Programmazione ad Oggetti
Relazione progetto Kalk

Daniel Rossi
Matricola 1125444

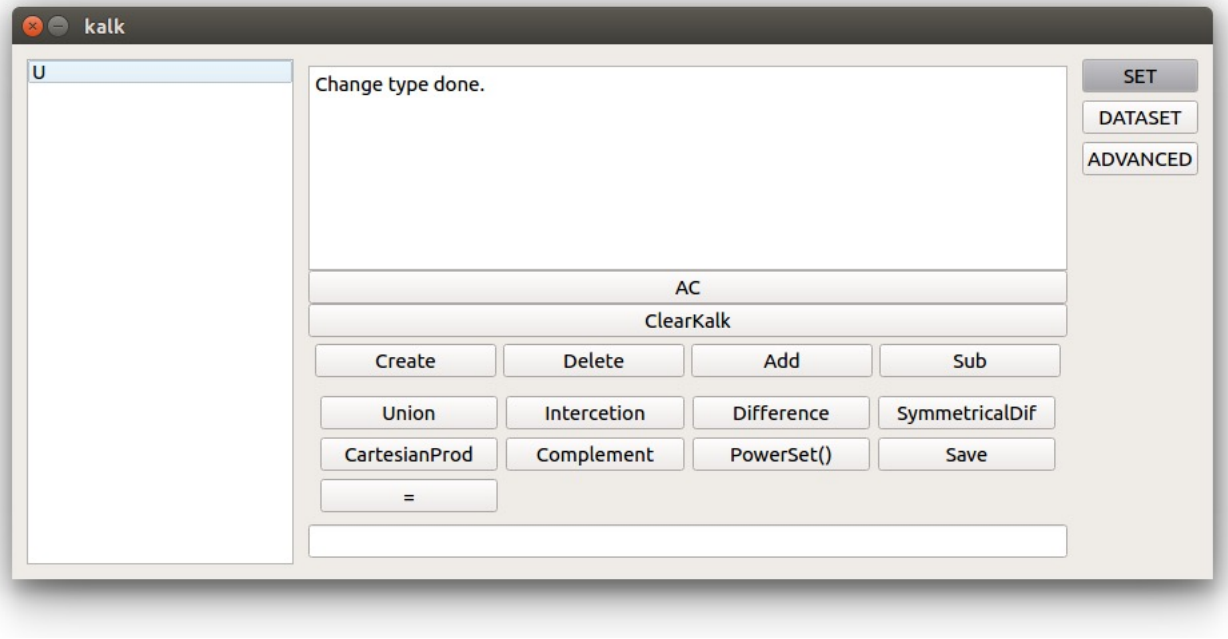

Figura 1: Schermata di default Kalk

Indice

1 Scopo del Progetto

Il progetto si prefiggeva la realizzazione di una calcolatrice che operasse su diversi tipi di dato numerico. I dati disponibili nella calcolatrice sono: gli insiemi numerici interi detti *Set*, questi permettono di eseguire le principali operazioni insiemistiche tra insiemi inseriti in un insieme universo detto *U*, con la possibilità di salvarne il risultato, ogni *Set* conterrà come impone la teoria degli insiemi una sola istanza dello stesso valore; i *Dataset* comuni sono insiemi campionari di numeri interi che posso comparire anche più volte nello stesso *Dataset*, questi rappresentano una sequenza di misurazioni intere e forniscono le principali operazioni statistiche su un determinato insieme campionario, come media varianza e deviazione standard; gli *Advanced* dataset sono dataset con valori compresi nell'intervallo [-60 , +60], questi insiemi campionari sono pensati per essere messi in relazione tra loro per ricavare informazioni sulla correlazione dei due insiemi stessi. La calcolatrice è divisa in tre schede, una per ogni tipo di dato, ci si prefiggeva di avere schede diverse per ogni insieme.

2 Principali modiche seconda consegna

Le modifiche più significative riguardano soprattuto le viste e la logica, mentre la gerarchia dei tipi di dato non è stata modificata se non per minimi cambiamenti. La gestione del cambio di status è stata completamente modificata realizzando questa sorta di parallelismo tra vista e logica. Le nuove classi inserite e degne di nota sono:

- Class AppKalk;
- Class Logic;
- Class View;
- Class Keyboard e derivate;
- Class BasicLogic e derivate;

3 Gerarchie utilizzate

3.1 Gerarchia dei tipi di dato

3.1.1 Class Numbers

La classe astratta Numbers rappresenta un'ipotetica sequenza di numeri interi. Numbers ha un campo dati di tipo stringa che identifica ciascuna sequenza di numeri, definisce inoltre diversi metodi virtuali puri con i seguenti contratti:

- **ris* in_const(const int) const**: ricerca nella lista di interi dell'oggetto *Numbers* un match restituendo un oggetto *Ris**, marcata *const* in quanto è invocata da funzioni *const*;
- **ris* in(const int)**: ricerca nella lista di interi dell'oggetto *Numbers* un match restituendo un oggetto *Ris**;
- **numbers clone() const**: resistuisce una copia dell'oggetto di invocazione;
- **std::string name() const**: restituisce il una stringa contenente il tipo di dato in formato testuale;
- **operator std::string()**: overoloading dell'operatore di conversione da *Numbers* verso *std::string*;
- **void clear()**: svuota la lista di interi;
- **void add_value(const int)**: aggiunge un intero all'oggetto *Numbers*;
- **void sub_value(const int)**: sottrae un intero all'oggetto *Numbers*;
- **void add_list(const std::list<int>&)**: aggiunge un lista di interi all'oggetto *Numbers*;
- **void sub_list(const std::list<int>&)**: sottrae una lista di interi all'oggetto *Numbers*;
- **numbers()**: distruttore virtuale puro della classe *Numbers*.

