Kalk

LEGROTTAGLIE GIONATA 1102654

Progetto del corso di Programmazione ad Oggetti a.a 2017/2018

Sommario

«Per attirare l'attenzione dei lettori su un elemento importante, può essere utile metterlo in evidenza con una barra laterale.»

Prefazione

Scopo del progetto

Lo scopo di questo progetto è di realizzare una calcolatrice utilizzando il linguaggio c++ e java che sia in grado di effettuare delle operazioni su tante figure geometriche diverse.

Strumenti utilizzati

La versione in java è utilizzabile solo da linea di comando.

La versione in c++ dispone anche di un'interfaccia grafica creata utilizzando la libreria Qt.

L'intero progetto è stato realizzato utilizzando Qt Creator 4.5 per Windows 10 ed in fine testato sulla macchina virtuale con Ubuntu messa a disposizione dal professore.

Sulla macchina virtuale è installata la versione 5.4.0 di Gcc.

Numero di ore richieste perla realizzazione del progetto

- 5 Ore per l'analisi totale del problema
- 5 ore per la progettazione del modello/GUI
- 2 ore per la progettazione dell'algoritmo "GramScan" e tutte le sue funzioni interne
- 10 ore per l'apprendimento della libreria Qt
- 20 ore per la codifica del modello
- 25 ore per la codifica della GUI
- 5 ore per il debugging e testing

Kalk

Funzionalità principali

Kalk è una calcolatrice che permette di fare alcune operazioni su figure geometriche.

Le figure geometriche implementate del modello sono:

1. Poligoni Convessi
    a. Triangolo
    b. Quadrilatero
    c. Pentagono
    d. Esagono
    e. ……..
    f. Icosagono (20 vertici)
2. Cerchio

Le operazioni sono effettuabili tra due Poligoni convessi o tra due cerchi.

Tra due poligoni convessi è possibile effettuare la somma, che ritorna un poligono convesso creato utilizzando l'unione dell'insieme dei vertici del primo poligono e del secondo(se l'insieme dei vertici non forma un poligono convesso, l'algoritmo GramScan rimuoverà alcuni vertici in modo da ottenere un insieme corretto), o la sottrazione che ritorna un poligono convesso creato utilizzando l'insieme di vertici del primo poligono convesso escludendo i vertici in comune con il secondo poligono convesso.

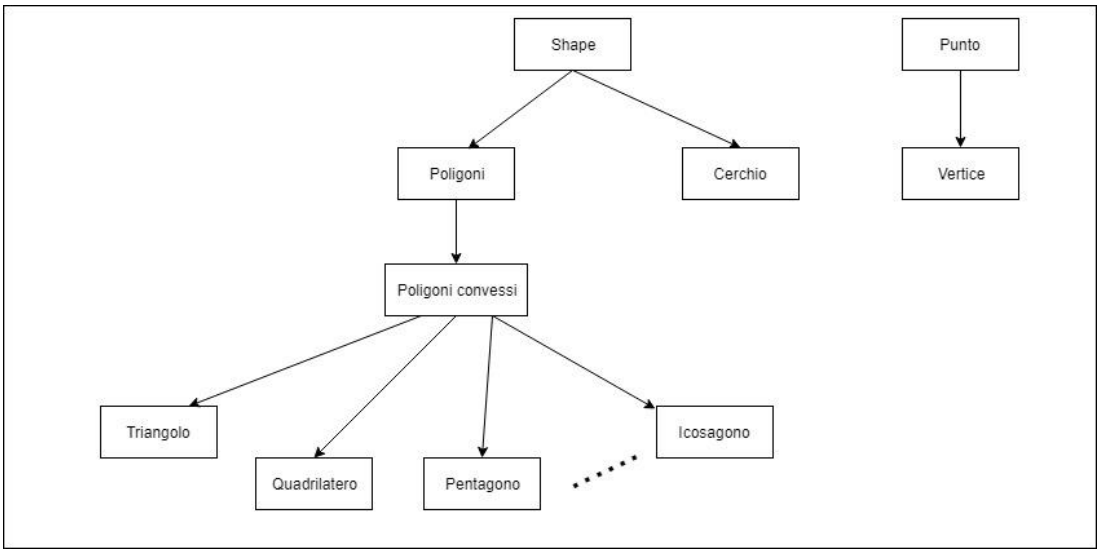
Su un **singolo poligono** convesso è possibile effettuare due operazioni, l'aggiunta o la rimozione di vertici.

