Università degli studi di Padova Dipartimento di Matematica

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

Nome Progetto: Kalk

Componenti:

Nome: Damien
Cognome: Ciagola
Matricola: 1099107

Indice:

1) Scopo del progetto:

Lo scopo del progetto KALK è lo sviluppo in C++ di una calcolatrice estendibile dotata di una interfaccia utente grafica (GUI) sviluppata in Qt. La calcolatrice è in grado di gestire i calcoli di 4 diversi tipi di dati:

1) Binario
2) Esadecimale
3) Ottale
4) Frazioni

Tutti e quattro i tipi permettono le operazioni elementari (+,-,*,/), e la conversione dal tipo scelto in base10, e viceversa, Si possono quindi:

- Sommare,Sottrarre Moltiplicare e Dividere due tipi Binario/Esadecimale/Ottale e Frazionario
- Convertire un tipo Binario/Esadecimale/Ottale e Frazionario del suo corrispettivo numero Decimale
- Convertire un numero Decimale in uno dei 4 tipi sopra elencati.
- I tipi: Binario, Esadecimale e Ottale permettono inoltre il calcolo della radice quadrata.

2) Ambiente di sviluppo:

Sistema Operativo: Ubuntu 16.10 64-bit

- Compilatore: GCC 6.2.0
- Versione libreria Qt: 5.5.1

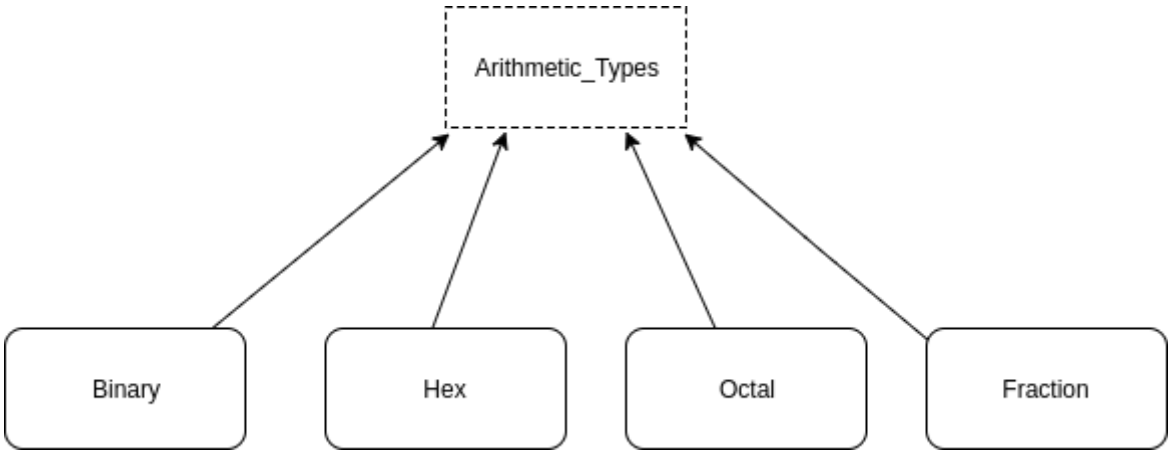
3) Ore di lavoro: 50

- Modello: 10 ore (3 ore di progettazione + 7 ore di codifica)
- Controller: 5 ore (2 ore di progettazione + 3 ore di codifica)
- View: 20 ore(5 ore di progettazione + 15 ore di codifica)
- Debug: 5 ore(Tutte impiegate sulla GUI e test per memory leak)
- Java: 10 ore (2 ore di progettazione + 4 ore comprensione della sintassi + 4 ore Debug)

4) Descrizione gerarchia:

Di seguito, verranno elencate le classi e gerarchie presenti nel progetto, e le relative caratteristiche di ognuna di esse.

modello:



Arithmetic_Types: E' la classe base polimorfa astratta, che funge da interfaccia comune per le classi derivate da essa. Gli oggetti "Arithmetic_Types" rappresentano una generica operazione matematica, essa non possiede alcun campo dati.
La classe è astratta, in quanto prevede i seguenti metodi virtuali puri:
• Un metodo di addizione mediante una ridefinizione di operator+:
Arithmetic_Types*operator+(Arithmetic_Types*)
• Un metodo di sottrazione mediante una ridefinizione di operator-:
Arithmetic_Types*operator-(Arithmetic_Types*)
• Un metodo di moltiplicazione mediante una ridefinizione di operator*:
Arithmetic_Types*operator*(Arithmetic_Types*)
• Un metodo di divisione mediante una ridefinizione di operator/:
Arithmetic_Types*operator/(Arithmetic_Types*)
• Un metodo di Conversione:

double *conversion_in_real*()