Inoltre definisce i seguenti metodi:

- **std::string get_name() const**: restituisce il nome dell'oggetto *Numbers*;
- **void change_name(std::string)**: modifica il nome dell'oggetto *Numbers*.

La classe *Numbers* una classe annidata protetta *Ris*, utilizzata per operazioni di ricerca sempre dalle classi derivate.

3.1.2 Class Set

La classe concreta *Set*, derivata dalla classe astratta *Numbers*, rappresenta un insieme numerico di interi univoci dove l'ordine con cui vengono inseriti non ha alcuna importanza. Implementa i metodi virtuali della classe *Numbers*:

- **ris* in_const(const int) const**: ricerca nella lista di interi dell'oggetto *Set* un match restituendo un oggetto *Ris**, marcata *const* in quanto è invocata da funzioni *const*;
- **ris* in(const int)**: ricerca nella lista di interi dell'oggetto *Set* un match restituendo un oggetto *Ris**;
- **set* clone() const**: resistuisce una copia dell'oggetto di invocazione *Set*;
- **std::string name() const**: restituisce la stringa *set*;
- **void clear()**: svuota la lista di interi;
- **void add_value(const int)**: aggiunge un intero all'oggetto *Set* se non è già presente;
- **void sub_value(const int)**: sottrae un intero all'oggetto *Set*;
- **void add_list(const std::list<int>&)**: aggiunge un lista di interi all'oggetto *Set* se non sono già presenti;
- **void sub_list(const std::list<int>&)**: sottrae una lista di interi all'oggetto *Set*;

Dispone dei seguenti metodi propri:

- **bool search(const int n)const**: resistuisce *text*true se l'intero n è presente nel'insieme altrimenti *false*;
- **void add_value_without_control(const int n)**: aggiunge un intero all'oggetto *Set* senza controllare che non sia già presente un duplicato;
- **void add_list_without_control(const std::list<int>&)**: aggiunge un lista di interi all'oggetto *Set* senza verificare che non siano già presenti duplicati;
- **std::string partition()const**: ritorna una stringa con tutti i possibili sottoinsiemi *Set*;
- **std::string combination(int, int, std::vector<int>&)const**: ritorna una stringa con tutti i possibili sottoinsieme contenenti un certo numero di elementi;
- **std::list<int> get_element() const**: ritorna la lista di interi del *Set*;

Esegue l'overloading dei seguenti metodi propri:

- **operator std::string()**: overoloading dell'operatore di conversione da *Set* verso *std::string*;
- **bool operator==(const set&)**: ritorna *true* se il nome dei due *Set* è uguale e se contengono gli stessi interi ordinati non per forza alla stessa maniera altrimenti ritorna *false*;
- **bool operator!=(const set&)**: ritorna *true* se il nome dei due *Set* è diverso o se non contengono gli stessi interi ordinati non per forza alla stessa maniera altrimenti ritorna *false*;
- **set& operator=(const set*)**: ritorna un oggetto *Set* distruggendo eventualmente l'oggetto di invocazione onde evitare garbage;
- **set& operator+(const set&) const**: ritorna un oggetto *Set* unendo le liste di interi dell'oggetto di invocazione edi quello passato per riferimento controllando non vi siano duplicati;
- **set& operator-(const set&) const**: ritorna un oggetto *Set* unendo le liste di interi dell'oggetto di invocazione edi quello passato per riferimento controllando non vi siano duplicati;
- **set& operator/(const set&) const**: ritorna un oggetto *Set* la cui lista di interi è formata dagli interi comuni alla lista di interi dell'oggetto di invocazione e da quella dell'oggetto passato per riferimento;
- **set& operator%(const set&) const**: ritorna un oggetto *Set* la cui lista di interi è formata dagli interi non comuni alla lista di interi dell'oggetto di invocazione e da quella dell'oggetto passato per riferimento;
- **std::string operator*(const set&) const**: ritorna una stringa contentente il prodotto cartesiano tra la lista di interi dell'oggetto di invocazione e quella dell'oggetto passato per riferimento.

3.1.3 Class Dataset

La classe concreta *Dataset*, derivata dalla classe astratta *Numbers*. *Dataset* rappresenta un insieme campionario di misurazioni intere ed è caratterizzato da una sequenza di valori anche ripetuti di cui non ha alcuna importanza l'ordine. Il *Dataset* non può avere meno di due valori nell'insieme campionario, se si crea un *Dataset* vuoto o incompleto a questo verranno aggiunti tanti 0 quanti ne serviranno per ottenere una dimensione minima di 2 valori, stessa cosa accade se si sottrae successivamente elementi. Implementa i metodi virtuali della classe *Numbers*:

- **ris* in_const(const int) const**: ricerca nella lista di interi dell'oggetto *Dataset* il match più a sinistra restituendo un oggetto *Ris**, marcata *const* in quanto è invocata da funzioni *const*;
- **ris* in(const int)**: ricerca nella lista di interi dell'oggetto *Dataset* il match più a sinistra restituendo un oggetto *Ris**;
- **set* clone() const**: resistuisce una copia dell'oggetto di invocazione *Dataset*;
- **std::string name() const**: restituisce la stringa *dataset*;
- **void clear()**: svuota la lista di interi;
- **void add_value(const int)**: aggiunge un intero all'oggetto *Dataset* in coda;
- **void sub_value(const int)**: sottrae un intero all'oggetto *Dataset*;
- **void add_list(const std::list<int>&)**: aggiunge un lista di interi all'oggetto *Dataset* in coda;
- **void sub_list(const std::list<int>&)**: sottrae una lista di interi all'oggetto *Dataset* dalla sinistra;

Dispone dei seguenti metodi propri:

