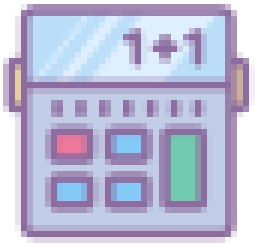
Progetto di Programmazione ad Oggetti
A.A. 2017/2018

Michele Roverato Matricola: 1143030
Francesco De Filippis Matricola: 1143408

Relazione di Michele Roverato

Progetto: Kalk

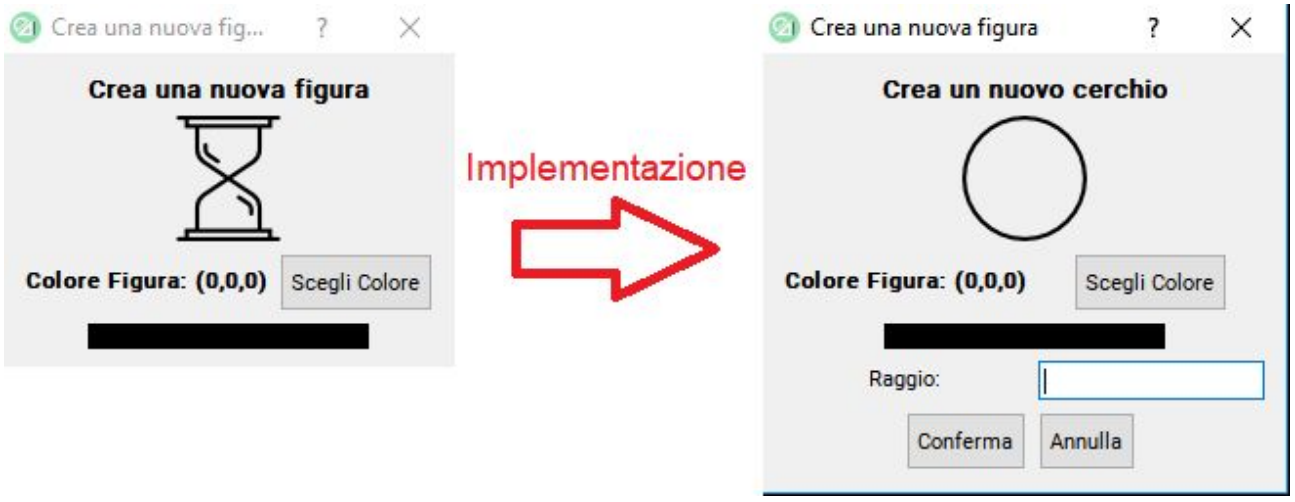


Indice

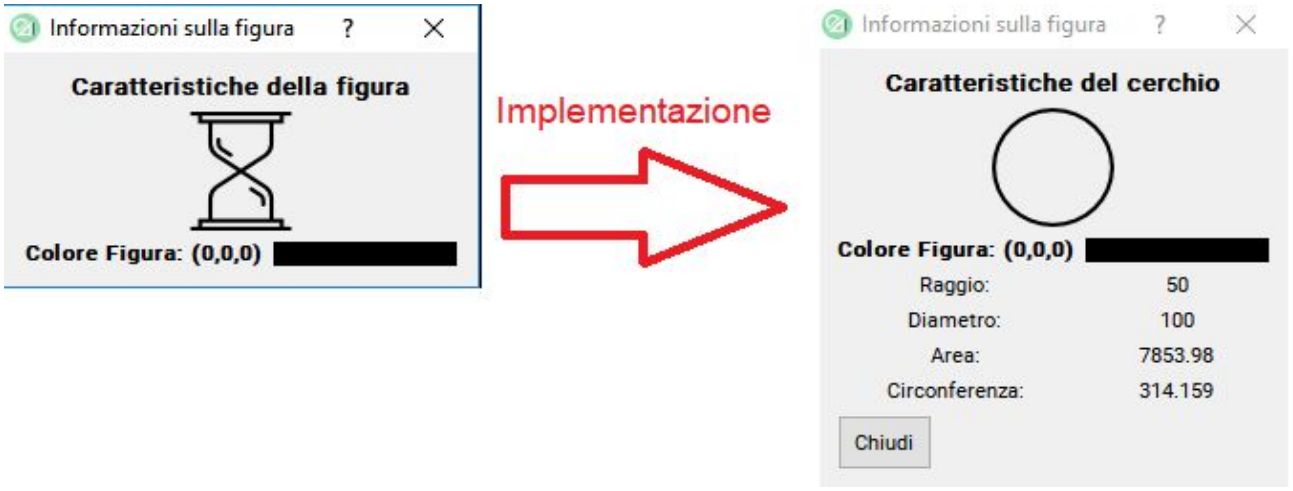
1. Descrizione gerarchie

Gerarchie di Finestre/Pannelli della UI

CreateShape è la classe base utilizzata per creare tutte le altre finestre utilizzate per ricevere i dati di creazione di una figura. Offre una finestra base che contiene un titolo, un'immagine, la scelta del colore che sarà assegnato alla figura, ed eventualmente i bottoni "Conferma" e "Annulla" che possono essere aggiunti mediante metodo apposito.
Per ogni figura è stata quindi creata una classe CreateNDShape, implementata aggiungendo alla finestra base delle caselle di testo in cui inserire i valori, per esempio la finestra Create2DCircle avrà la casella di testo in cui inserire il raggio del cerchio.
