Geometry Calc

Progetto Programmazione a oggetti

Bisacca Luca
Matricola : 1227589

1. Generalità

1.1 Introduzione

GeometryCalc è un'applicazione desktop che offre la possibilità di calcolare l'area ed il perimetro di figure geometriche piane a partire dai dati della figura in questione.
In particolare l'applicazione è dotata di una cronologia capace di mostrare, oltre ai risultati, anche il tipo di figura ed i dati relativi a tutti i calcoli svolti per ciascuna figura.

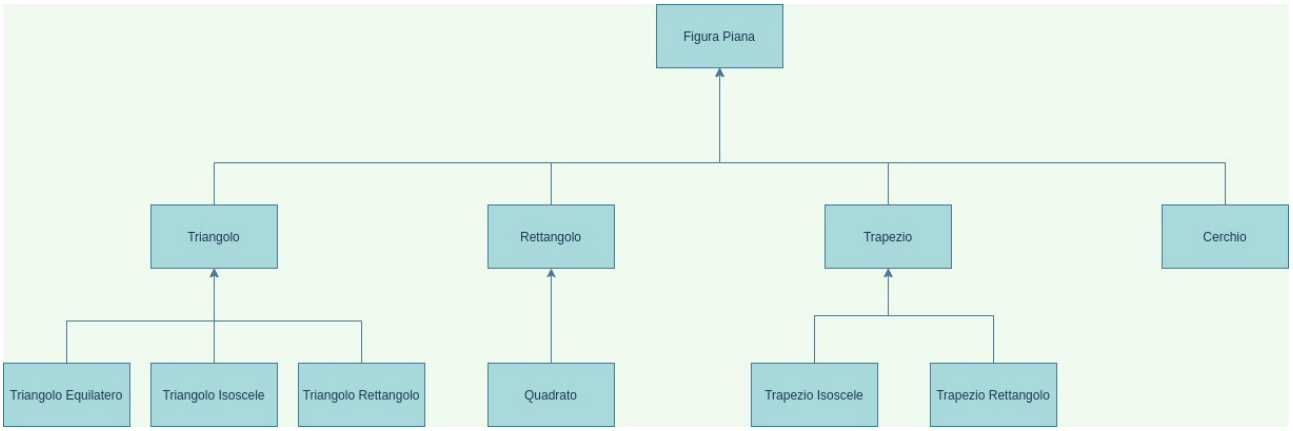
1.2 Ambiente di sviluppo

Sistema operativo : Ubuntu 20.04
IDE : Qt creator 4.14.0
Versione C++ : C++ 11
Versione Qt: 5.12.2
Compilatore : gcc version 9.3.0

1.3 Compilazione ed esecuzione

Per la compilazione del progetto viene fornito, insieme agli headers e sources files, un file 'GeometryCalc.pro'.
Recandosi da terminale all'interno della cartella in cui è presente il file sopracitato ed seguendo il comando ' **qmake** ' verrà generato il makefile per la compilazione dell'applicazione.
A questo punto basterà avviare il comando ' **make** ' così da ottenere l'eseguibile dell'applicazione.
Per far partire l'applicazione desktop è sufficiente eseguire il comando '**./GeometryCalc** '.

2. DESCRIZIONE DELLA GERARCHIA



La figura sopra riportata mostra la gerarchia che è stata presa in considerazione per lo sviluppo dell'applicazione.
In particolare:

La classe base **FiguraPiana** è una classe astratta in quanto non è possibile stanziare oggetti di tipo FiguraPiana poiché, in tale applicativo, privi di senso logico.

Campi dati FiguraPiana:

doble area è un campo dati marcato protetto in quanto deve essere accessibile alle classi derivate da quest'ultima. Esso rappresenta l'area occupata da una figura.

doble perimetro è un campo dati marcato protetto in quanto deve essere accessibile alle classi derivate da quest'ultima. Esso rappresenta il perimetro occupato da una figura.

