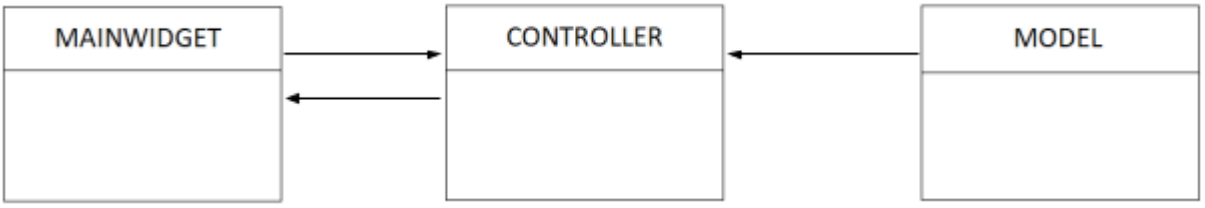
3. INPUT/OUTPUT

L'input/output avviene attraverso dei file di tipo .XML. Questo tipo di file permette la definizione di elementi che hanno degli attributi e a loro volta degli elementi "figli". Prendiamo ad esempio il seguente file generato:

```
<Grafico Titolo="CartChart" Asse_x="" Tipo="Cartesiano" Asse_y="asse_y">
 <l1>
  <punto_0 x="1" y="0.00"/>
  <punto_1 x="2" y="1.00"/>
 </l1>
 <l2>
  <punto_0 x="2" y="0.00"/>
  <punto_1 x="1" y="1.00"/>
 </l2>
</Grafico>
```

L'elemento principale è l'intero oggetto, a questo vengono assegnati degli attributi generali per tutto il grafico ovvero il *titolo*, il *tipo di grafico* ed il *nome degli assi*. Questo elemento ha dei figli, ovvero gli elementi che compongono il grafico definiti dal nome *-l1* e *l2-* e questi a loro volta sono composti da più elementi "figli" ovvero i punti di ogni elemento. A questi ultimi vengono assegnati gli attributi per definire il valore di *x* e *y* del *punto_i* (con $i = 0,1,...$).

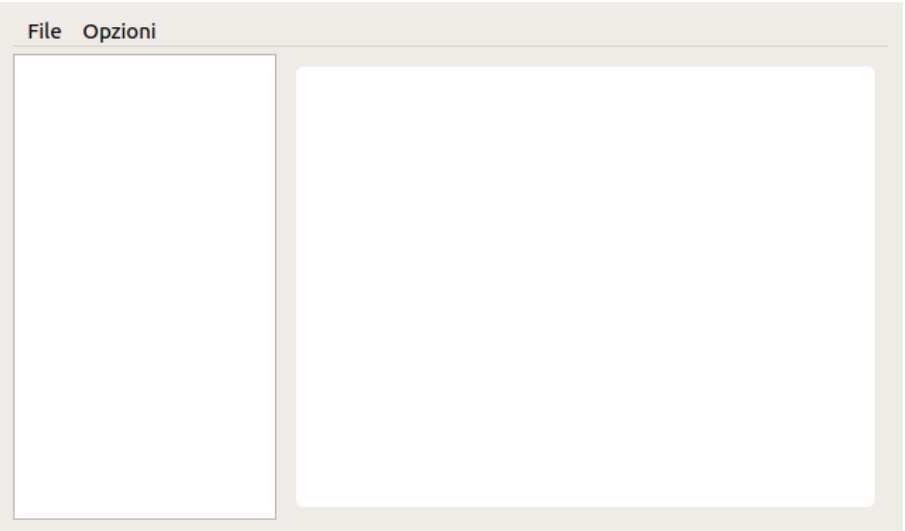
4. GUI



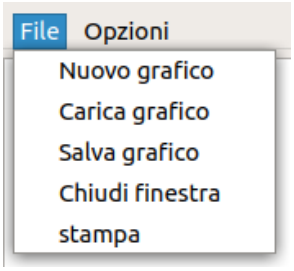
La GUI è stata definita aderendo al desing pattern **Model-View-Presenter**, una derivazione dell"MVC. Ho deciso di adottare questo pattern perchè lo trovo personalmente più comprensibile. Ovvero il Controller ha il ruolo di "attore principale" che riceve informazioni sia dal Model che dalla Vista e di conseguenza dice cosa fare. Mentre Model e View sono "limitati"a svolgere le operazioni su di loro a seconda di quanto gli è stato detto dal Controller oppure, nel caso della View, se è una operazione che non avrebbe nessun side effect su Controller o Model e quindi può essere gestita all'interno della sola View.
Il Model altro non è che un puntatore alla classe base astratta *Grafico* e nel quale definiamo tutte le operazioni da fare sull'oggetto che deve rappresentare il grafico. In *Controller* invece abbiamo un puntatore al modello e uno alla vista in modo tale da dare comandi ad entrambi sulle operazioni da fare. In *Mainwidget* invece abbiamo solo un puntatore al controller con il quale possiamo comunicare.