Inoltre ogni implementazione ha la responsabilità di gestire il bottone di conferma (se creato), in quanto il suo slot è dichiarato come astratto. Di conseguenza, ogni finestra Create implementa uno slot Confirm(), che controllerà la validità dei dati inseriti prima di passarli con sicurezza alla View.

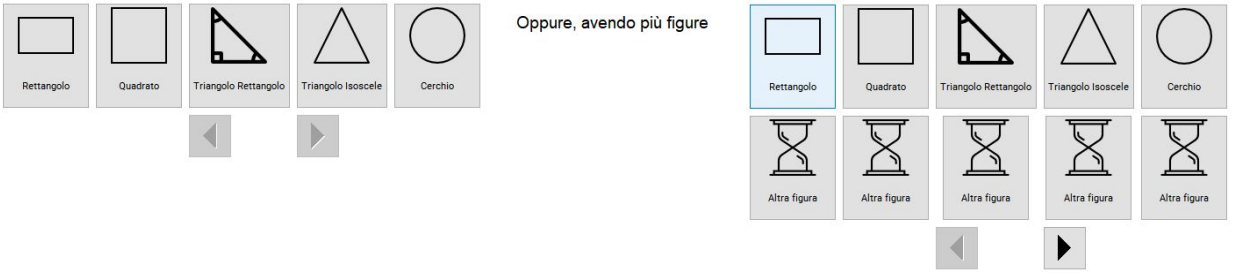


MoreInfoShape è la classe base utilizzata per creare le finestre che rappresenteranno le informazioni di una figura. La finestra base offre un titolo, un'immagine e la rappresentazione del colore della figura. Inoltre è possibile aggiungere mediante metodo il bottone "Chiudi" assegnandogli anche il comportamento di default.
Per ogni figura quindi ho creato una AboutNDShape, implementata aggiungendo alla finestra base delle QLabel che rappresenteranno le informazioni della figura, passate mediante opportuni metodi set(…).