Metodi FiguraPiana :

doble getArea() const : getter per l'area della figura. Restituisce un doble con la dimensione occupata dall'area della figura.

doble getPerimetro() const : getter per il perimetro della figura. Restituisce un doble con la dimensione occupata dal perimetro della figura.

voi setArea() : metodo virtuale puro, setter per l'area della figura. Sarà implementato un override di questo metodo nelle classi figlie. Lo scopo di tale metodo è quello di settare l'area della specifica figura utilizzando le formule relative alla figura in questione.

voi setPerimetro() : metodo virtuale puro, setter per il perimetro della figura. Sarà implementato un override di questo metodo nelle classi figlie. Lo scopo di tale metodo è quello di settare il perimetro della specifica figura utilizzando le formule relative alla figura in questione.

FiguraPiana* clone() const : metodo virtuale puro, sarà implementato un override che permette alle classi figlie di essere copiate in modalità profonda. Ciascuno dei suoi override si servirà della covarianza restituendo quindi un puntatore ad un oggetti allocato sullo heap. Il puntatore sarà un puntatore ad un oggetto di tipo T con T tipo della figura allocata sullo heap.

La classe **Triangolo** è una classe concreta derivata pubblicamente da FiguraPiana. Rappresenta un triangolo qualsiasi.

Campi dati Triangolo:

double lato1 è un campo dati marcato privato. Indica la misura di uno dei lati del Triangolo.

double lato2 è un campo dati marcato privato. Indica la misura di uno dei lati del Triangolo.

double base è un campo dati marcato privato. Indica la misura della base del Triangolo.

Metodi Triangolo:

bool isTriangle() const : metodo per controllare che le misure dei lati siano regolari.

double getLato1() const : getter per il campo dati lato1 dell'oggetto di tipo Triangolo. Restituisce un double con la dimensione del lato1 dell'oggetto di tipo Triangolo.

double getLato2() const : getter per il campo dati lato2 dell'oggetto di tipo Triangolo. Restituisce un double con la dimensione del lato2 dell'oggetto di tipo Triangolo.

doble getBase() const : getter per il campo dati base dell'oggetto di tipo Triangolo. Restituisce un double con la dimensione della base dell'oggetto di tipo Triangolo.

doble altezza() const : metodo virtuale, Restituisce un double con il valore dell'altezza dell'oggetto di tipo Triangolo.

Altri metodi sono ereditati dalla classe base FiguraPiana. Per alcuni di questi metodi è fornito anche un override.

La classe **TriangoloEquilatero** è una classe concreta derivata pubblicamente da Triangolo. Rappresenta un triangolo i cui lati sono tutti uguali. Implementa override di alcuni metodi di Triangolo.

La classe **TriangoloIsoscele** è una classe concreta derivata pubblicamente da Triangolo. Rappresenta un triangolo avente due lati uguali tra loro. Implementa override di alcuni metodi di Triangolo.

La classe **TriangoloRettangolo** è una classe concreta derivata pubblicamente da Triangolo. Rappresenta un triangolo avente un angolo retto. Implementa override di alcuni metodi di Triangolo.

La classe **Rettangolo** è una classe concreta derivata pubblicamente da FiguraPiana. Rappresenta appunto un rettangolo con lati uguali a due a due.

Campi dati Rettangolo:

double base : campo dati marcato privato. Indica la misura della base del Rettangolo.
double altezza : campo dati marcato privato. Indica la misura dell'altezza del rettangolo.

Metodi Rettangolo:

double getBase() const : getter per il campo dati base dell'oggetto di tipo Rettangolo. Restituisce un double con la dimensione della base dell'oggetto di tipo Rettangolo.

double getAltezza() const : getter per il campo dati altezza dell'oggetto di tipo Rettangolo. Restituisce un double con la dimensione dell'altezza dell'oggetto di tipo Rettangolo.

Altri metodi sono ereditati dalla classe base Figura Piana. Per alcuni di questi metodi è fornito anche un override.

La classe **Quadrato** è una classe concreta derivata pubblicamente da Rettangolo. Rappresenta appunto un Quadrato (nonché un rettangolo con tutti e quattro i lati uguali tra loro). Implementa override di alcuni metodi di Rettangolo.