4.1. MANUALE UTENTE

Appena aperta la finestra appariranno due riquadri bianchi, uno per la tabella e uno per il grafico, ed un menu non le voci *File* e *Opzioni*, come mostrato in figura.

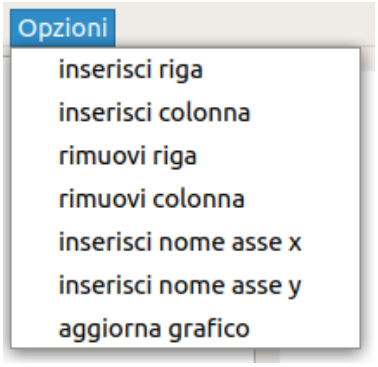


Premendo su *File* si aprirà il menu contenente le operazioni disponibili sul file che stiamo creando:



- *Nuovo grafico*: apre un dialog per la selezione del tipo di grafico e la definizione del titolo;
- *Carica grafico*: aprirà la selezione di un file presente nel dispositivo da aprire per caricare un grafico che era già stato creato;
- *Salva grafico*: permette il salvataggio su file del grafico sul quale stiamo lavorando;
- *Chiudi finestra*: chiude la finestra di lavoro (non salva ciò su cui stiamo lavorando);
- *stampa* serve invece per stampare su terminale l'oggetto creato in modo da vedere se le informazioni sono corrette;

Altrettanto succede premendo su *Opzioni*, solamente che se non abbiamo aperto nessun grafico nessuna sarà disponibile o farà effettivamente qualcosa. Infatti questo menù contiene tutte le operazioni eseguibili sul grafico e sulla tabella:



- *Inserisci riga*: apre un dialog che chiede il nome della nuova riga, ovvero il nome dell'elemento da inserire nel grafico;

- *Inserisci colonna*: apre un dialog che chiede il nome della nuova colonna. Questa è un'operazione non disponibile per gli Areogrammi. Inoltre per i Cartesiani chiede unicamente un double (visto che il valore della colonna corrisponde al valore y delle coordinate del punto), mentre per gli Istogrammi corrisponde al nome del nuovo gruppo da inserire;

- *Rimuovi riga*: apre un dialog che chiede l'indice della colonna da rimuovere. Questo rimuoverà ovviamente l'elemento associato a tale riga sul grafico;

- *Rimuovi colonna*: apre un dialog che chiede l'indice della colonna da rimuovere. Questo rimuoverà da tutti gli elementi il valore di quella colonna. Inoltre questa operazione non è disponibile per i grafici di tipo Areogramma;

- *Inserisci nome asse x*: apre dialog per inserire il nome da assegnare all'asse x. Questo è effettivamente disponibile solamente per i grafici Cartesiani;

- *Inserisci nome asse y*: apre dialog per inserire il nome da assegnare all'asse y. Questo è effettivamente disponibile solamente per i grafici Cartesiani e Istogrammi;

- *Aggiorna grafico*: aggiorna la visualizzazione del grafico applicando tutte le modifiche introdotte precedentemente. Solo le modifiche infatti vengono applicate al grafico, però la visualizzazione non viene aggiornata, per fare ciò bisogna usare questa operazione;

5. COMPILAZIONE

Nella cartella consegnata è stato anche inserito il file *ChartU.pro* perché necessario alla compilazione. Questa viene fatta seguendo i passi (dopo essersi messi nella cartella dei file):

1) `qmake ChartU.pro`
2) `qmake`
3) `make`

Dopodichè per far partire l'applicazione usare il comando: `./ChartU`

6. ORE DI LAVORO RICHIESTE

Analisi dei requisiti	1 ora
Progettazione modello e GUI	15 ore
Documentazione Qt	7 ore
Codifica modello e GUI	20 ore
Input/Outup da e su file	2 ore
Debugging e testing	15 ore
TOTALE	**60 ore**

Il monte ore di lavoro richiesto di 50 è quindi stato superato, ciò è avvenuto per via delle difficoltà iniziali di progettazione di una gerarchia e di un modello che si adattassero a ciò che avevo in mente di fare. Inoltre molto tempo è stato speso anche per il controllo e il debugging delle funzionalità della GUI in modo da mantenere una certa integrità del sistema, questo con i conseguenti test necessari ha portato via più tempo in assoluto.