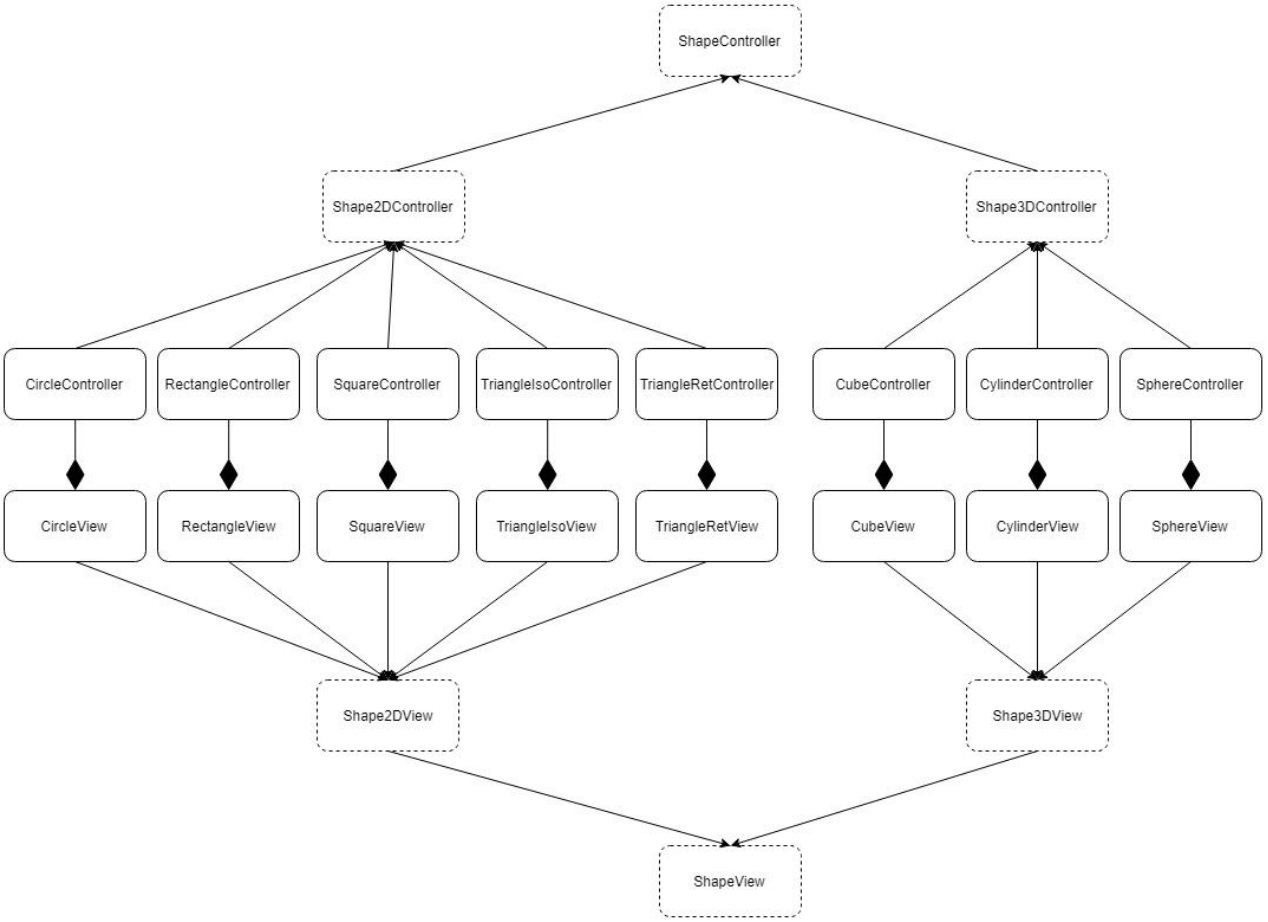
ShapePanel è un pannello, realizzato per rendere più comoda l'aggiunta di figure. Questo pannello infatti, richiede un vector di bottoni e indipendentemente da quanti ne vengono inseriti, è fatto per adattarsi e mostrarne al massimo 10 per pagina. Se sono presenti più di 10 bottoni vengono attivati i pulsanti per cambiare pagina.

Questo pannello è stato implementato per le figure 2D e 3D, ottenendo quindi le classi ShapePanel2D e ShapePanel3D che creano i propri bottoni per le figure.



CalculatorOperations è un pannello che contiene i pulsanti con le possibili operazioni che si possono svolgere. È stato esteso in quanto le figure 2D possono essere traslate, quindi una sua sottoclasse ne aggiunge un pulsante.

Gerarchia del Model-View-Controller



(Purtroppo il secondo albero è dal basso verso l'alto per motivi di spazio)