La classe **Trapezio** è una classe concreta derivata pubblicamente da FiguraPiana. Rappresenta un trapezio qualsiasi.

Campi dati Trapezio:
double baseMaggiore : campo dati marcato privato. Indica la misura della base maggiore del Trapezio.

double baseMinore : campo dati marcato privato. Indica la misura della base minore del Trapezio.

double lato1 : campo dati marcato privato. Indica la misura di uno dei due lati del Trapezio.

double lato2 : campo dati marcato privato. Indica la misura di uno dei due lati del Trapezio

Metodi Trapezio:

void swapBase() : metodo implementato dal costruttore al fine di controllare che la base maggiore sia effettivamente maggiore rispetto alla base minore. Qualora non lo fosse scambia i valori dei due dati facendo in modo da mantenere la correttezza delle formule.

bool isTrapezio() const : metodo per controllare che le misure dei lati siano regolari.

double getLato1() const : getter per il campo dati lato1 dell'oggetto di tipo Trapezio. Restituisce un double con la dimensione del lato1 dell'oggetto di tipo Trapezio.

double getLato2() const : getter per il campo dati lato2 dell'oggetto di tipo Trapezio. Restituisce un double con la dimensione del lato2 dell'oggetto di tipo Trapezio.

double getBaseMaggiore() const : getter per il campo dati baseMaggiore dell'oggetto di tipo Trapezio. Restituisce un double con la dimensione della base maggiore dell'oggetto di tipo Trapezio.

double getBaseMinore() const : getter per il campo dati baseMinore dell'oggetto di tipo Trapezio. Restituisce un double con la dimensione della base minore dell'oggetto di tipo Trapezio.

double altezza() const : metodo virtuale, restituisce un doble con la dimensione dell'altezza dell'oggetto di tipo Trapezio.

void setBaseMaggiore() : setter per la dimendione della base maggiore.

void setBaseMinore() : setter per la dimendione della base minore.

void setLato1() : setter per la dimensione del lato1

void setLato2() : setter per la dimensione del lato2

Altri metodi sono ereditati dalla classe base FiguraPiana. Per alcuni di questi metodi è fornito anche un override.

La classe **TrapezioIsoscele** è una classe concreta derivata pubblicamente da Trapezio. Rappresenta un trapezio avente i due lati obliqui uguali tra loro. Implementa override di alcuni metodi di Trapezio.

La classe **TrapezioRettangolo** è una classe concreta derivata pubblicamente da Trapezio. Rappresenta un trapezio avente due angoli retti. Implementa override di alcuni metodi di Trapezio.

La classe **Cerchio** è una classe concreta derivata pubblicamente da FiguraPiana. Rappresenta un cerchio qualsiasi.

Campi dati Cerchio:

double raggio : campo dati marcato privato. Indica la misura del raggio del Cerchio.

Metodi Cerchio:

double getRaggio() const : getter per il campo dati raggio dell'oggetto di tipo Cerchio. Restituisce un doble con la dimensione del raggio dell'oggetto di tipo Cerchio.

3. SCELTE PROGETTUALI

3.1 Pattern Architetturale

È stato scelto il pattern model-view-controller così da ottenere una migliore separazione e gestione di model e view.
Il pattern ha permesso di creare un model indipendente dalla view.
In particolare il model gestisce le classi che compongono la gerarchia e fornisce tutti gli strumenti necessari all'applicazione.
La View, in questo caso, implementa solo la GUI e le interazioni con l'utente, senza mai trovarsi a collaborare in nessun modo con il model.
Il collegamento tra le due realtà appena descritte viene fornito dal controller, il quale si occupa di fornire alla view le informazioni necessarie da visualizzare utilizzando gli strumenti messi a disposizione dal model.

3.2 Contenitore e puntatori smart
Il model implementa un contenitore FiguraPianaContainer di puntatori smart ad oggetti di tipo FiguraPiana

3.2.1 Contenitore
Si è scelto di implementare un contenitore traendo ispirazione dal vector della STL di C++.

È stato scelto di implementare questo tipo di contenitore poiché il vector è un contenitore che supporta l'accesso casuale agli elementi in tempo costante. Questo favorisce maggiore efficienza nelle funzionalità di accesso alla cronologia.

