Università degli Studi di Padova

Relazione per la calcolatrice

KalkRpg

Per giochi di ruolo di fantasia

Realizzata da:

Mirko Gibin, matricola 1102450

In collaborazione con

Giovanni Cavallin, matricola 1148957

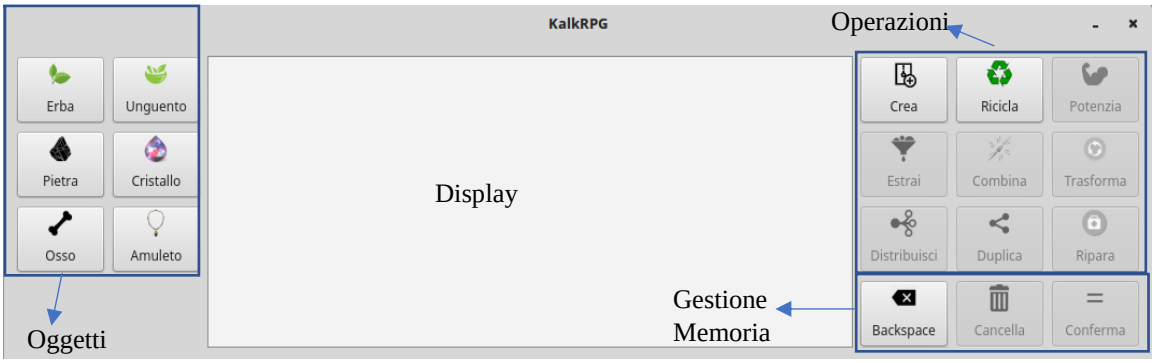
Presentazione

La calcolatrice KalkRpg

KalkRpg è una calcolatrice che simula delle operazioni tra oggetti di un gioco di ruolo di fantasia.

L'interazione con l'utente è resa possibile grazie a:

- pulsanti che permettono di selezionare operazioni e creare oggetti;
- abilitazione e disabilitazione di questi pulsanti, in base alle operazioni realmente possibili dati determinati oggetti;
- procedure guidate che permettono all'utente di impostare i parametri e i valori desiderati;
- un display che indica gli oggetti, le operazioni selezionate e il risultato ottenuto.



Installazione

qmake untitled.pro
make
./KalkRpg

Guida all'uso

La calcolatrice offre sei tipi di oggetto su cui effettuare nove operazioni.

Oggetti

Tutti gli oggetti disponibili presentano una struttura simile, composta da *livello*, *rarità*, e una lista di parametri o statistiche, tra i quali lo *spirito* è l'unico che appartiene a tutti gli oggetti.

Si possono identificare tre tipi di oggetto base (Erba, Pietra, Osso) e tre tipi di oggetto da questi derivati (Unguento, Cristallo, Amuleto):

Oggetti base	**Oggetti derivati**
Erba : *Livello, Rarità, Spirito, Vitalità*	Unguento: *Livello, Rarità, Spirito, Vitalità, Energia*
Pietra: *Livello, Rarità, Spirito, Durezza*	Cristallo: *Livello, Rarità, Spirito, Durezza, Magia*
Osso: *Livello, Rarità, Spirito, Attacco, Difesa*	Amuleto: *Livello, Rarità, Spirito, Attacco, Difesa, Fortuna*

Come si può vedere dalla tabella, gli oggetti derivati ereditano i parametri dagli oggetti base e ne aggiungono.

Queste sono le regole del gioco che abbiamo stabilito:

- Tutti gli oggetti possono avere *livello* e *rarità* compresi tra 1 e 10 rispettivamente;
- Ogni statistica, alla creazione, deve avere un valore compreso tra 1 e 150. Tramite alcune operazioni il livello delle statistiche può però superare questo limite.
- La somma di tutti i valori delle statistiche, livello e rarità esclusi, non può essere superiore a questa formula: *150 * livello * (numero statistiche oggetto).*
  Qualora questo limite venga superato, i valori delle statistiche vengono normalizzati in modo proporzionale.
- Il mana è l'energia necessaria per creare o potenziare un'oggetto e dipende da *livello*, *rarità*, e statistiche di quest'ultimo.

