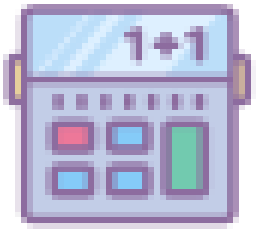
Progetto di Programmazione ad Oggetti
A.A. 2017/2018

Michele Roverato Matricola: 1143030
Francesco De Filippis Matricola: 1143408

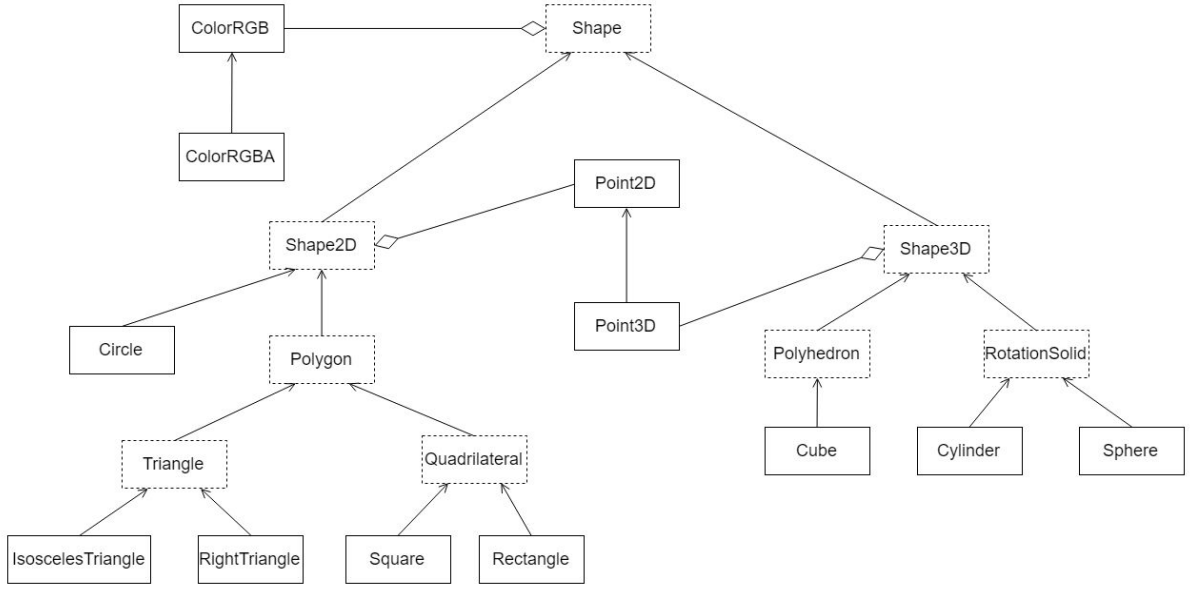
Relazione di Francesco De Filippis

Progetto: Kalk



Indice

1. Descrizione delle gerarchie



Nell'immagine sopra indicata è presente l'intera gerarchia riguardante la parte logica e funzionale del progetto, in particolare le classi tratteggiate sono classi polimorfe astratte, mentre le restanti sono concrete.

Tutte le classi T che possiedono i quattro operatori algebrici di somma, sottrazione, moltiplicazione e divisione hanno la firma di questi definita in tal modo: T* operator<+-*/>(const T&) const.
Tutte le classi T che possiedono il metodo clone() hanno la firma di questi definita come T* clone() con il seguente contratto: per ogni puntatore p a T una invocazione di p->clone() ritorna un nuovo puntatore T* che rappresenta una copia di *p.
Tutte le classi implementano o ereditano l'operatore di uguaglianza virtuale bool operator==(const T&).

ColorRGB è una classe concreta che rappresenta un colore di tipo RGB rappresentato da tre campi, uno per ciascun canale:

- double Red
- double Green
- double Blue

Tale classe implementa i seguenti metodi virtuali:

- virtual ColorRGB* clone() che è un metodo di clonazione descritto sopra
- Implementa tutti gli operatori algebrici nel modo descritto sopra

ColorRGBA è una classe concreta derivata da ColorRGB che rappresenta un colore di tipo RGBA, aggiunge quindi un campo per il nuovo canale:

- double Alpha

Tale classe implementa i seguenti metodi:

- ColorRGBA* clone() che è un metodo di clonazione
- Implementa tutti gli operatori algebrici ereditati nel modo descritto sopra

Point2D è una classe concreta che rappresenta un punto nello spazio 2D ed è caratterizzata dalle due coordinate:

- double x

- double y

Tale classe implementa i seguenti metodi:

- static double distance(const Point2D&, const Point2D&) è un metodo statico con il seguente contratto: dati due punti in input restituisce un valore che rappresenta la distanza nel piano tra il primo e il secondo punto.