Il vector inoltre permette inserimento in coda in tempo ammortizzato costante. Questo fattore è importante poiché ogni calcolo eseguito dall'utente viene immediatamente inserito come ultimo elemento nel contenitore senza preoccuparsi di dover scorrere prima tutti gli elementi precedentemente inseriti.

Per quanto riguarda la rimozione di elementi in posizione arbitraria sarebbe stato sicuramente più efficiente utilizzare un contenitore come list che però non supporta l'accesso casuale agli elementi perdendo dunque l'efficienza precedentemente descritta.

Ritenendo dunque l'accesso casuale un fattore di maggior peso nella scelta, alla fine, la scelta è ricaduta su un contenitore simile al vector rinunciando dunque all'ottimizzazione di efficienza in rimozione ma tenendo contro degli altri benefici sopracitati.

Il contenitore è stato implementato con la classe templatizzata FiguraPianaContainer ed è stato dotato di metodi, iteratori e const_iteratori con funzionalità simili a quelle del vector.

3.2.2 Puntatori smart
Per quanto riguarda i puntatori smart si è scelto di implementare una classe templatizzata DeepPtr che supporta le modalità di copia, assegnazione e distruzione in modalità profonda.
Altre funzionalità sono quelle standard del puntatore quali operatori * e → e la possibilità di convertire in un valore di tipo bool il puntatore che qualora dovesse essere inizializzato a nullptr restituisce valore false, true altrimenti.

4. USO DEL POLIMORFISMO

Nella classe model sono presenti alcune funzioni che fanno uso di chiamate polimorfe.

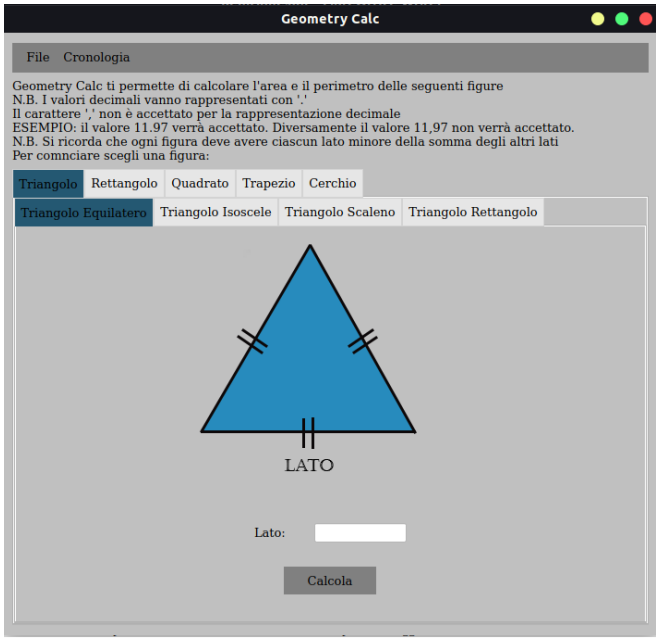
La funzione **getPerimetro()** chiama a sua volta la funzione getShape() che restituisce un DeepPtr di tipo FiguraPiana con dinamicamente allocato un oggetto di tipo G dove G è un tipo istanziabile della gerarchia sopracitata. Questo puntatore a sua volta chiama la funzione setArea() che essendo virtuale chiamerà la funzione setArea() guardando al tipo dinamico del DeepPtr.
Successivamente la funzione ritorna un double che sarà ottenuto chiamando ulteriormente la funzione getShape() che restituirà un puntatore con il quale sarà chiamata la funzione getPerimetro() che essendo virtuale chiamerà getPerimetro() guardando al tipo dinamico del DeepPtr.