Operazioni

Sono disponibili 9 operazioni unarie e binarie:

Operazione	**Tipo**	**Descrizione**	**Particolarità**
Crea	Unario	Richiede un oggetto, del mana da dedicare alla sua creazione, e un eventuale parametro. L'oggetto verrà inizializzato utilizzando il mana indicato in base al livello e rarità desiderati; se viene indicato un parametro, a questo verrà dedicato metà del mana.	È necessario cliccare sul pulsante Crea, e successivamente indicare l'oggetto desiderato, impostando i parametri richiesti. Può essere usata per tutti gli oggetti.
Ricicla	Unario	Richiede un oggetto. Restituisce il mana che si ottiene dal suo riciclo.	È necessario cliccare sul pulsante Ricicla, e successivamente sull'oggetto desiderato, impostando i parametri richiesti. Può essere usata su tutti gli oggetti.
Potenzia	Unaria	Richiede un oggetto, che dovrà essere inizializzato. Richiede del mana che verrà utilizzato per incrementare le statistiche dell'oggetto. È possibile selezionare anche un parametro a cui dedicare tutto il potenziamento.	È necessario cliccare sull'oggetto desiderato, impostare i suoi valori, e cliccare sul pulsante dell'operazione. Può essere usata su tutti gli oggetti, il suo comportamento cambierà in base all'oggetto su cui verrà effettuata.
Estrai	Binaria	Richiede un oggetto iniziale da cui estrarre, e l'oggetto che si desidera. Estrae dall'oggetto iniziale, se possibile, l'oggetto indicato.	Cliccare sull'oggetto iniziale, poi sull'operazione, e in seguito sull'oggetto desiderato. Può essere invocata su tutti gli oggetti, ma andrà a buon fine solo se si tenta di estrarre da un oggetto base il suo corrispettivo oggetto derivato. In caso di insuccesso viene visualizzato un messaggio esplicativo di errore.
Combina	Binaria	Richiede un'oggetto iniziale, le cui statistiche verranno incrementate grazie a quelle del secondo oggetto che verrà selezionato.	Cliccare sull'oggetto iniziale, impostandone i parametri, poi sull'operazione, e in seguito sul secondo oggetto. L'operazione modifica le statistiche del primo. Può essere usata su tutti gli oggetti.

Trasforma	Binaria	Richiede un'oggetto iniziale, che verrà trasformato nel secondo scelto, se possibile. I parametri del secondo oggetto dipenderanno da quelli del primo.	Cliccare sull'oggetto che si desidera trasformare, impostandone i parametri, poi sull'operazione e infine sull'oggetto desiderato. La trasformazione non è possibile se l'oggetto di destinazione ha più parametri dell'oggetto iniziale. In questo caso viene visualizzato un messaggio di errore.
Distribuisci	Binaria	Può essere invocata unicamente a partire da un cristallo. Il secondo oggetto verrà potenziato in base alla magia e alla durezza del cristallo.	Cliccare sul cristallo, impostarne i valori, cliccare sull'operazione e in seguito sull'oggetto che si desidera potenziare. Il potenziamento provocherà effetti sull'oggetto target solo se la durezza del cristallo sarà sufficiente.
Duplica	Binaria	Invocabile solo su amuleto. Permette di duplicare un qualsiasi oggetto target. Tale oggetto avrà statistiche minori o uguali all'oggetto originale, in base alla rarità dell'amuleto.	Cliccare su amuleto, impostarne i parametri, cliccare sull'operazione e infine sull'oggetto desiderato.
Ripara	Binaria	Invocabile solo su unguento. In base alla sua rarità, questa operazione permette di impostare la statistica più bassa, o tutte le statistiche, dell'oggetto target al valore massimo presente tra le sue statistiche.	Cliccare su unguento, impostarne i valori, cliccare sull'operazione e infine sull'oggetto indicato. Se l'energia sarà sufficiente, l'operazione modificherà i parametri dell'oggetto target.

Pulsanti di gestione

Tutte le operazioni devono essere confermate tramite l'apposito pulsante *Conferma*. Prima di confermare un'operazione è possibile, tramite il pulsante *Backspace*, annullare l'ultima selezione. Una volta confermata l'operazione, verrà visualizzato a display il risultato. E' sempre possibile annullare tutto e pulire il display tramite il pulsante *Cancella.*

Progettazione

Nel progettare la calcolatrice si è seguito il pattern Model-View-Controller, discusso nelle prossime sezioni.

Suddivisione del lavoro

La fase iniziale del lavoro ha compreso la progettazione della gerarchia delle classi, la definizione della sua struttura e l'idealizzazione delle operazioni disponibili. E' stata la parte che ha coinvolto attivamente entrambi i membri. Una volta stabilite le linee guida da seguire, i compiti sono stati divisi nel seguente modo:

- Mirko: implementazione gerarchia, implementazione metodi e grafica della calcolatrice;
- Giovanni: logica d'uso della calcolatrice, relazione e collegamento con la gerarchia e i suoi metodi.

