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

**Favaro Marco Matricola: 1123187**

**Clerici Michele Matricola: 1123….**

**Relazione di Favaro Marco**

**Progetto: Kalk**

**Indice:**

1. Abstract
2. Descrizione/uso gerarchia e codice polimorfo
3. Manuale utente
4. Analisi delle tempistiche
5. Suddivisione del lavoro

**1 Abstract**

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

Può calcolare la distanza e l’intersezione fra tutti i tipi, calcolare l’area e il perimetro dei poligoni, la retta passante per due punti, e, dati un punto P e una retta R può calcolare la retta passante per P e parallela o perpendicolare a R.

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/uso gerarchia e codice polimorfo**

**MODEL**

La gerarchia utilizzata nel progetto rappresenta, come spiegato nell’Abstract, più figure geometriche “dominate” da una classe padre Inputitem.

Triangolo

Razionale

Punto

Retta

Quadrato

Poligono

InputItem

**RAZIONALE:**

La classe razionale specifica il dato su cui andremo a fare i calcoli, rendendo più semplice le operazioni e più leggibile, lato utilizzo, la lettura dei risultati e l’inserimento dei dati.

Sono presenti due attributi privati di tipo double: *num* e *den* (rispettivamente numeratore e denominatore). La classe fornisce i metodi di interrogazione *getnum*() e *getden*() , un costruttore di default, un costruttore a un double , uno a due double e un costruttore con due parametri di tipo razionale.

All’interno dei costruttori sono invocati due metodi: riduzione() che viene invocata con lo scopo di ridurre la frazione creata, mentre conteggio() (invocata solo nel costruttore a un double e di tipo statica perché non necessità di un oggetto di invocazione) verifica e calcola quanti numeri dopo la virgola sono presenti nel double per trasformare, per esempio, 5.7 in 57/10 facilitando poi eventuali operazione di riduzione.

Sono presenti gli overloading di moltiplicazione con double e razionale, operatore di sottrazione, somma e divisione con un razionale, operatori di incremento (postfisso e prefisso), operatore di uguaglianza con double e razionale, un metodo converti che converte l’oggetto in un double, infine un metodo tostring() che converte un razionale in una stringa (utilizzato nella gui ).

Tutti i metodi e overloading sono costanti per non verificare side-effect sugli oggetti di invocazione, il mero scopo degli operatori e metodi è eseguire calcoli e ritornare nuovi oggetti.

**INPUTITEM :**

E’ la classe base astratta che ha come unico scopo dare un’identità iniziale all’input inserito dall’utente. La classe non ha campi dati ma solo la definizione e implementazione di due metodi statici *pars\_start()* e *iniz\_input()* e la definizione di due metodi virtuali puri, *intersect()* e *distance().*

Le prime due sono statiche per il semplice motivo che vengono invocate quando l’oggetto deve ancora essere creato. La funzione pars\_start() ha il compito di identificare il tipo di figura in input, svolge semplicemente un’operazione di parsing preliminare sulla stringa inserita. Il tipo di ritorno può essere un punto, una retta, o 0. Pars\_start() viene invocato all’interno di iniz\_input() dove, in base al suo tipo di ritorno decide se invocare il parser di punto , retta o poligono e quindi creando effettivamente l’oggetto che verrà ritornato al chiamante di iniz\_input().

I due metodi virtuali puri sono implementati all’interno di punto, retta e poligono.

Il distruttore della classe è virtuale per permettere la distruzione di oggetti in presenza di puntatori polimorfi.

**PUNTO :**

**3 Manuale utente**

Non è necessario il manuale. All’apertura di Kalk verrà visualizzato un wizard che aiuterà l’utente nel prendere dimestichezza con l’interfaccia e nell’inserire in maniera corretta l’input desiderato.

**4 Analisi delle Tempistiche**

Verranno specificate le tempistiche soggette alla mia parte. La soglia di ore disponibili è stata leggermente sforata. Su certi punti verrà giustificato l’ammontare di ore.

● analisi preliminare del problema(~05h);

● progettazione modello e GUI(~20h);

Diversi dubbi su come implementare lo schema model-view.

Problema su come implementare il grafico. Dopo diversi tentativi, come già specificato abbiamo optato per un codice sorgente esterno.

● apprendimento libreria Qt(~15h);

Ho cercato a fondo soluzioni per implementare il drag and drop e per implementare il sistema di salvataggio

● codifica modello ~~e~~ GUI(~13h);

● debugging(~05h);

Problemi con la gestione del garbage.

● testing(~03h).

**5 Suddivisione del lavoro**

L’applicazione è stata progettata e discussa in stretto contatto da entrambi i partecipanti, a partire dall’analisi del problema fino alla sua implementazione. Inizialmente mi sono occupato dei primi abbozzi di gerarchia ma successivamente questa parte è stata completamente realizzata dal mio collega. Io, invece, mi sono dedicato principalmente alla progettazione e alla realizzazione della gui con tutti i problemi e le decisioni ad essa annessi che vanno, ad esempio, dalla scelta di utilizzare QCustomPlot fino alla gestione del sistema di salvataggio delle preferenze personali. Il mio compagno, invece, si è occupato della realizzazione del modello sia in c++ che in java inclusa la completa gestione delle eccezioni. La fase di test e di debugging è stata svolta assieme, soprattutto la gestione del garbage.

**4 Ambiente di sviluppo e di test**

**A casa:**

● Sistema operativo: Mac OS 10.13.4

● Compilatore: clang\_64bit

● Libreria Qt: 5.5.1

**In laboratorio:**

● Sistema operativo: Ubuntu 16.04 64-bit

● Compilatore: gcc 5.4.0

● Libreria Qt: 5.5.1

**6 Comandi per la compilazione ed esecuzione**

Il progetto viene presentato con una cartella col progetto in C++ ed una col progetto in Java

* **C++** Spostarsi col terminale nella cartella apposita (utilizzare il file .pro incluso) e dare i comandi: qmake make ./Kalk
* **Java** Spostarsi col terminale nella cartella apposita e dare i comandi javac…