Point3D è una classe concreta derivata da Point2D che rappresenta un punto nello spazio 3D e aggiunge la terza coordinata:

- double z

Tale classe implementa i seguenti metodi:

- static double distance(const Point3D, const Point3D) è un metodo statico che dati due punti in input restituisce un valore che rappresenta la distanza nel piano tra il primo e il secondo punto.
- static Point3D p3dTop3d(const Point2D&) è un metodo statico che dato un Point2D in input lo converte in un Point3D (inizializzando la coordinata z con 0) e poi lo ritorna.

Shape è una classe base astratta che serve per rappresentare tutte le figure della gerarchia ed è caratterizzata da:

- std::string name
- ColorRGB* color

Tale classe è astratta in quanto offre i seguenti metodi virtuali puri:

- virtual Shape* clone() che è un metodo di clonazione descritto sopra
- virtual unsigned int getNumberVertex() che restituisce il numero di vertici di una figura

Inoltre avendo un campo puntatore implementa la regola del tre, quindi assegnazione, copia e distruzione profonda.

Shape2D è una classe astratta derivata da Shape che rappresenta tutte le figure 2D ed è caratterizzata da:

- std::vector<Point2D> points (protetto poichè era utile e sensato che ciascuna figura 2D derivata potesse accedere ai suoi punti)

Tale classe è astratta in quanto offre i seguenti metodi virtuali puri:

- virtual Shape2D* clone() che è un metodo di clonazione descritto sopra
- Operatori algebrici nel modo descritto sopra
- virtual double sideToOperator()
- void translate(const Point2D&)

Inoltre la classe implementa il metodo virtuale puro unsigned int getNumberVertex() di Shape nel seguente modo: una invocazione di f->getNumberVertex() dove f è un puntatore poliformo contenente una classe concreta restituisce il numero di punti presenti in points.

Circle è una classe concreta derivata da Shape2D che implementa i metodi virtuali puri di Shape2D come segue:

- Circle* clone() che è un metodo di clonazione come descritto sopra
- double sideToOperator() che ritorna il raggio del cerchio
- Implementa tutti gli operatori algebrici ereditati nel modo descritto sopra

Polygon è una classe astratta derivata da Shape2D in quanto possiede i seguenti metodi virtuali puri:

- virtual double area()
- virtual double base()
- virtual double height()
- Tutti gli operatori algebrici ereditati da Shape2D
- double sideToOperator() e Shape2D* clone() ereditati da Shape2D

Quadrilateral e **Triangle** sono classi astratte derivate da Polygon in quanto possiedono i seguenti metodi virtuali puri:

- Tutti gli operatori algebrici ereditati da Polygon
- double sideToOperator() ereditato da Polygon
- Shape2D* clone() ereditato da Shape2D
- I metodi base(), height() e area() ereditati da Polygon

Square è una classe concreta derivata da Quadrilateral in quanto implementa i seguenti metodi virtuali puri come segue:

- Square* clone() che è un metodo di clonazione descritto sopra
- double sideToOperator() che ritorna il lato del quadrato
- Implementa tutti gli operatori algebrici nel modo descritto sopra
- I metodi area(), base() e height() di Quadrilateral

Rectangle è una classe concreta derivata da Quadrilateral in quanto implementa i seguenti metodi virtuali puri come segue:

- Rectangle* clone() che è un metodo di clonazione come descritto sopra
- double sideToOperator() che ritorna la somma di base e altezza del rettangolo
- Implementa tutti gli operatori algebrici ereditati nel modo descritto sopra
- I metodi area(), base() e height() di Quadrilateral

IsoscelesTriangle è una classe concreta derivata da Triangle in quanto implementa i seguenti metodi virtuali puri come segue:

- IsoscelesTriangle* clone() che è un metodo di clonazione come descritto sopra
- double sideToOperator() che ritorna la somma di base e altezza del triangolo isoscele
- Implementa tutti gli operatori algebrici ereditati nel modo descritto sopra
- I metodi area(), base() e height() di Triangle

RightTriangle è una classe concreta derivata da Triangle in quanto implementa i seguenti metodi virtuali puri come segue:

- RightTriangle* clone() che è un metodo di clonazione come descritto sopra
- double sideToOperator() che ritorna la somma di base e altezza del triangolo rettangolo
- Implementa tutti gli operatori algebrici ereditati nel modo descritto sopra
- I metodi area(), base() e height() di Quadrilateral

Shape3D è una classe astratta derivata da Shape che rappresenta tutte le figure 3D ed è caratterizzata da:

- std::vector<Point3D> points (protetto poiché era utile e sensato che ciascuna figura 3D derivata potesse accedere ai suoi punti)
- Shape2D* baseShape (protetto poiché era utile e sensato che ciascuna figura 3D derivata potesse accedere alla sua figura di base)

Tale classe è astratta in quanto offre i seguenti metodi virtuali puri:

- virtual Shape3D* clone() che è un metodo di clonazione
- Operatori algebrici nel modo descritto sopra.
- virtual double volume()
- virtual double totalSurface()

Inoltre la classe implementa il metodo virtuale puro unsigned int getNumberVertex() di Shape nel seguente modo: una invocazione di f->getNumberVertex() dove f è un puntatore poliformo contenente una classe concreta restituisce il numero di punti presenti in points.

Inoltre avendo un campo puntatore implementa la regola del tre, quindi assegnazione, copia e distruzione profonda.

Polyhedron è una classe astratta derivata da Shape3D in quanto possiede i seguenti metodi virtuali puri:

- Tutti gli operatori ereditati da Shape3D
- virtual double volume() ereditato da Shape3D
- virtual double totalSurface() ereditato da Shape3D
- virtual Shape3D* clone() ereditato da Shape3D
- virtual double lateralSurface()
- virtual double getNumberCorner()
- virtual double getNumberFaces()

RotationSolid è una classe astratta derivata da Shape3D in quanto possiede i seguenti metodi virtuali puri:

- Tutti gli operatori ereditati da Shape3D
- virtual double volume() ereditato da Shape3D
- virtual double totalSurface() ereditato da Shape3D
- virtual Shape3D* clone() ereditato da Shape3D

Cube è una classe concreta derivata da Polyhedron in quanto implementa i seguenti metodi virtuali puri come segue:

- Cube* clone() che è un metodo di clonazione come descritto sopra
- Tutti gli operatori algebrici ereditati nel modo descritto sopra
- I metodi getNumberCorner(), getNumberFaces(), lateralSurface(), totalSurface(), volume() di Polyhedron

Cylinder è una classe concreta derivata da RotationSolid in quanto implementa i seguenti metodi virtuali puri come segue :

- Cylinder* clone() che è un metodo di clonazione come descritto sopra
- Implementa tutti gli operatori algebrici ereditati nel modo descritto sopra
- I metodi, totalSurface(), volume() di RotationSolid

Sphere è una classe concreta derivata da RotationSolid in quanto implementa i seguenti metodi virtuali puri come segue :

- Sphere* clone() che è un metodo di clonazione come descritto sopra
- Implementa tutti gli operatori algebrici ereditati nel modo descritto sopra
- I metodi totalSurface(), volume() di RotationSolid

Sono presenti inoltre altre due gerarchie, una per la GUI e una per MVC pattern.

Gerarchia GUI

- **MoreInfoShape** è una classe base astratta che fa da interfaccia comune per le finestre che visualizzano le informazioni delle figure(operandi e risultato).
- **About2DCircle, About2DSquare, About2DRectangle, About2DTriangleIso, About2DTriangleRet, About2DCube, About2DCylinder, About2DSphere** sono classi concrete derivate da MoreInfoShape che implementano dei metodi per inizializzare i componenti grafici con i dati della figura utilizzando alcuni metodi protetti della classe base.
- **CreateShape** è una classe base astratta che fa da interfaccia comune per la creazione di finestre necessarie per l'inserimento dei dati da parte dell'utente.

- **Create2DCircle, Create2DSquare, Create2DRectangle, Create2DTriangleIso, CreateDTriangleRet, Create2DCube, Create2DCylinder, Create2DSphere** sono classi concrete derivate da CreateShape che aggiungono i componenti per gestire l'input dell'utente utilizzando alcuni metodi protetti della classe base.
- **ShapePanel** è una classe base astratta che fa da interfaccia comune per i pannelli che contengono i tipi con i quali è possibile svolgere i calcoli. In particolare si occupa di gestire un numero potenzialmente infinito di tipi in quanto è stato implementato un sistema modulare di pagine tra cui navigare attraverso dei pulsanti.
- **ShapePanel2D, ShapePanel3D** sono classi concrete che estendono ShapePanel che contengono i tipi concreti presenti attualmente in Kalk con i quali è possibile svolgere calcoli.