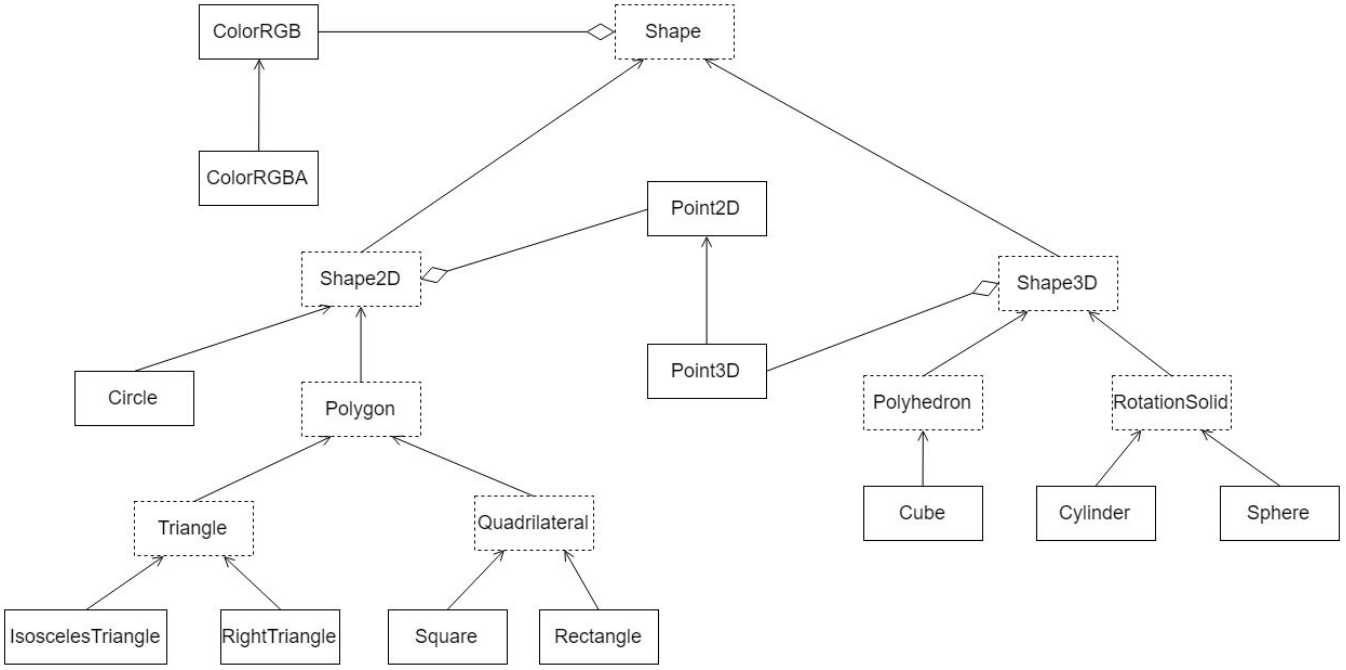
ShapeController è la classe base astratta dei controller, in questo caso vuota, estesa da **Shape2DController** e da **Shape3DController**. Queste due contengono le definizioni astratte dei metodi che ogni controller dovrà implementare. In queste due classi alcuni metodi sono simili e differiscono per parametri e/o tipi di ritorno, come sum(…) o sub(..) per richiamare gli operatori algebrici sui model. Altri metodi invece sono caratteristici, come Translate(…) che è esclusivamente per le figure 2D, e quindi solo nel controller 2D.

Ogni Controller concreto definisce quindi tutti i metodi astratti, inoltre contiene un riferimento al suo model, che (per esempio) nel caso di CircleController sarà un Circle*. Utilizzando questo riferimento, il controller definisce altri metodi caratteristici della figura, quali base, area, circonferenza e altri.

ShapeView è la classe base astratta per le view, non contiene nulla, ma è estesa da **Shape2DView** e da **Shape3DView** che nell'header definiscono vari metodi astratti. Inoltre contengono dei riferimenti a componenti necessari alla view, tra cui un riferimento alla UI principale e ad una finestra CreateShape, per l'input dei dati della figura. Contengono anche qualche flag, per esempio per segnare se una figura è utilizzata come primo o secondo operando.

Ogni View concreta quindi, definisce tutti i metodi astratti definite dalle View sopra di lei, contiene inoltre un riferimento al proprio Controller, che verrà' creato solo una volta ricevuti i dati dalla finestra di input. Il controller sarà poi utilizzato per leggere le informazioni sulla figura e svolgere le varie operazioni.

Gerarchia della parte logica (Realizzata dal mio compagno)



Un Point2D è semplicemente rappresentato dalle coordinate (x,y). Un Point3D è un Point2D a cui è aggiunta la coordinata z. Entrambi contengono l'operatore di uguaglianza, ed anche un metodo statico per calcolare la distanza tra due punti dello stesso tipo.

Un ColorRGB è rappresentato dal valore red, green e blue del colore. Un ColorRGBA è un ColorRGB con l'aggiunta del valore alpha. Entrambi contengono l'operatore di output, di uguaglianza, e gli operatori aritmetici.

La parte logica è composta da una gerarchia di Shape, divise in Shape2D e Shape3D, tutte e tre astratte. Tutte le Shape hanno un nome (rappresentato da una semplice stringa) ed un colore(ColorRGB*), una Shape2D è composta da un insieme di Point2D mentre una shape3d è composta da una Shape2D di base e un insieme di Point3D.

Shape2D e Shape3D definiscono alcuni metodi astratti, come gli operatori, che ogni Shape concreta implementerà.

Una shape concreta quindi, implementa tutti i metodi astratti, tra cui: gli operatori matematici (+,-,*,/) e il metodo clone() per avere una copia della figura.