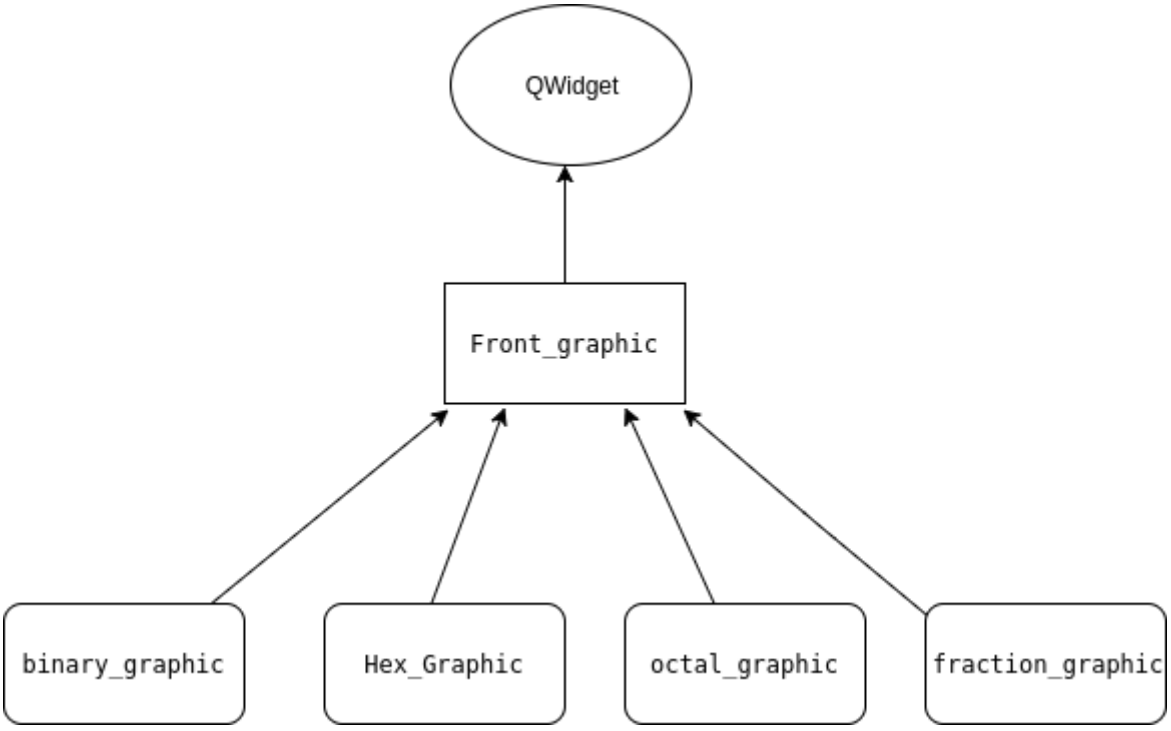
– **Binary**: è una classe concreta , derivata da Arithmetic_Types, i cui oggetti rappresentano un numero intero in forma binaria,
Ogni oggetto di tipo Binary, è caratterizzato da :
• Una lista di char che rappresentano un numero in forma binaria.
La classe implementa i metodi virtuali puri di Arithmetic_Types :
• Binary* operator+(Arithmetic_Types*)
• Binary* operator+(Arithmetic_Types*)
• Binary* operator+(Arithmetic_Types*)
• Binary* operator+(Arithmetic_Types*)
• double *conversion_in_real*()

– **Hex**: è una classe concreta , derivata da Arithmetic_Types, i cui oggetti rappresentano un numero intero in forma esadecimale,
Ogni oggetto di tipo Hex, è caratterizzato da :
• Una lista di char che rappresentano un numero in forma esadecimale.
La classe implementa i metodi virtuali puri di Arithmetic_Types :
• Hex* operator+(Arithmetic_Types*)
• Hex* operator+(Arithmetic_Types*)
• Hex* operator+(Arithmetic_Types*)
• Hex* operator+(Arithmetic_Types*)
• double *conversion_in_real*()