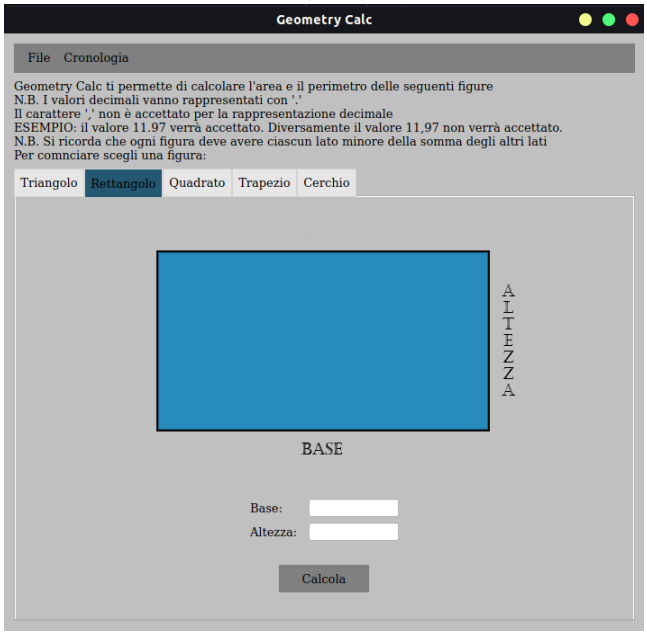
La funzione **getArea()** funziona in modo del tutto analogo a quello descritto per la funzione getPerimetro() con chiamate alle funzioni virtuali setArea() e getArea() che andranno a guardare al tipo dinamico del puntatore con cui saranno chiamate per scegliere la giusta funzione da chiamare.

Un altra funzione in cui è possibile notare l'uso del polimorfismo è la funzione getInfo() : questa funzione viene usata per scrivere i dati della figura all'interno della cronologia.
La funzione viene chiamata dall'omonima funzione getInfo() della classe Controller per mezzo di un puntatore.
Quando chiamata, la funzione, fa uso di RTTI per capire da quale tipo è stata chiamata e quindi a sua volta quali funzioni chiamare usando ancora dei metodi virtuali.

5. MANUALE UTENTE

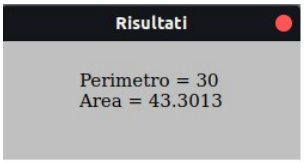
L'applicazione all'avvio mostrerà la sua finestra di interazione principale con l'utente (figura sulla sinistra).

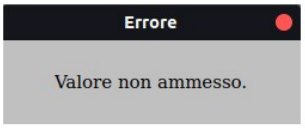




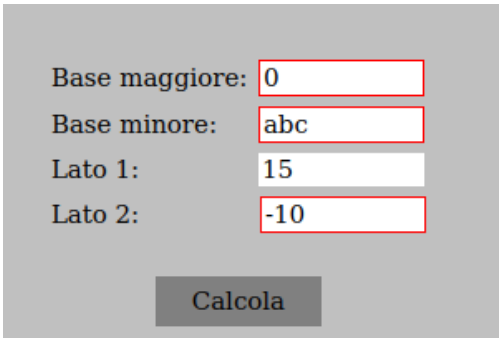
È possibile muoversi tra i possibili tipi di figura cliccando sui tab interessati. Il tab selezionato risulterà evidenziato in blu (ad esempio nella figura sulla destra si è scelto di muoversi nel tab dedicato al rettangolo).
Una volta selezionato il tab di interesse sarà mostrata la pagina relativa alla figura scelta.
Come visibile dalle immagini ciascun tab è composto da una finestra con una figura e dei box di inserimento dati.
All'interno dei box è possibile inserire soltanto valori numerici positivi maggiori di 0. I valori numerici decimali vanno indicati con il carattere '. e non con il carattere ',' come spiegato sulle istruzioni descrittive riportate sopra i tab.

Immediatamente sotto i box di inserimento vi è un tasto che ogni qual volta viene cliccato estrae i dati effettuando i calcoli necessari. Il click di questo tasto mostrerà una finestra di dialogo.
Nel caso in cui i valori inseriti sono valori ammissibili appariranno i risultati desiderati all'interno di una finestra di dialogo descrittiva.
Qualora i dati non dovessero essere ammissibili la finestra di dialogo mostrerà un messaggio di errore.