- **std::list<int> get_element() const**: ritorna la lista di interi del *Dataset*;
- **int somme()const**: ritorna la somma degli interi contenuti nella lista dell'oggetto di inviocazione;
- **int size() const**: ritorna il numero di interi contenuti nella lista dell'oggetto di inviocazione;
- **double average()const**: ritorna la media degli interi contenuti nella lista dell'oggetto di inviocazione;
- **int GL()const**: ritorna i gradi di libertà della lista dell'oggetto di inviocazione;
- **double variance()const**: ritorna la varianza degli interi contenuti nella lista dell'oggetto di inviocazione;
- **double DS()const**: ritorna la deviazione standard degli interi contenuti nella lista dell'oggetto di inviocazione;
- **double deviance()const**: ritorna la devianza degli interi contenuti nella lista dell'oggetto di inviocazione;
- **std::list<double> deviation()const**: ritorna una lista contenente la deviazione degli interi contenuti nella lista dell'oggetto di inviocazione;
- **std::list<double> power2_deviation()const**: ritorna una lista contenente la $deviazione^2$ degli interi contenuti nella lista dell'oggetto di inviocazione;
- **void checkpoint(const dataset& s)const**: lancia una eccezione al verificarsi di alcune condizioni;

Esegue l'overloading dei seguenti metodi propri:

- **operator std::string()**: overoloading dell'operatore di conversione da *Dataset* verso *std::string*;
- **bool operator!=(const dataset&)**: ritorna *true* se il nome dei due *Dataset* è diverso o se non contengono gli stessi interi ordinati alla stessa maniera altrimenti ritorna *false*;
- **dataset& operator=(const dataset*)**: ritorna un oggetto *Set* distruggendo eventualmente l'oggetto di invocazione onde evitare garbage;
- **std::list<int> operator*(const dataset&) const**: ritorna una lista di interi contentente il prodotto tra la lista di interi dell'oggetto di invocazione e quella dell'oggetto passato per riferimento.

3.1.4 Class Advanced

La classe concreta *Advanced*, derivata dalla classe concreta *Dataset*, rappresenta un insieme campionario di misurazioni intere comprese nell'intervallo intero [-60;+60]. La classe *Advanced* è caratterizzata da una sequenza di valori interi anche ripetuti; è di fondamentale importanza l'ordine in cui si susseguono i valori, questo perché gli *Advanced* sono fatti per essere messi in relazione tra loro come valori di ascissa e ordinata di un grafico cartesiano. Inoltre dispone di alcuni campi dati di tipo decimale che contengono il risultato delle operazioni di base del *Dataset* come la somma delle misurazioni, la loro media, i gradi di libertà, la varianza ed altre. Implementa i metodi virtuali della classe *Dataset*:

- **advanced* clone() const**: resistuisce una copia dell'oggetto di invocazione *Advanced*;
- **std::string name() const**: restituisce la stringa *advanced*;
- **void clear()**: richiama lo stesso metodo della classe *Dataset* e poi invoca il metodo *void update()*;
- **void add_list(const std::list<int>&)**: richiama lo stesso metodo della classe *Dataset* e poi invoca il metodo *void update()*;
- **void sub_list(const std::list<int>&)**: richiama lo stesso metodo della classe *Dataset* e poi invoca il metodo *void update()*;

Dispone dei seguenti metodi propri:

- **void update()**: aggiorna i campi dati successivamente ad una modifica richiamando alcuni metodi della classe *Dataset*;
- **std::list<double> coscarto(const advanced&)const**: ritorna una lista di interi contente lo scarto tra la lista di interi dell'oggetto di invocazione e l'oggetto passato per riferimento;
- **double codevianza(const advanced&)const**: ritorna la codevianza tra la lista di interi dell'oggetto di invocazione e l'oggetto passato per riferimento;
- **double covarianza(const advanced&)const**: ritorna la covarianza tra la lista di interi dell'oggetto di invocazione e l'oggetto passato per riferimento;
- **double correlazione(const advanced&)const**: ritorna la correlazione tra la lista di interi dell'oggetto di invocazione e l'oggetto passato per riferimento;
- **double coeff_determinazione(const advanced&)const**: ritorna il coefficiente di determinazione tra la lista di interi dell'oggetto di invocazione e l'oggetto passato per riferimento;
- **double regressione(const advanced&)const**: ritorna la regressione tra la lista di interi dell'oggetto di invocazione e l'oggetto passato per riferimento;
- **double intercetta(const advanced&)const**: ritorna l'intersezione tra la lista di interi dell'oggetto di invocazione e l'oggetto passato per riferimento;

3.2 Gerarchia della vista

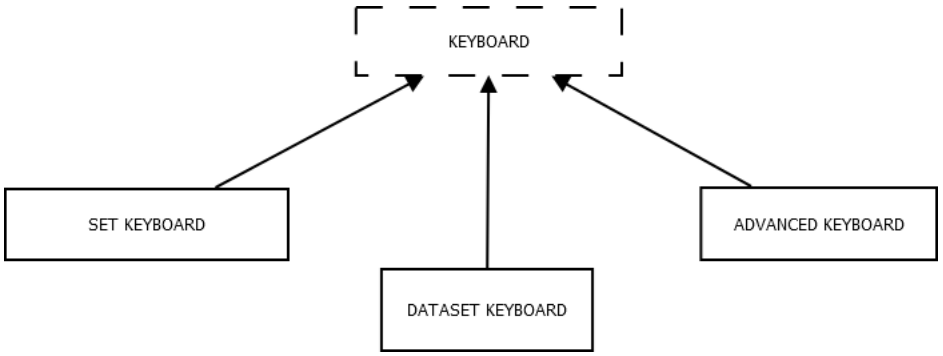

Figura 2: Gerarchia logica

3.2.1 Classe Keyboard

La gerarchia dei tastierini è determinata attraverso la classe astratta *KeyBoard*, che deriva dalla classe *QWidget*. Questa classe contiene i seguenti metodi virtuali puri:

- **std::vector<QString> getSingleOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che utilizzano un solo operando;
- **std::vector<QString> getMultiOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che utilizzano due operandi;
- **std::vector<QString> getExtraOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che utilizzano un solo operando.