– **Octal**: è una classe concreta , derivata da Arithmetic_Types, i cui oggetti rappresentano un numero intero in forma ottale,
Ogni oggetto di tipo Octal, è caratterizzato da :
• Una lista di char che rappresentano un numero in forma ottale.
La classe implementa i metodi virtuali puri di Arithmetic_Types :
• Binary* operator+(Arithmetic_Types*)
• Binary* operator+(Arithmetic_Types*)
• Binary* operator+(Arithmetic_Types*)
• Binary* operator+(Arithmetic_Types*)
• double *conversion_in_real*()

– **Fraction**: è una classe concreta , derivata da Arithmetic_Types, i cui oggetti rappresentano un numero frazionario
Ogni oggetto di tipo Fraction, è caratterizzato da :
• un numeratore intero
• un denominatore intero
La classe implementa i metodi virtuali puri di Arithmetic_Types :
• Binary* operator+(Arithmetic_Types*)
• Binary* operator+(Arithmetic_Types*)
• Binary* operator+(Arithmetic_Types*)
• Binary* operator+(Arithmetic_Types*)
• double *conversion_in_real*()

View:



Note:
Si è cercato di mantenere (quanto più possibile) separati le librerie della gui dal model, (preferendo contenitori STL) e adottando il modello: MODEL-VIEW-CONTROLLER.
E' presente inoltre una semplice gerarchia di Eccezioni per la corretta gestione degli errori (input non ammessi).

5) Descrizione codice polimorfo:

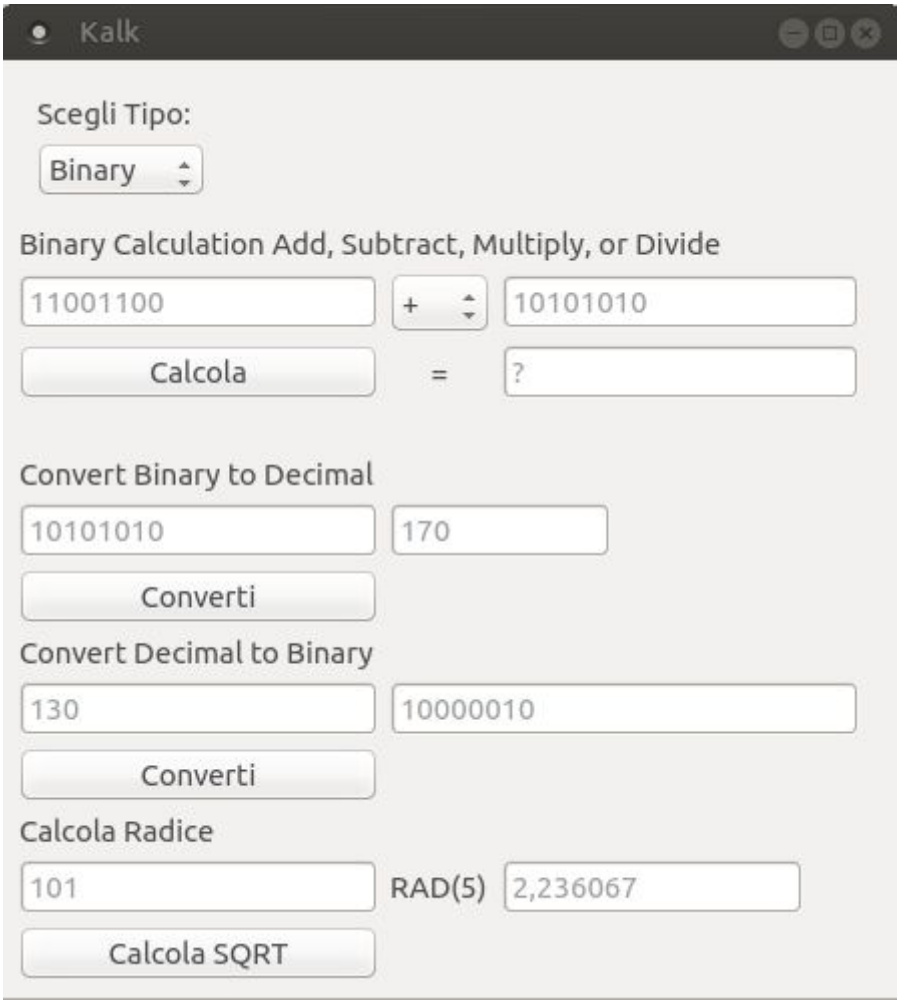
– Nella classe "**binary_controller**", è presente l'utilizzo dei metodi virtuali (operator+, operator-, operator*, operator/) nel metodo "calcolaop1op2" invece nei metodi "calcolaconvBtoD" e "bintodec" è presente l'utilizzo del metodo "*conversion_in_real*": in entrambi i casi, grazie al polimorfismo, e' possibile effettuare la moltiplicazione, addizione, sottrazione, divisione e conversione in reale ricorrendo al late binding. Verrà quindi invocato il metodo corrispondente sulla base del TD di Arithmetic_Types* pa (calcolaop1op2) e Arithmetic_Types* p/binario (calcolaconvBtoD/bintodec).