Gerarchia MVC

In questa gerarchia sono presenti le seguenti classi:

- **ShapeController** è una classe base astratta che fa da interfaccia comune per tutti controller.
- **Shape2DController** è una classe astratta derivata da ShapeController. Oltre ad implementare la regola del tre e un metodo di clonazione, offre dei metodi virtuali puri che rappresentano le operazioni che si possono fare sulle figure 2D che servono per far interagire la GUI con i model(che sono costituiti dalla parte logica).
- **CircleController, SquareController, RectangleController, IsoscelesTriangleController, RightTriangleController** derivate da Shape2DController sono classi concrete che implementano tutti i metodi virtuali puri di Shape2DController e alcuni metodi propri per ottenere informazioni su caratteristiche aggiuntive specifiche di ciascuna figura. Ciascuna di esse ha un campo model del tipo della figura(es. Circle*, Square*, ecc.).
- **Shape3DController** analoga a Shape2DController e con la stessa tipologia di metodi virtuali puri.
- **CubeController, CylinderController, SphereController** derivate da Shape3DController sono classi concrete che implementano tutti i metodi virtuali puri di Shape3DController e alcuni metodi propri per ottenere informazioni su caratteristiche aggiuntive specifiche di ciascuna figura. Ciascuna di esse ha un campo model del tipo della figura(es. Cube*, Cylinder*, ecc.).
- **Shapeview** è una classe base astratta che fa da interfaccia comune per tutte le view.
- **Shape2DView** è una classe astratta derivata da ShapeView e possiede un campo CalculatorInterface2D* view che rappresenta la view e altri campi che permettono di gestire lo stato degli operandi nella GUI(per esempio chi è il primo operando, se ha subito modifiche, ecc.) e possiede una serie di metodi per controllare l'input dell'utente e scambiare i dati tra la GUI e i controller.
- **CircleView, SquareView, RectangleView, IsoscelesTriangleView, RightTriangleView** sono classi concrete derivate da Shape2DView che implementano i metodi virtuali puri ereditati. Possiedono un campo di tipo Controller congruo con il tipo della view (CircleView ha un CircleController).
- **Shape3DView** analoga a Shape2DView ma con campo CalculatorInterface3D* view.
- **CubeView, CylinderView, SphereView** sono classi concrete derivate da Shape3DView che implementano i metodi virtuali puri ereditati. Possiedono un campo di tipo Controller congruo con il tipo della view (SphereView ha un SphereController).

2. Descrizione uso codice polimorfo

- Nella gerarchia ColorRGB viene sfruttato il late binding per chiamare il metodo clone(), e gli operatori algebrici. Il metodo clone in particolare viene utilizzato da Shape per effettuare le copie profonde, mentre gli operatori gli algebrici vengono utilizzati nei controller per effettuare le operazioni.
- Operatori algebrici utilizzati nei calcoli poiché a run-time viene selezionato l'operatore corrispondente al tipo dinamico del primo operando.
- Negli operatori delle figure geometriche 2D, in particolare nelle classi concrete, viene invocato il metodo sideToOperator() su un oggetto di tipo Shape2D e con il late binding viene ritornato il valore di questa funzione in base al tipo dinamico della variabile di tipo Shape2D, ovvero la figura concreta che istanziata al suo interno.
- Distruttore virtuale profondo nella classe Shape e distruttore profondo nella classe Shape2D, distruttore virtuale in ColorRGB e Point, in modo da non lasciare garbage in memoria.

3. Ambiente di sviluppo e di test e istruzioni di compilazione

- Sistema operativo: Windows 10 Pro 64-bit
- Compilatore: MinGW 5.3.0 32-bit
- Libreria Qt: 5.9.3

- Sistema operativo: Ubuntu 16.04 64-bit
- Compilatore: gcc 5.4.0
- Libreria Qt: 5.5.1

Oltre ai files .h e .cpp contenenti il codice sorgente, un file relazione.pdf, viene fornito progetto.pro necessario per la corretta compilazione del progetto. Di conseguenza per compilare il codice sarà sufficiente lanciare qmake a make dentro la cartella ./cpp/KalkCpp/. Per compilare Java, è sufficiente eseguire il comando javac -classpath ":." *.java nella classe ./Java/src/ (Questo nei sistemi unix-like, in Windows ":." diventa ";.")