Queste rappresentano le tre principali famiglie di operazioni disponibili nella calcolatrice. Dispone inoltre di un metodo proprio:

- **void configure()**: configura il tastierino chiamando dinamicamente i metodi sopra citati e crea ed inserisce i tasti, questi poi vengono predisposti, attraverso delle *connect* e dei *SignalMapper*, per invocare dei segnali qualora cliccati.

Dispone dei seguenti segnali:

- **void multioperation(int)**: segnale che invia l'indice dell'operazione tra due *Numbers*;
- **void singleOperation(int)**: segnale che invia l'indice dell'operazione da seguire su di un*Numbers*;
- **void extraoperation(int)**: segnale che invia l'indice dell'operazione da eseguire;

3.2.2 Classe SetKeyboard

La classe *SetKeyboard* è utilizzata per definire quali operazioni per il tipo di dato *Set* sia possibile eseguire. Vengono definite tre tipi di operazioni principali, i cui nomi vengono definiti all'interno di questi metodi:

- **std::vector<QString> getSingleOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che utilizzano un solo operando, le operazioni disponibili sono:
  1. *Complement*: complementare di un set *Set* rispetto l'insieme universo *U* definito nella parte logica
  2. *PowerSet()*: insieme delle parti di un *Set*
- **std::vector<QString> getMultiOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che utilizzano due operandi, le operazioni disponibili sono:
  1. *Union*: unione tra due *Set*
  2. *Intercetion*: intersezione tra due *Set*
  3. *Difference*: unione tra due *Set*
  4. *SymmetricalDif*: differenza simmetrica tra due *Set*
  5. *CartesianProd*: prodotto cartesiano tra due *Set*

- **std::vector<QString> getExtraOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che non ricadono nelle precedenti due famiglie, sono disponibili le seguenti operazioni:

    1. *Save*: il risultato dell'ultima operazione che ha restituito un oggetto *Set* viene inserito nella lista generale dei *Set* disponibili
    2. =: richiesta di esecuzione di un'operazione multipla

3.2.3 Classe DatasetKeyboard

La classe *DatasetKeyboard* è utilizzata per definire quali operazioni per il tipo di dato *Dataset* sia possibile eseguire. Vengono definite tre tipi di operazioni principali, i cui nomi vengono definiti all'interno di questi metodi:

- **std::vector<QString> getSingleOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che utilizzano un solo operando, le operazioni disponibili sono:

    1. *Somme*: somma degli interi contenuti nel *Dataset*
    2. *Size*: numero degli interi contenuti nel *Dataset*
    3. *Average*: media degli interi contenuti nel *Dataset*
    4. *GL*: gradi di libertà degli interi contenuti nel *Dataset*
    5. *Variance*: varianza degli interi contenuti nel *Dataset*
    6. *DS*: deviazione standard degli interi contenuti nel *Dataset*
    7. *Deviance*: devianza degli interi contenuti nel *Dataset*
    8. *Deviation*: deviazione degli interi contenuti nel *Dataset*
    9. *Deviation*^{2}: *deviazione*^{2} degli interi contenuti nel *Dataset*

- **std::vector<QString> getMultiOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che utilizzano due operandi, al momento nessuna funzione di questo tipo è disponibile;

- **std::vector<QString> getExtraOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che non ricadono nelle precedenti due famiglie, sono disponibili le seguenti operazioni: due operandi, al momento nessuna funzione di questo tipo è disponibile;

3.2.4 Classe AdvancedKeyboard

La classe *AdvancedKeyboard* è utilizzata per definire quali operazioni per il tipo di dato *Advanced* sia possibile eseguire. Vengono definite tre tipi di operazioni principali, i cui nomi vengono definiti all'interno di questi metodi:

- **std::vector<QString> getSingleOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che utilizzano un solo operando, al momento nessuna funzione di questo tipo è disponibile;

- **std::vector<QString> getMultiOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che utilizzano due operandi, le operazioni disponibili sono:

    1. *AllInfo*: resitutisce tutte le informazioni successive in un'unica volta
    2. *Coscarto*: coscarto tra due *Advanced*
    3. *Codevianza*: codevianza tra due *Advanced*
    4. *Covarianza*: covarianza tra due *Advanced*
    5. *Correlazione*: collerazione tra due *Advanced*
    6. *Coef. Determ.*: coefficiente di determinazione tra due *Advanced*
    7. *Regressione*: coefficiente angolare della retta risultante tra i due *Advanced*
    8. *Intercetta*: punto di intersezione con l'asse delle y della retta risultante tra i due *Advanced*
    9. *Retta*: retta risultante tra i due *Advanced*
    10. *Grafic*: grafico della retta risultante tra i due *Advanced*, con punti annessi.

- **std::vector<QString> getExtraOperationkeyboard()const**: ritorna un vettore contenente i nomi delle operazioni che non ricadono nelle precedenti due famiglie, sono disponibili le seguenti operazioni:

    1. =: richiesta di esecuzione di un'operazione multipla

3.3 Gerarchia della logica

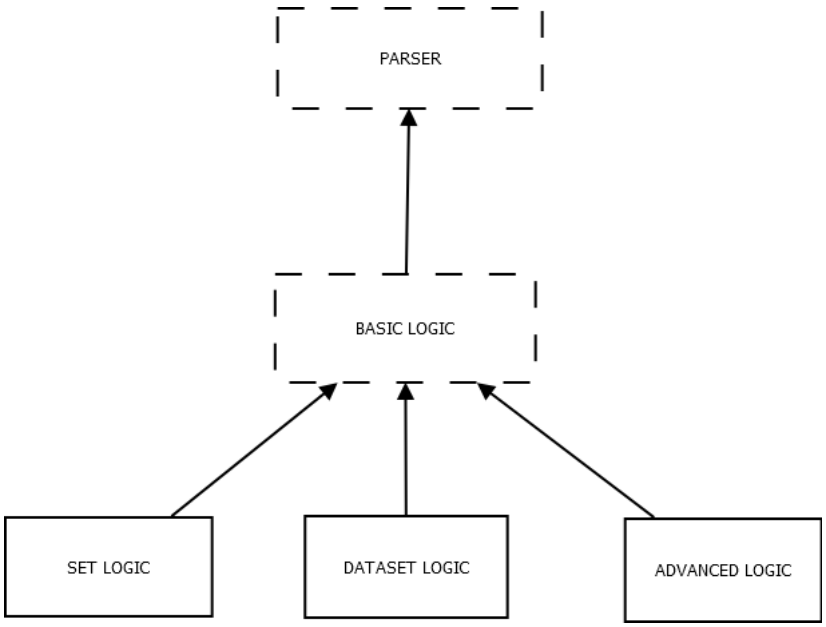

Figura 3: Gerarchia logica

La gerarchia della logica è formata da una classe base atratta *Parser*, da cui deriva una classe base atratta *BasicLogic* da cui derivano tutte le classi concrete della logica come *SetLogic*, *DatasetLogic* e *AdvancedLogic*.

3.3.1 Classe Parser

La classe base astratta *Parser* permette di eseguire un *parsing* delle stringhe prese in input, definisce un metodo virtuale puro con il seguente contratto:

- **bool condition()const**: ritorna *true* se le condizioni di *parsing* sono state rispettate altrimenti ritorna *false*.
- **QString getNameType()**: ritorna il tipo dinamico dell'oggetto d'invocazione come una stringa;

Inoltre definisce i seguenti metodi:

- **std::list<int> parserData(QString)**: prende in input una *QString* e se il *parsing* va a buon fine restituisce una *std::list<int>* contenente gli interi individuati dal parser;
- **std::string parserName(QString)**: prende in input una *QString* e se è corrisponde al carattere "" gli riassegna il nome nome del tipo di dato dinamico e un intero positivo progressivo;
- **void restoreDefault()**: reimposta a valori di default i campi dati privati della classe.

3.3.2 Classe BasicLogic

La classe *BasicLogic* permette di eseguire le principali operazioni di base della calcolatrice, ha un campo dati di tipo *std::list<Numbers*>** al cui interno sono presenti oggetti di tipo statico *Numbers* e tipo dinamico uno qualsiasi della gerarchia. Definisce inoltre diversi metodi virtuali puri con i seguenti contratti:

- **numbers* getObjectLogicClass(std::string,std::list<int>)**: ritorna un oggetto di tipo statico *Numbers* costruito con suddetti parametri.

Inoltre definisce i seguenti metodi virtuali:

- **void clearKalkElements()**: elimina tutte le occorrenze nella lista generale di oggetti *Numbers* con *name() == getNameType()*.
- **void SetOperand(std::string, std::string)**: inserisce il nome dell'operando rispettivamente o nel primo o nel secondo operando;
- **void multioperation(int)**: fissa quale operazione multi insieme ci si aspetta e si attende il secondo operando;

- **void selectOperand(QListWidgetItem*)**: selezionea il nome dell'operando e si richiama il metodo *SetOperand(std::string,std::string)*;
- **void singleOperation(int)**: fissa quale operazione multi insieme ci si aspetta ed esegue l'operazione;
- **void add_set(QString,QString)**: crea e successivamente aggiunge nella lista generale di oggetti *Numbers* l'oggetto restituito dal metodo *getObjectLogicClass(...,...)* dopo aver parsato le QStirng in input;
- **void sub_set(QString)**: sottrae dalla lista generale di oggetti *Numbers* quello con nome corrispondete al parametro QString e con tipo dinamico corrispondente a *getNameType()*, altrimenti lancia un'eccezione;
- **void add_elements(QString,QString)**: aggiunge gli interi dati dalla lista *parserData(...)* all'oggetto *Numbers* con nome corrispondente al primo parametro e stesso *getNameType()* altrimenti lancia un'eccezione;
- **void add_set(QString,QString)**: sottrae gli interi dati dalla lista *parserData(...)* all'oggetto *Numbers* con nome corrispondente al primo parametro e stesso *getNameType()* altrimenti lancia un'eccezione;
- **void results()**: se non è disponibile nessun risultato lancia un'eccezione altrimenti lo inserisce nella lista generale di *Numbers*.

Sono disponibili i seguenti metodi:

- **QString getNameType()**: restituisce il campo dati *nameType* che rappresenta il tipo di dato dinamico a cui è inizializzata la *BasicLogic*;
- **bool checkType(std::string)const**: restituisce *true* se i tipi di dato dinamici sono uguali *false* altrimenti;
- **void AC()**: resetta gli operandi, l'operazione e il risultato corrente;
- **void getElements()**: emette un segnale con cui invia una *std::list<QString>* con i nomi dei tipi di dato per cui checkType(...) ritorna *true*;
- **void update()**: metodo richiamato quando avviene un cambio di tipo di dato, aggiorna la lista di oggetti disponibili, chiude eventuali finestre di input aperte;

Dispone dei seguenti segnali:

- **listOfElements(std::list<QString>)**: lista di oggetti con tipo dinamico uguale a quello dato da *getNameType()*;
- **void setBarra(QString)**: setta la barra principale con il messaggio passato come parametro;
- **void setError(QString)**: setta la barra degli errori con il messaggio passato come parametro;
- **void setErrorInput(QString)**: invia un segnale errore alla gestione dell'input;
- **void setErrorInput(QString)**: invia un segnale di errore alla gestione dell'input;
- **void closeInputWindow()**: invia un segnale con cui si autorizza a chiudere la schermata di input corrente;