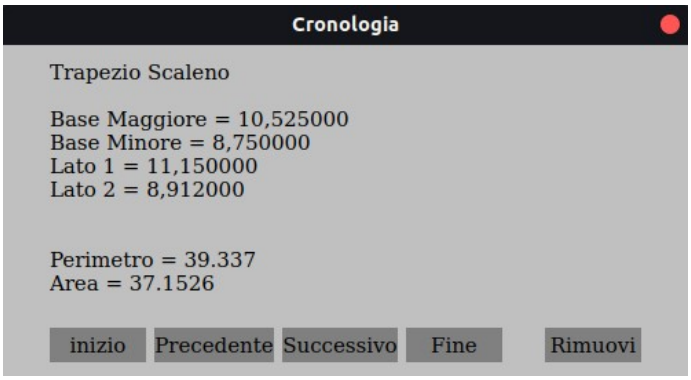


Quando l'utente inserisce dati non ammissibili, oltre al mostrare il dialogo di errore, l'applicazione si cura di evidenziare in rosso il box (o i box) che contengono il valore sbagliato.



In alto è presente un menu. Da questo è possibile accedere alla cronologia.
Cliccando su 'Cronologia' verrà mostrato un menù con le impostazioni 'Mostra cronologia' e 'Cancella cronologia'.

La cronologia mostra una finestra di dialogo con le informazioni descrittive delle figure che sono state calcolate e dei tasti che permettono lo scorrimento della cronologia o la rimozione. All'apertura la cronologia mostra sempre l'ultima figura di cui è stato effettuato il calcolo e la cronologia è ordinata con gli inserimenti dei dati in ordine cronologico di calcolo.



Tasti:
Inizio : mostra il più vecchio degli elementi presenti in cronologia se l'elemento corrente è diverso dal primo, altrimenti il tasto non comporta cambiamenti significativi.

Precedente : scorre la cronologia indietro di un elemento se l'elemento corrente è diverso dal primo, altrimenti il tasto non comporta cambiamenti significativi.

Successivo : scorre la cronologia in avanti di un elemento se l'elemento corrente è diverso dall'ultimo, altrimenti il tasto non comporta cambiamenti significativi.

Fine : mostra l'ultimo degli elementi presenti in cronologia se l'elemento corrente è diverso dall'ultimo, altrimenti il tasto non comporta cambiamenti significativi.

Rimuovi : rimuove permanentemente dalla cronologia l'elemento corrente e mostra il successivo se è presente un elemento successivo, il precedente altrimenti. Qualora l'elemento rimosso dovesse essere l'unico presente in cronologia, dopo aver rimosso l'elemento apparirà una finestra di dialogo con un avviso di cronologia vuota.

Nel caso in cui la cronologia dovesse essere vuota al click dell'azione mostra cronologia apparirà un dialogo nel quale appare scritto che nessun elemento è presente nella cronologia.

Nel caso invece si scegliesse di cancellare l'intera cronologia apparirà una finestra di conferma di cancellazione in modo da evitare eventuali cancellazioni indesiderate dall'utente.

6. TEMPO IMPIEGATO

ANALISI PRELIMINARE : ≃ 4 ore.
PROGETTAZIONE MODELLO E GUI : ≃ 8 ore
APPRENDIMENTO LIBRERIA Qt : ≃ 16 ore
CODIFICA MODELLO : ≃ 14 ore
CODIFICA GUI : ≃ 13 ore
TESTING + DEBUGGING : ≃ 7 ore

Il totale di ore ammonta quindi a **≃ 62 ore**.

Il superamento delle circa 50 ore massime è sicuramente dovuto al lungo periodo di apprendimento delle librerie di Qt e a qualche intoppo in fase di codifica di alcune funzionalità.

Aggiornamento nuova consegna:
Aggiunta circa un'ora di lavoro in seguito a brevi migliorie nel codice quali controlli riguardanti la correttezza dell'inserimento dei dati in modo tale che questi siano ammissibili, arrivando così ad un totale di **≃ 63 ore**.