Ogni Shape concreta inoltre, ha un costruttore adatto ai dati di cui necessita (per esempio solo il raggio per il cerchio), ma anche un costruttore che richiede direttamente la lista di punti della figura. I dati passati al costruttore sono controllati da un metodo, che definisce se sono validi o meno. Se non sono validi, costruisce una figura unitaria. I dati inseriti da UI tuttavia, sono controllati dalla View, che quindi passerà al Controller ed al model soltanto dati validi.

2. Descrizione uso codice polimorfo

Le classi CalculatorInterface2D e CalculatorInterface3D contengono un riferimento generico ad Shape2DView e Shape3DView. Qui verrà utilizzato il late binding per accedere al Controller corrispondente di quelle View su cui svolgere le operazioni, utilizzando il metodo getController(). Il riferimento generico alla ShapeView viene anche utilizzato in late binding per aprire la finestra di altre informazioni sulla figura, utilizzando il metodo moreInformation().

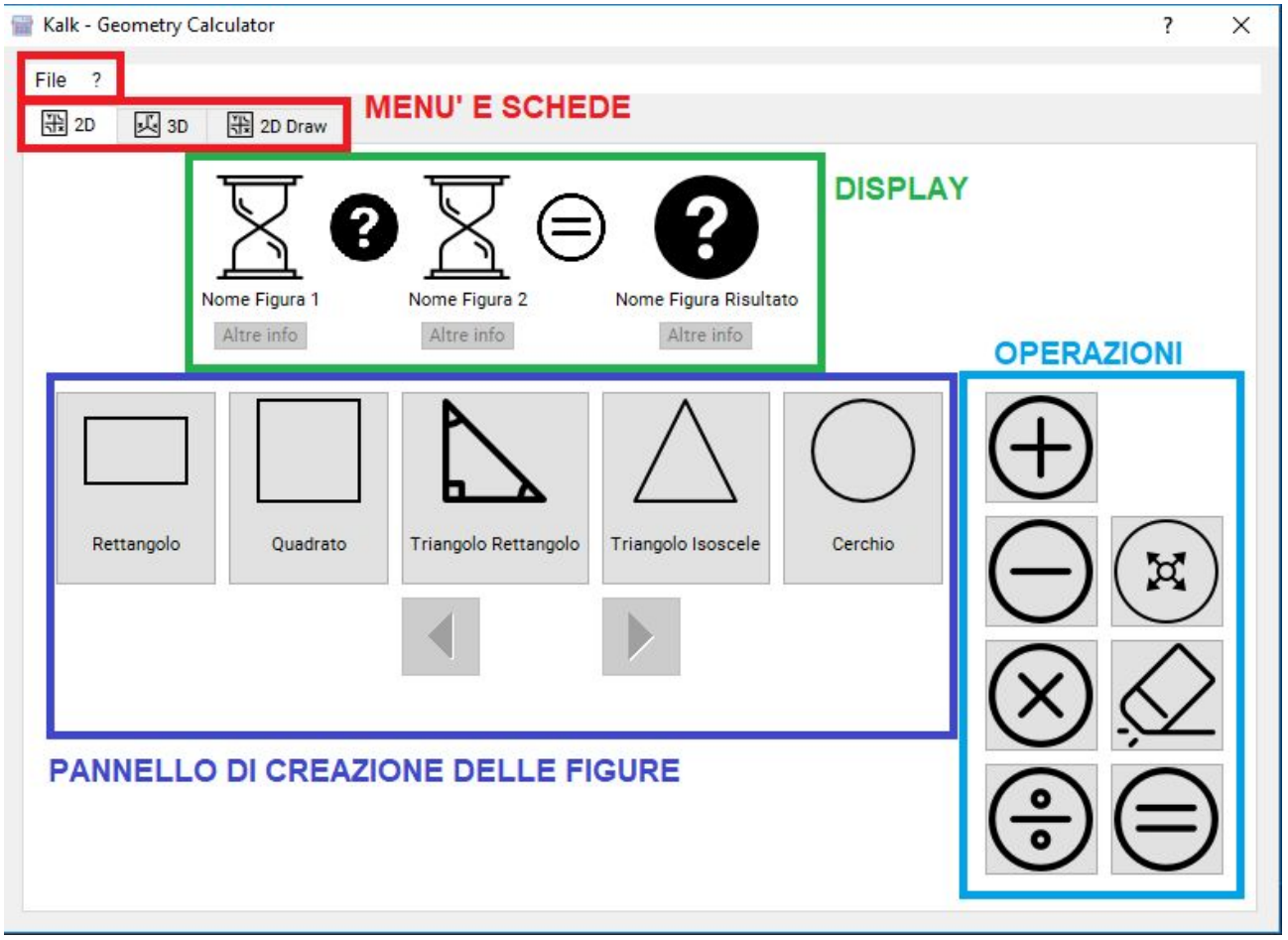
La finestra che riceve i dati di input per la traslazione, contiene una Shape2DView generica, su cui chiama in late binding ReceiveTranslateData().