– Nella classe "**hex_controller**", è presente l'utilizzo dei metodi virtuali (operator+, operator-, operator*, operator/) nel metodo "calcolaop1op2" invece nei metodi "calcolaconvHtoD" e "hextodec" è presente l'utilizzo del metodo "*conversion_in_real*": in entrambi i casi, grazie al polimorfismo, e' possibile effettuare la moltiplicazione, addizione, sottrazione, divisione e conversione in reale ricorrendo al late binding. Verrà quindi invocato il metodo corrispondente sulla base del TD di Arithmetic_Types* pa (calcolaop1op2) e Arithmetic_Types* p/hexadecima (calcolaconvHtoD/hextodec).

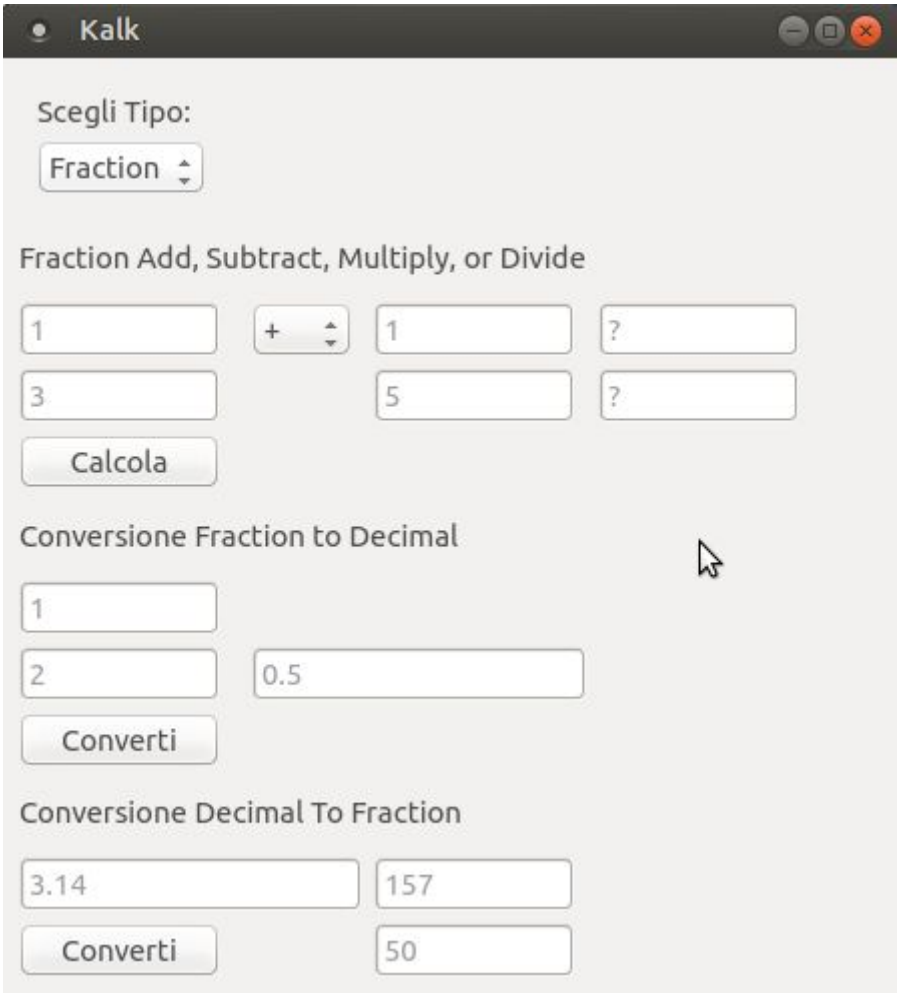
– Nella classe “**octal_controller**”, è presente l'utilizzo dei metodi virtuali (operator+, operator-, operator*, operator/) nel metodo “calcolaop1op2” invece nei metodi “calcolaconvOtoD” e “octtodec” è presente l’utilizzo del metodo “*conversion_in_real*”: in entrambi i casi, grazie al polimorfismo, e' possibile effettuare la moltiplicazione, addizione, sottrazione, divisione e conversione in reale ricorrendo al late binding. Verrà quindi invocato il metodo corrispondente sulla base del TD di Arithmetic_Types* pa (calcolaop1op2) e Arithmetic_Types* p/ottale (calcolaconvOtoD/octtodec).
– Nella classe “**fraction_controller**”, è presente l'utilizzo dei metodi virtuali (operator+, operator-, operator*, operator/) nel metodo “calcolaop1op2” invece nel metodo “calcolaconvFtoD” è presente l’utilizzo del metodo “*conversion_in_real*”: in entrambi i casi, grazie al polimorfismo, e' possibile effettuare la moltiplicazione, addizione, sottrazione, divisione e conversione in reale ricorrendo al late binding. Verrà quindi invocato il metodo corrispondente sulla base del TD di Arithmetic_Types* pa (calcolaop1op2) e Arithmetic_Types* p (calcolaconvFtoD).