3.3.3 Classe SetLogic

La classe *SetLogic* permette di eseguire le operazioni riguardanti il tipo di dato *Set*, dispone dei seguenti metodi:

- **void singleOperation(int)**: esegue l'operazione data dal parametro int;
- **void selectOperand(QListWidgetItem*)**: mostra il parametro selezionato e richiama setOperand() se il nome del *Set* richiesto non è "U";
- **void add_set(QString,QString)**: crea un nuovo oggetto se non ne esiste già uno con nome uguale e aggiunge la sua lista di interi al *Set* "U";
- **void sub_set(QString)**: Elimina l'oggetto *Set* dalla lista di oggetti e sottrae il contenuto da "U" se non è contenuto in nessun'altro *Set*, verifica ciò grazie alla funzione *bool in(const int,std::string)const*;
- **void add_elements(QString,QString)**: aggiugne la lista di interi al *Set* con nome ricercato e al *Set* "U";
- **void sub_elements(QString,QString)**: sottrae la lista di interi al *Set* con nome ricercato e al *Set* "U";
- **bool condition()const;**: condizione con cui si personalizza il parsing;
- **bool in(const int,std::string)const**: ritorna *true* se l'intero non è presente negli altri insiemi altrimenti *false*;

- **set* getObjectLogicClass(std::string,std::list<int>)**: ritorna un *Set* costruito con i parametri;
- **void results()**: esegue l'operazione data dal parametro dagli operandi e dall'operazione precedentemente fissati;
- **void clearKalkElements()**: elimina ogni occorrenza di oggetti di tipo *Set* dalla lista generale di tipo *Numbers*;
- **void extraoperation(int)**: esegue l'operazione data dal parametro int;

Dispone dei seguenti segnali:

- **listOfElements(std::list<QString>)**: lista di oggetti con tipo dinamico uguale a quello dato da *getNameType()*;
- **void setBarra(QString)**: setta la barra principale con il messaggio passato come parametro;
- **void setError(QString)**: setta la barra degli errori con il messaggio passato come parametro;
- **void setErrorInput(QString)**: invia un segnale errore alla gestione dell'input;
- **void setErrorInput(QString)**: invia un segnale di errore alla gestione dell'input;
- **void closeInputWindow()**: invia un segnale con cui si autorizza a chiudere la schermata di input corrente;

3.3.4 Classe DatasetLogic

La classe *DatasetLogic* permette di eseguire le operazioni riguardanti il tipo di dato *Dataset*, dispone dei seguenti metodi:

- **void singleOperation(int)**: esegue l'operazione data dal parametro int;
- **bool condition()const;**: condizione con cui si personalizza il parsing;
- **dataset* getObjectLogicClass(std::string,std::list<int>)**: ritorna un *Dataset* costruito con i parametri;

3.3.5 Classe AdvancedLogic

La classe *AdvancedLogic* permette di eseguire le operazioni riguardanti il tipo di dato *Advanced*, dispone dei seguenti metodi:

- **void singleOperation(int)**: esegue l'operazione data dal parametro int;
- **bool condition()const;**: condizione con cui si personalizza il parsing;
- **advanced* getObjectLogicClass(std::string,std::list<int>)**: ritorna un *Advanced* costruito con i parametri;

4 Classi contenitore AppKalk

La classe *AppKalk* è la classe contenitore generale che permette contiene al suo interno 3 oggetti di tipo:

- **Logic**: gestore della logica;
- **Input**: gestore dell'input;
- **View**: gestore della vista;

4.1 Classe Logic

La classe *Logic* permette lo scambio del tipo di dato tra *Set*,*Dataset* e *Advanced* a livello logico e gestisce tutte le operazioni dei diversi tipi di dato, deriva dal tipo di dato *QObject*. Possiede un campo dati di tipo *std::vector<BasicLogic*>* detta *Logics* per lo store delle unità logiche. Definisce poi un *std::list<const numbers*>** detto *Uset* che viene passato per riferimento alle singole unità logiche che opereranno con condivisione di memoria, sarà poi compito della classe *Logic* distruggere suddetta lista. Definisce un campo *BasicLogic** detto *uni* rappresentante un riferimento all'unità logica corrente. Definisce i seguenti metodi SLOT:

- **void AC()**: invoca su *uni* il metodo *void AC()*;
- **void multioperation(int)**: invoca su *uni* il metodo *void multioperation(int)*;
- **void singleOperation(int)**: invoca su *uni* il metodo *void AC()*;

- **void selectOperand(QListWidgetItem*)**: invoca su *uni* il metodo *void singleOperation(int)*;
- **void changeLogic(int)**: assegna a *uni* la logica nella posizione data dal parametro dentro *logics*;
- **void executeOperation(int,QString,QString)**: esegue un'operazione di input invocando su *uni* i metodi *add_set(QString,Qstring)*, *sub_set(QString)*, *add_elements(QString,Qstring)*, *sub_elements(QString,Qstring)*;
- **void clearKalkElements()**: invoca su *uni* il metodo *void changeLogic(int)*;
- **void extraoperation(int)**: invoca su *uni* il metodo *void extraoperation(int)*;

Definisce anche i seguenti metodi:

- **void setLogics()**: popola *logics* inserendo le unità logiche;
- **std::vector<QString> nameType()const**: restituisce un vettore contenente *QString* con tutti i nomi dei tipi di dato;

4.2 Classe View

La classe *View* permette lo scambio del tipo di dato tra *Set*,*Dataset* e *Advanced* a livello di vista, deriva dal tipo di dato *QWidget*. Possiede una serie di campi da dati che fanno a formare la GUI, di seguito la lista:

- **int currentStatus**: indice della vista corrente;
- **QPalette* pal**: QPalette per colorare il QPushButton corrispondente al tipo di dato che si sta utilizzando;
- **QTextEdit* Barra**: sulla Barra verranno stampati tutti i messaggi che non siano errori, è situata nella parte alta;
- **QLineEdit* errori**: sulla barra degli errori verranno stampati tutti gli errori dovuti a tutte le operazioni tranne quelli dovuti ad errate operazioni di input;
- **QListWidget* elenco**: elenco su cui verrano mostrati gli oggetti *Set*,*Dataset* e *Advanced* disponibili;
- **QHBoxLayout* kalk**: kalk è il contenitore di tutti i widget;
- **std::vector<QPushButton*> statusButton**: :contenitore dei pulsanti della barra dei pulsanti di status;
- **std::vector<Keyboard*> views**: vettore con le i tastierini dei diversi tipi di dato.