Tra due cerchi è possibile effettuare la somma, che ritorna un cerchio che ha lo stesso origine del primo e il raggio uguale alla somma del raggio dei due cerchi, la sottrazione che ritorna un cerchio che ha lo stesso origine del primo ed il raggio che è la differenza tra il raggio del primo cerchio ed il raggio del secondo cerchio.

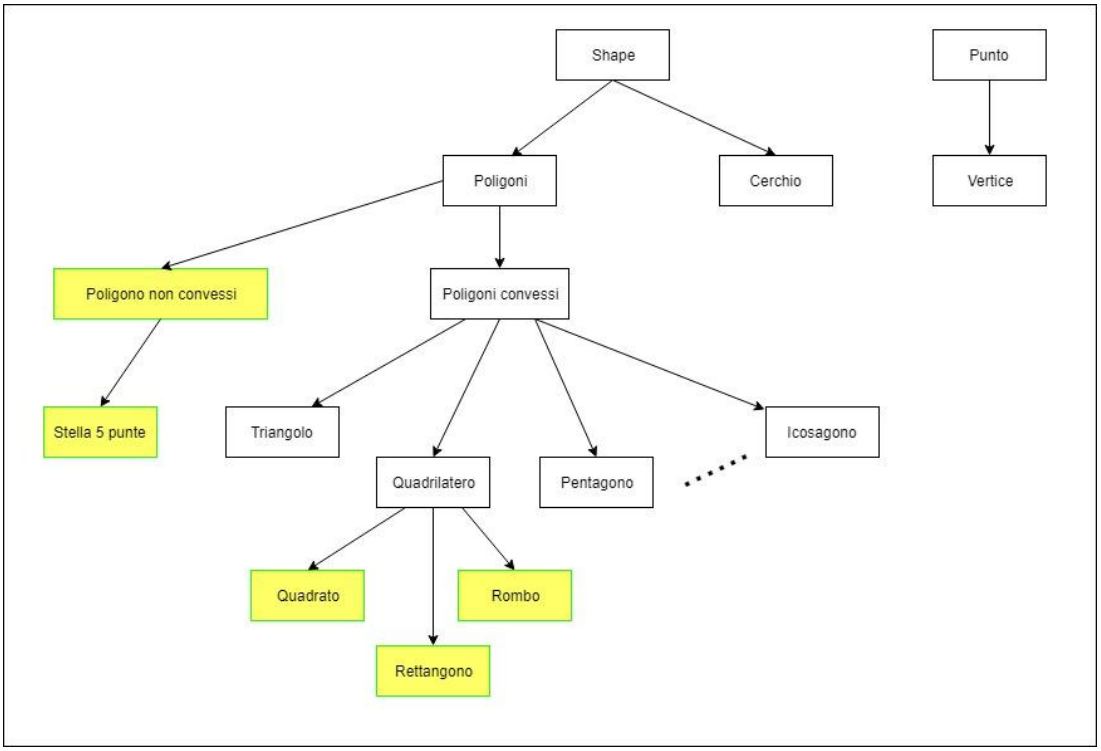
Gerarchia

La gerarchia del modello

Gerarchia realizzata.



Possibile estensione della gerarchia



Punto e Vertice

Queste sono due classi concrete.