Le View ed i Controller hanno inoltre il metodo virtuale clone() necessario per una corretta copia in late binding, in quanto nella UI ho bisogno di una copia della Shape2D/3DView di cui non conosco il tipo dinamico. Anche nella parte logica del programma sono presenti i metodi clone() per copiare correttamente una figura.

Nella parte logica inoltre, tutti gli operatori algebrici sono virtuali, e sfruttano il metodo sideToOperator() invocato tramite late binding da una Shape2D/3D generica, per ottenere il valore da utilizzare nella operazione.

Infine ci sono vari distruttori virtuali in classi base e non, per evitare di lasciare qualsiasi tipo di garbage in memoria.

3. Manuale utente della GUI



Una volta avviato il programma, la UI si presenta così all'utente.

Il menù in alto consente di aprire varie finestre come: la guida, le opzioni o le informazioni su di noi.

Le schede appena sotto, consentono di cambiare la modalità di calcolo, quindi tra figure 2D, tra figure 3D, o di visualizzare le figure 2D create nel piano cartesiano (Scheda 2D Draw).

Il pannello di calcolo per le figure 2D (mostrato in figura) è diviso in 3 semplici parti:

- **Il display** della calcolatrice: che mostrerà gli operandi, l'operatore corrente e il risultato dell'operazione, consentendo di cliccare su "Altre info" per visualizzare altre informazioni sulla figura.
- La **lista delle operazioni** è un pannello contenente tutte le operazioni offerte, nel caso dell'immagine sono +,-,*,/,= ed inoltre la traslazione di una figura e la cancellazione completa del display. Naturalmente non si può traslare una figura che non è presente nel display e svolgere l'operazione se non sono presenti i due operandi e l'operatore.
- Il **pannello con la lista delle figure** è un pannello che contiene la lista di tutte le figure che si possono creare, una volta cliccato su una, sarà possibile inserire i dati e crearla.

L'interfaccia di calcolo per le figure 3D è molto simile a questa, tuttavia non offre la traslazione come operazione e offre un numero minore di figure a disposizione.

L’interfaccia che mostra le figure nel piano cartesiano è composta da una semplice griglia e dagli assi X,Y. Una volta creata almeno una figura nel pannello delle figure 2D sarà possibile visualizzarla qui con il corrispondente colore. Questa interfaccia si aggiorna solo quando viene creata, cancellata o traslata una figura, non impattando sulle performance.

4. Ambiente di sviluppo e di test

A casa:

- Sistema operativo: Windows 10 Pro 64-bit
- Compilatore: MinGW 5.3.0 32-bit
- Libreria Qt: 5.9.2

In laboratorio:

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

Istruzioni di compilazione

Nel progetto abbiamo deciso di lasciare e consegnare il file .pro (progetto.pro), in quanto abbiamo aggiunto alcuni parametri, come un’istruzione per utilizzare C++11 in quanto facciamo ampio uso di keyword come *nullptr*.

Di conseguenza per compilare il codice sarà sufficiente fare **qmake** a **make** dentro la cartella ./cpp/KalkCpp/.

Per compilare Java, è sufficiente eseguire il comando **javac -classpath ":." *.java** nella cartella ./Java/src/ (Questo nei sistemi unix-like, in Windows “:.” diventa “;.”)

5. 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 la parte logica dell’applicazione è stata sviluppata da Francesco De Filippis, mentre l’interfaccia grafica compresi views e controllers è stata sviluppata da Michele Roverato.

Analisi delle Tempistiche:

- **analisi preliminare del problema(~02h);**
- **progettazione modello e GUI(~03h);**
- **apprendimento libreria Qt(~10h);**
- **codifica modello e GUI(~30h);**
- **debugging(~05h);**
- **testing(~05h).**

Il quantitativo di ore richiesto è stato appena superiore di quello previsto (circa 5 ore), in quanto è stato necessario del tempo aggiuntivo per trovare e capire come implementare alcune funzionalità della libreria Qt.