Definisce i seguenti metodi SLOT:

- **void refresh(std::list<QString>)**: aggiorna l'Elenco degli oggetti disponibili;
- **void setAC()**: All Clear invia un segnale con cui resetta gli operandi, le operazioni e ripulisce la Barra;
- **void setBarra(QString)**: setta la Barra con il parametro *QString*;
- **void setError(QString)**: setta la barra degli errori con il parametro *QString*;
- **void changeLogic(int)**: emette un segnale con cui cambia l'unità logica e cambia anche il tastierino;
- **void changePalette(int)**: resetta il pulsante con la QPalette standard e setta con Pal il pulsante corrispondente al nuovo tipo corrente;
- **void clear()**: ripulsice Barra e barra degli errori;

Per il metodo *changePalette()* sono presenti tre versioni che producono risultati estetici simili, ossia quello di rendere il pulsante dello status corente di un colore diverso rispetto agli altri, tutte e tre risultano funzionanti nella mia macchina, mentre nei laboratori solo quella decommentata risulta funzionare correttamente. Definisce anche i seguenti metodi:

- **void setViews()**: popola *logics* inserendo i tastierini;
- **std::vector<QString> getBasicOperation()const**: restituisce un vettore contenente *QString* le etichette per i pulsanti di operazione di input;

Definisce anche i seguenti segnali:

- **void input(int)**: segnale che trasmette la necessità di aprire un pannello di input e inizializarlo col parametro passato;
- **void selectOperand(QListWidgetItem*)**: richiede la selezione del parametro come operando;

- **void changelogic(int)**: segnale per cambiare l'unità logica;
- **void cancel()**: segnale per cancellare operandi e l'operazione;
- **void multioperation(int)**: segnale trasmette indice per operazione multi insieme;
- **void singleOperation(int)**: segnale trasmette indice per operazione multi insieme;
- **void clearKalkElements()**: richiede di eliminare tutti gli oggeti con tipo uguale a quello corrente;
- **void CloseIfExist()**: emette un segnale con cui si chiede di chiudere eventuali pannelli di input ancora aperti;
- **void extraoperation(int)**: segnale trasmette indice per operazione extra insieme;

4.3 Classe Input

La classe *Input* permette lo di eseguire l'input delle informazioni con cui costruire gli oggetti utilizzati poi dalla calcolatrice. Possiede una serie di campi da dati che fanno a formare il pannello di input, di seguito la lista:

- **int operation**: indice dell'operazione di input;
- **bool isUsed**: *true* allora esiste già un pannello di input, altrimenti *false*;
- **extrapanel* input_view**: puntatore ad un pannello di input;
- **QMessageBox* MessageError**: pannello con messaggio d'errore per input che generino un errore;

Definisce i seguenti metodi SLOT:

- **void manageInput(int)**: imposta il pannello secondo l'operazione richiesta;
- **void unlock()**: sblocca il pannello di input;
- **void SendDataInput(QString,QString)**: invia gli input ricevuti;
- **void setErrorInput(QString)**: crea una QMessageBox ad esplicitare l'errore incorso;
- **void ifExistClose()**: se esiste chiude il pannello di input corrente;

Definisce anche i seguenti metodi:

- **void configureExtrapanel()**: permette di configurare il tastierino secondo l'operazione richiesta; .

Definisce anche i seguenti segnali:

- **void executeOperation(int,QString,QString)**: segnale che trasmette verso la logica l'operazione di input eseguita e i dati ottenuti.

5 Descrizione del codice polimorfo

Nelle diverse classi precedentemente descritte sono presenti non pochi metodi che eseguono chiamate polimorfe. Per fare esempi di chiamate polimorfe ogni chiamata fatta dalla classe *Logic* sul campo dati *uni* è polimorfa, il tipo statico del puntatore è di tipo *BasicLogic* mentre l'oggetto puntato potrebbe essere di tipo *SetLogic*,*DatasetLogic* o *AdvancedLogic*. Altri esempi di chiamate polimorfe possono essere individuati nella classe *BasicLogic* nelle operazioni di input, dopo vengono chiamati i metodi *add_list* ed altri.

6 Estensibilità

Questa sezione si propone di essere una guida all'estensione della calcolatrice con ulteriori tipi di dato.

6.1 Prerequisiti

Come prerequisito fondamentale diciamo che il nuovo tipo di dato deve estendere la classe *Numbers*, implementando i suoi metodi virtuali puri. Il nuovo tipo di dato può anche estendere una delle classi già disponibili eseguendo l'override dei metodi disponibili. Il nuovo tipo di dato potrà essere una sequenza di numeri interi.

6.2 View

La view del nuovo tipo di dato consisterà nel creare una classe derivata dalla classe *Keyboard* implementando i tre metodi virtuali puri:

- std::vector<QString> getSingleOperationkeyboard()const: inserire qui i nomi delle operazioni singole;
- std::vector<QString> getSingleOperationkeyboard()const: inserire qui i nomi delle operazioni multiple;
- std::vector<QString> getSingleOperationkeyboard()const: inserire qui i nomi delle operazioni extra.