La classe **Punto** ha due campi double protected che indicano le coordinate sul piano cartesiano ed una serie di metodi pubblici che aggiungono funzionalità alla classe:

- costruttore a zero parametri che crea un punto nell'origine: (0,0)
- costruttore a due parametri che crea un punto nelle coordinate x e y: (x,y)
- costruttore di copia
- distruttore virtuale
- il metodo toString() che restituisce un oggetto QString contenente i dati di quel punto
- gli overloading degli operatori: ==, != <;

Al di fuori della classe c'è l'overloading dell'operatore di output, il metodo to_string_nozero ed il metodo distanza che ritorna la distanza tra due punti.

La classe **Vertice** è un'estensione della classe Punto. L'unica cosa che le differenzia è la possibilità di calcolare un angolo tra 3 vertici con il metodo get_Angolo(a,b,c) ed il metodo toString che nella descrizione del Punto aggiunge la lettera "V" per indicare che si tratta di un vertice.

Shape e derivati

La classe astratta **Shape** ha un campo protected con indica un indice univoco per per ogni shape costruita e non ancora distrutta. Il conteggio di tutte le shape costruite avviene tramite un campo statico privato che viene incrementato direttamente tramite il costruttore.

La classe shape è possiede i vari costruttori, quattro metodi virtuali puri ed un metodo statico protetto che viene utilizzato nei vari distruttori per diminuire il numero di shape presenti.

La classe astratta **Poligono** è una classe creata per gestire meglio le future estensioni della gerarchia. Qui non vengono aggiunti campi dati, ma solo il metodo virtuale puro size() che ritorna il numero di vertici del poligono.

La classe astratta **PoligonoConvesso** è il vero nucleo della gerarchia.

Il campo dati protected Lista è un QVector<Vertici> che contiene tutti i vertici del poligono.

La classe è corredata di costruttore di copia pubblico, ed un costruttore a 3 parametri e 4 parametri protected. Questi ultimi sono stati marcati protetti per vietare la costruzione tamite

la chiamata ai costruttori e permetterla solo tramite la chiamata al metodo di classe crea_poligono.

Quest'ultimo ritorna un poligono convesso creato da una lista di vertici che prima di essere passata al costruttore a 3 parametri viene controllata e filtrata da una serie di metodi di classe privati che ordinano in ordine polare la lista di vertici ed eliminano eventuali doppioni o vertici che non permettono la creazione di un poligono convesso. Quindi può capitare di chiamare il metodo crea_poligono, passandogli una lista di 10 vertici e ricevere come risultato un puntatore polimorfo ad un triangolo (in questo caso i 7 vertici che sono stati eliminati erano doppioni, punti interni alla figura o vertici sul perimetro con angolo di 180 gradi).

Il metodo size ritorna la dimensione del vector Lista ed il metodo getPunt_vertici ritorna un riferimento costante alla lista dei vertici.

La classe PoligonoConvesso oltre ad ereditare i metodi virtuali puri di Shape e Poligono ne aggiunge altri due: get_lati e set_lati;

Le 17 classi che estendono PoligonoConvesso sono quasi identiche, tranne per qualche funzione interna.

Ad esempio, il triangolo ed il quadrilatero utilizzano un algoritmo differente per il calcolo dell'area. Tutte le restanti 15 classi per calcolare l'area dividono la superficie in tanti piccoli triangoli, calcolano l'area di ogni triangolino ed in fine restituiscono la somma di quest'ultimi.

Il metodo get_lati(unsigned i) ritorna la lunghezza del lato "i" ed il metodo protetto set_lati viene utilizzato per calcolare tutte le lughezze dei lati ed inserirle in un array di double di nome Lati[] che è campo dati di tutte le classi sotto PoligonoConvesso. Oltre a questo campo dati hanno anche il campo dati nLati che indica il numero di lati di quella figura.

Il metodo get_nomeClasse() ritorna il nome della classe, utile per identificare il tipo di figura ritornato dal metodo crea_poligono.