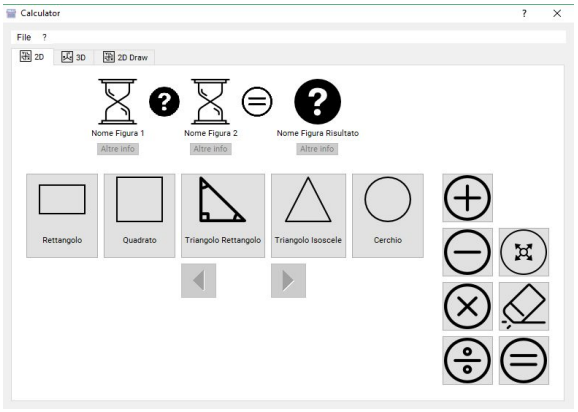
4. Suddivisione del lavoro progettuale e tempistiche

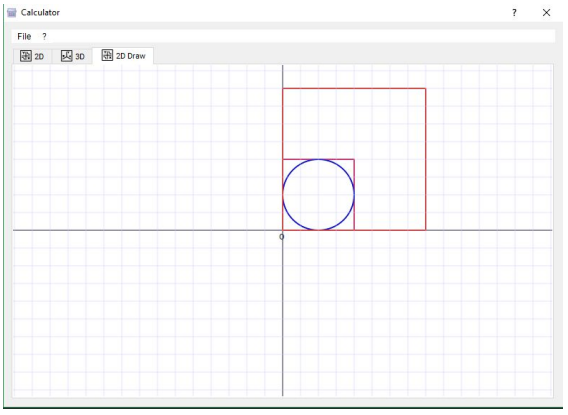
L'applicazione è stata progettata e discussa in stretto contatto da entrambi i partecipanti, a partire dall'analisi del problema fino alla sua implementazione.
Ai fini pratici quello che concerne la parte logica dell'applicazione è stata sviluppata da Francesco De Filippis, quindi tutto l'albero gerarchico delle figure geometriche sia in C++ che in Java.
L'interfaccia grafica compresi views e controllers è stata sviluppata da Michele Roverato ovvero tutta la gerarchia dei vari componenti grafici e la gestione di MVC.
Tempistiche(ore): analisi preliminare del problema(2), progettazione modello e GUI(2), apprendimento libreria Qt(6), codifica modello(30), debugging(9), testing(6). Le ore aggiuntive sono dovute all'apprendimento degli IDE utilizzati e al refactoring del codice.

5. Manuale utente della GUI

All'avvio l'applicazione si presenta come l'immagine a sinistra, l'interfaccia è stata progettata per essere il più pulita e intuitiva possibile. In alto a sinistra sono presenti due menù a tendina File e "?".

Su File è possibile aprire le impostazione ed è possibile uscire dall'applicazione con un tasto "Esci".
Su "?" è possibile visualizzare delle informazioni sull'applicazione e una guida che spiega brevemente come usarla.
In basso a File sono presenti i pannelli in cui ci sono i tipi di calcolo e le operazioni offerte dalla calcolatrice.
A destra il pannello "Draw" descritto al punto successivo.





Le operazioni (sia con le figure 2D che 3D) si svolgono in questo modo:

- Si clicca su uno dei bottoni con le figure e nella finestra che si apre si inseriscono i dati della figura. Questa operazione si fa due volte, una per il primo operando e una per il secondo.
- Successivamente si preme uno dei pulsanti a destra nella prima colonna per scegliere l'operatore desiderato.
- Infine si preme il tasto "=", l'ultimo in basso a destra per visualizzare il risultato tramite una finestra che si aprirà. (I tasti "Altre info" sotto le figure nella parte alta aprono una finestra che visualizza le informazioni della figura).
- Inoltre è presente un tasto a forma di gomma nella seconda colonna che permette di cancellare tutto lo stato del calcolo attuale.
- Nel pannello "Draw" è presente un piano cartesiano in cui sono disegnate le figure del calcolo ovvero primo e secondo operando, e risultato evidenziate con il loro colore.

Lo stesso procedimento è analogo per le figure 3D, eccetto il pannello di disegno e il primo tasto della seconda colonna che non è disponibile per le 3D.
Il primo tasto della seconda colonna nel pannello del calcolo delle figure 2D permette di effettuare la traslazione di una figura a scelta tra quelle presenti negli operandi e nel risultato e poi visualizzare il risultato nel pannello "Draw". Le modalità di utilizzo sono analoghe alla creazione di una figura.