Per ciascuno di questi metodi si inizializzerà un vettore predisposto a contenente QString, si procederà a *pushare* al suo interno i nomi delle operazioni, l'ordine con cui verranno inserite sarà l'ordine in cui verranno richiamate nella parte logica. Da questo momento tutti i pulsanti sono già connessi nella parte logica.
Ora spostiamoci nella classe *View*, nel metodo *void setViews()*, pushate in coda un oggetto allocato nello *heap* del nuovo tastierino.

6.3 Logica

Per indrodurre il nuovo tipo di dato nella calcolatrice dovrà essere creata una classe derivata dalla classe *BasicLogic*, che dovrà estendere almeno i metodi virtuali puri:

- **numbers* getObjectLogicClass(QString,QString)**: ritornare un dato costruito con i parametri del tipo di dato che si gestisce;
- **QString getNameType()**: il nome del tipo di dato che si gestisce;
- **bool condition()const**: condizioni personalizzazione del parsing;

Se si volesse dare un'implementazione alle operazioni multiple, singole o extra create nel tastierino, si dovrà procede all'override dei seguenti metodi:

- **void singleOperation(int index)**: operazione per i tipi di dato singoli;
- **void results()**: operazione per i tipi di dato multipli;
- **void extraOperation(int index)**: operazione per i tipi di dato singoli, vanno inserite le operazioni in ordine

Le operazioni vanno inserite nello stesso ordine con cui sono stati inseriti i pulsanti nel tastierino. Un istanza di un oggetto allocato nello *heap* dovrà essere inserito in coda alla fine del metodo *void setLogics()* della classe *Logic*

6.4 Simmetria LogicView

Risulta essere di incredibile importanza la simmetria tra vista e logica, infatti affinchè le chiamate risultino sensate la vista deve operare con il tipo di dato corretto, per questo motivo i due nuovi oggetti dovranno essere inseriti in posizioni corrispondenti.

7 Manuale utente

7.1 Avvio

All'avvio dell'applicazione Kalk questa si troverà nella scheda *Set*, per gli altri tipi di dato basterà premere i tasti in alto a destra cambiando così tipo di dato su cui operare.

7.2 Struttura

La calcolatrice è divisa in tre aree principali: nell'area sulla sinistra è presente una lista che in base al tipo di dato su cui si sta lavorando mostra tutti gli oggetti creati fino a quel momento per il tipo di dato corrente, la seconda parte è a sua volta divisa in altre quattro parti: nella parte superiore è presente un campo di output su cui vengono visualizzati gli insiemi selezionati e i risultati delle operazioni, nella parte immediatamente inferiore sono presenti due pulsanti, uno etichettato *ClearKalk* per il reset della calcolatrice per quel tipo di dato, uno etichettato *AC* per desettare gli operandi e l'operazione selezionata, nella parte centrale è presente un tastierino con quattro tasti con le operazioni di input, immediatamente sotto sono presenti una serie di pulsanti sui quali viene riportata una etichetta con l'operazione corrispondente.
Infine nella parte inferiore è presente una barra di output dove verranno mostrati eventuali errori dovuti a operazioni non disponibili o a irregolarità nella costruzione di un insieme numerico.

7.3 Tipo dato corrente

Il tipo di dato corrente della calcolatrice è indicato in alto a destra dal pulsante di colore più scuro. Nella terza ed ultima parte, a destra della calcolatrice, sono presenti tre pulsanti con i quali è possibile cambiare lo status, premendo uno dei pulsanti la scheda verrà aggiornata al tipo selezionato.

7.4 Dinamica operazioni

La dinamica di selezione di operandi e operazioni è la stessa che in una normale calcolatrice:

- per effettuare operazioni tra due insiemi, si seleziona il primo insieme e successivamente l'operazione. Da questo momento non sarà più possibile cambiare il primo operando se non premendo il pulsante AC, che resetterà l'operazione corrente. Scelta l'operazione si procede scegliendo il secondo operando e premendo il pulsante "=", verrà quindi stampato a schermo il risultato indicando anche l'operazione corrispondente. Nel caso di errori o irregolarità verrà riportato l'errore corrispondente nella barra inferiore;
- Per operazioni di calcolo su singolo insieme, basta selezionare l'insieme su cui si vuole fare l'operazione e subito dopo l'operazione corrispondente; restituirà il risultato nella barra superiore;
- Per operazioni di manipolazione di insiemi: creazione o eliminazione di insiemi, modifica dei valori di un insieme, selezionare il pulsante corrispondente e seguire la procedura indicata nella finestra che comparirà.

La scheda Set rende disponibile fin da subito un set chiamato U, questo è l'insieme universo dove sarà possibile vedere tutti gli elementi inseriti in ogni Set della scheda. Non è possibile eseguire operazioni su questo insieme se non quelle di inserimento o rimozione di valori, se si inserisce un valore questo resterà "spaiato", se si elimina un valore dall'universo verrà rimosso da tutti i Set in cui compariva. Set rende disponibile anche una operazione di salvataggio del risultato corrente, la quale salva il risultato nella barra sulla sinistra e la rende disponibile per ulteriori operazioni.

8 Ore utilizzate

Sono state utilizzate le seguenti ore alla consegna:

- **Progettazione**: 2 ore;
- **Scrittura codice gerarchia tipi**: 0.30 ore;
- **Scrittura codice logica di controllo**: 10 ore;
- **Scrittura codice parte grafica**: 10 ore;
- **Scrittura relazione**: 4 ore;
- **Test**: 2 ore;
- **Traduzione gerarchia tipi in Java**: 0.30 ore.

9 Ambiente di sviluppo

- Sistema operativo: Ubuntu 16.04.3 LTS
- Compilatore: gcc version 5.4.0 20160609
- QT: 4.8.7
- QMake version 2.01a