Cerchio

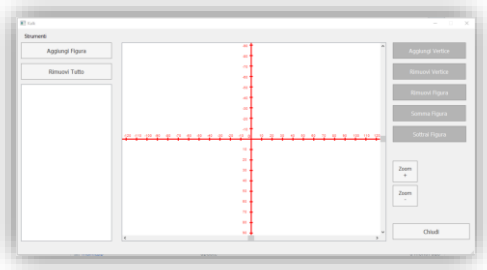
La classe **Cerchio** ha due campi privati: Vertice di origine e raggio (sempre positivo)

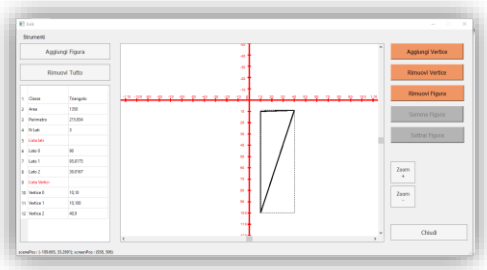
La classe viene corredata di costruttore di copia e costruttore a due parametri, ridefinisce tutti i metodi ereditati da shape ed aggiunge il metodo get_raggio, get_origine e set_origine che permette di spostare l'origine del cerchio.

Sono stati ridefiniti gli operatori +=, =, +, - e gli operatori booleani ==, !=, <, >, <=, >=;

La GUI

MainWindow

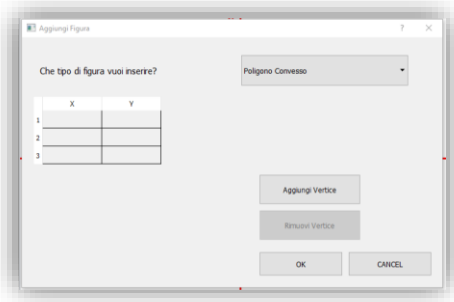


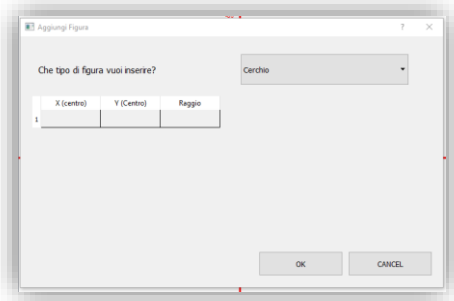


La mainwindow principale è divisa in tre zone.

1. La parte sinistra con i pulsanti sempre attivi che servono per eliminare tutte le figure presenti nel piano cartesiano e per inserirne di nuove. Sotto di c'è uno spaziodove verrà inserita a run-time una tabella contentente una serie di dati della figura selezionata.
2. La parte centrale con il piano cartesiano.
3. La parte di destra con tutti i pulsanti utili per effettuare tutte le operazioni sulla/e figura/e selezionata/e, lo zoom sul piano cartesiano ed il pulsante per chiudere l'applicazione

Form per l'inserimento di una nuova figura





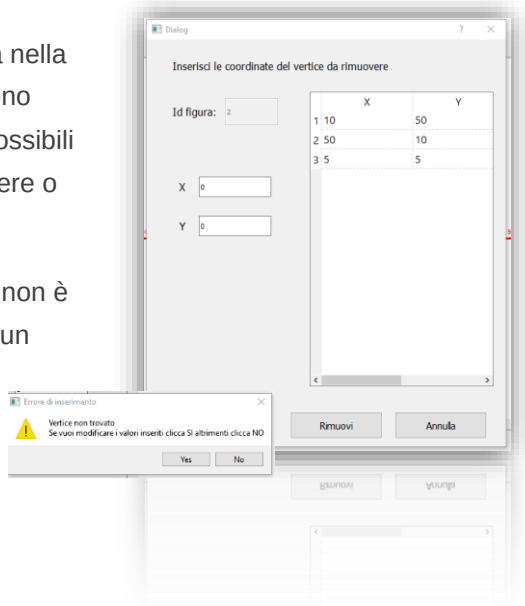
Tramite questo form è possibile inserire nuovi poligoni o cerchi. È possibile scegliere tra poligono o cerchio tramite il menù a tendina.

