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

**----- Matricola: -------**

**----- Matricola: -------**

**Relazione di -------**

**Progetto: Kalk**

**Indice:**

1. **Abstract**
2. **Descrizione gerarchia**
3. **Manuale utente**

**1 Abstract**

La calcolatrice Kalk effettua operazioni sulle figure, precisamente su punti, rette e poligoni fino a quattro lati.

Con questi tipi può effettuare tali operazioni:

intersezione

distanza tra due punti

distanza generica tra due elementi

il perimetro

l’area

data un punto e una retta vedere se una retta è perpendicolare.

Tutte le figure sono costituite da punti dove a loro volta sono composti da coordinate definite tramite razionali per assicurare maggiore precisione.

La gerarchia è stata pensata e costruita in modo tale che chi volesse, in seguito, può ampliarla aggiungendo il proprio tipo nel modello a patto che implementi operazioni adeguate per il proprio tipo se non dovessero bastare quelle già presenti. Inoltre, con pochissime linee di codice aggiuntive è possibile aggiungere nella barra laterale di sinistra nuove azioni sul nuovo tipo di dato.

**2 Descrizione gerarchia**

**MODEL**

La gerarchia per il modello è la seguente (verrà trattata superficialmente perché è compito del mio compagno.

Triangolo

Razionale

Quadrato

Retta

Punto

Poligono

InputItem

L’utente inserisce una stringa nel box per l‘input, se questo sarà corretto non verrà sollevata nessuna eccezione ed il parser ritornerà un elemento di tipo inputItem. A questo punto verranno fatti i dovuti controlli a Runtime per capire di che elemento grafico stiamo parlando (punto, retta, triangolo, ecc.) e verrà disegnato. Infine si potrà selezionare un’azione con il tipo ricevuto sulla barra laterale sinistra.

Per questo motivo ogni elemento della gerarchia è figlio di InputItem ed è sempre per questo motivo che inputItem è una classa virtuale astratta.

**VIEW**

Barra superiore: QToolBar

Lo schema per la gui è la seguente.

Viene schematizzata solo “finestra” e “mainGui” (ogni entità ha [mini descrizione: nome classe]), le altre schermate sono immediate.

Dopo questa panoramica verranno descritte solo le classi interessanti dal punto di vista ingegneristico.

Gli elementi di “finestra” sono caratterizzati dal colore grigio, tutti gli altri elementi appartengono a “mainGui.h”, quest’ultima è contenuta in “finestra.h”.

Barra laterale: QToolBar

Errori dei vari slot: QLabel

Slot input: myQline

Slot input: myQline

Slot input: myQline

Oggetti Salvati:

QListView

Grafico: QCustomPlot

Display: QLineEdit

Bottoni di uso basilare

**finestra.h**

finestra.h agisce da pseudo-controller. Permette di collegare il model con la view ed inoltre collega le preferenze (impostazioni.h) ed il wizard di benvenuto (“wizard.h). Un esempio utile di aver ingegnerizzato in questo modo la struttura è il check all’uscita delle finestre aperte, se si chiude la finestra principale verranno chiuse a cascata anche le altre (motivo per cui impostazioni.h e wizard.h sono state incapsulate in uno smart pointer).

**grafico.h**

La classe grafico.h è ereditata da QCustomPlot. **Classe che non fa parte del framework** di Qt per noi fondamentale poiché, usando solo i metodi strettamente necessari ci permette di disegnare gli elementi da noi parsati. Abbiamo optato per QCustomPlot per due ragioni: è un “semplice” file sorgente che garantisce assoluta compatibilità ma soprattutto perché avremmo sforato di troppo il tempo a disposizione per creare una struttura di plotting così complessa andando così fuori tema. I Warning presenti durante la compilazione sono dovuti solo a questa classe ed per l’appunto non essendo di nostra proprietà ci sentiamo esclusi dal mettere mano a questa parte di codice.

**myQLine.h**

Degna di nota è la classe myQline.h derivata direttamente da QLineEdit.h della libreria grafica di Qt. Questa scelta è stata dettata dalla implementazione del drag and drop. L’idea era quella di poter trascinare i risultati salvati da una QListView.h direttamente su uno degli slot di input ed il modo migliore era questo, ovvero ereditare la classe per poter ridefinire dragMoveEvent e dropEvent i due metodi virtuali protetti fondamentali per questo scopo.