6) Manuale utente:

La GUI dell'applicativo è stato pensato per essere il più semplice e user-friendly possibile:
Ecco come si presenta all'avvio:



Sotto la scritta “Scegli Tipo” troviamo un menù a tendina dove è possibile scegliere tra quattro diversi tipi di dato (Binario,Decimale,Ottale,Frazionario) quando si clicca su un tipo la finestra cambia aspetto portando l’utente a visualizzare solo le operazioni consentite su quel tipo di dato ad esempio se scegliamo il tipo “Fraction” ecco come viene modificata e si presenta:



Tutti e quattro i tipi di dato permettono le 4 operazioni elementari (+,-,*,/) e la conversione dal tipo di dato scelto a Decimale e viceversa, i tipi: Binary,Octal,Hex mettono a disposizione il calcolo della radice quadrata (SQRT), inserito uno dei tre tipi esso viene convertito in decimale e di tale numero viene fatta la radice quadrata, tutto viene mostrato all'utente.

E' possibile operare solo su numeri interi, non è possibile inserire numeri Binari/Esadecimali e Ottali in virgola mobile ad esp(101.101 OR ADB.12 OR 183.15673), non è possibile convertire/sommare/moltiplicare/dividere un numero negativo espresso come Binario/Esadecimale/Ottale.

7) Materiale consegnato:

• File sorgente .h e .cpp necessari per la compilazione del progetto, ordinatamente ripartiti in directory MODEL-VIEW-CONTROLLER.
• File .pro
• Kalk-(Model)-java contenente il modello del progetto scritto in Java
• relazione.pdf

7.1) Comandi per la compilazione:

$ qmake

$ make

$ ./Kalk

8) Parte Java

La parte in java è stata sviluppata in maniera molto simile a quella in C++, si è creata una classe astratta “arithmetic_types” che non contiene nessun campo dati, quindi una “Interfaccia”, le funzioni di somma, sottrazione, moltiplicazione e divisione sono stati implementati attraverso dei metodi astratti (ADD,SUB,MUL,DIV), le classi della gerarchia che estendono ed implementano (attraverso un Override) i metodi di “arithmetic_types”, Sono quattro (Binary,Hex,Octal,Fraction), è presente una Classe Use con un main che mostra degli esempi di uso delle rispettive classi.

8.1) Ambiente di sviluppo:

Sistema Operativo: Ubuntu 16.10 64-bit

- Compilatore: javac 1.8.0
- Versione IDE: Netbeans 8.1

8.2) Comandi per la compilazione:

$ javac *.jav

END.