Il layout del form è predisposto per l'inserimento di un poligono, perché è il primo nella lista del menù a tendina. Se selezioniamo il cerchio il form si trasforma ed aggiunge le caselle utili per l'inserimento del vertice di origine e del raggio.

Cliccando sul pulsante aggiungi vertici viene aggiunta una riga alla tabella contenente tutti i vertici. Per rimuovere le righe inutili sarà sufficiente cliccare su rimuovi vertice.

Form per l’inserimento e la rimozione dei vertici di una figura

Selezionando una figura sul piano cartesiano vengono mostrati tutti i dettagli di quella figura nella tabella a sinistra del piano cartesiano e vengono attivati tu tti i pulsanti relativi alle operazioni possibili su un'unica figura, come ad esempio aggiungere o rimuovere i vertici.



Se viene inserito un vertice da rimuovere che non è presente nella lista, il sistema ci avviserà con un allert che ci permetterà di modificare quello inserire o di chiudere la finestra.

Guida all’utilizzo

L’utilizzo di questa GUI è molto semplice.

All’inizio, le uniche operazioni possibili sono chiudere la finestra o aggiungere una nuova figura. Selezionando “aggiungi figura” si apre il form per l’inserimento dei dati della figura.

Per l’inserimento di un poligono è possibile inserire massimo 20 vertici e minimo 3, se quest’ultima condizione non viene rispettata un allert avvisa, impedendo il proseguimento.

Se tutti i vertici sono corretti e permettono la creazione del poligono, allora viene creato e disegnato sul piano cartesiano. Per rimuovere la figura o modificare la lista dei vertici è sufficiente selezionare la figura ed utilizzare i pulsanti che vengono attivati. Quest’ultima funzionalità viene attivata solo se entrambe le figure sono poligono

Se vengono selezionate due figure (utilizzare il tasto ctrl per la selezione multipla) vengono attivati anche i pulsanti per la somma e la sottrazione. Quest’ultima funzionalità viene attivata solo se il tipo delle due figure è uguale (cerchio o poligonoconvesso).

La logica della GUI ed il polimorfismo

La GUI è stata realizzata utilizzando 8 classi che estendono alcune classi della libreria Qt.

La classe astratta Figure rappresenta la figura disegnata sul piano cartesiano ed estende la classe QgraphicsItem, ed ha come campo dati un puntatore a shape, questo sarà un

puntatore polimorfo che permetterà l'utilizzo di tutti i metodi virtuali della classe Shape e derivate.

Il polimorfismo viene utilizzato molto spesso, ad esempio per recuperare il perimetro o l'area, utilizzando direttamente un puntatore a Shape oppure se viene effettuato un cast a PoligonoConvesso è possibile utilizzare il metodo getLati per recuperare la lista dei lati.

Le classi concrete che delle figure disegnate sul grafico sono figureellipse e figurepolygon. In queste classe sono stati ridefiniti i metodi paint e boundingRect.

La classe GraphicsCoordinateAxisItem estende la classe QGraphicsItem per disegnare i due assi sul piano cartesiano con i relativi trattini e numeri.

La classe MainWindow è la classe principale che possiede quattro puntatori come campo dati, che puntano, all'interfaccia grafica della MainWindow, alla GraphicScena, al contoller, e al controller che gestisce la tabella con i dati della figura selezionata.

Nella classe del controller è presente un campo dati QList<Figure*> listaFigure che memorizza tutte le figure presenti nella GraphicsScene, un puntatore alla mainwindows ed un puntatore alla scena creata. Grazie a questi puntatori è possibile interfacciare in maniera semplice il modello e la GUI.

Materiale consegnato

La cartella principale contiene 3 elementi:

1. Cartella del progetto c++
2. Cartella del progetto Java
3. Relazione

- Cartella del progetto c++

1. Tutti i file .h
2. Tutti i file .cpp
3. File Kalk.pro

N.b: Per generare il MakeFile è indispensabile utilizzare il file Kalk.pro

- Cartella del progetto Java contiene tutti i file .java