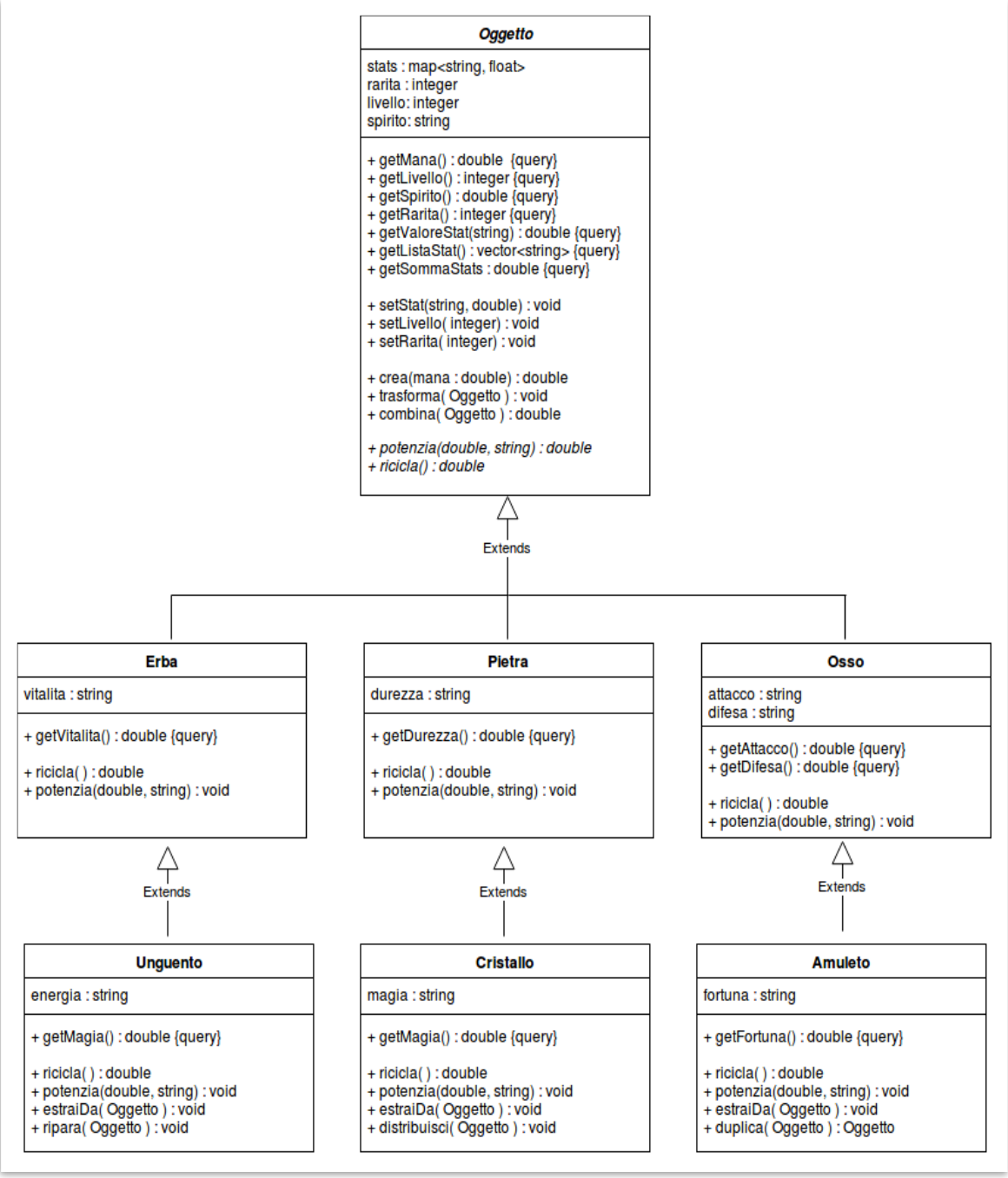
Tale divisione ha permesso di avanzare in modo parallelo e senza significative attese. Ciascun membro ha comunque contribuito, con interventi, consigli e osservazioni, ai ruoli dell'altra.

Model

La parte di Model comprende la gerarchia delle classi e la sua installazione nella classe Model.

Gerarchia:

L'immagine seguente rappresenta la gerarchia delle classi utilizzata per KalkRpg.



- Oggetto è una classe astratta da cui derivano tutte le altre classi. I suoi campi privati sono:
    - Due interi, che contraddistinguono il *Livello* e la *Rarità;*
    - Una stringa contenente il nome del suo parametro caratterizzante: *Spirito;*

- Una *Map<string, double>* contenente come *key* il nome dei parametri dell'oggetto (es. *Spirito)* e come *value* il suo valore.
  La scelta di usare una mappa è stata dettata dalla necessità di poter conoscere sia il nome del parametro che il suo valore. Il suo uso ha inoltre permesso di generalizzare alcuni comportamenti, evitando eccessive specializzazioni all'interno delle classi derivate e il rischio di scrivere metodi dal codice molto simile tra loro.

Nella parte pubblica troviamo:

- i metodi *Getter* e *Setter* che permettono di interagire con i campi privati di *Oggetto*, riducendo l'impatto che una modifica alla progettazione della classe base causerebbe alle classi derivate. Allo stesso scopo si è deciso di non usare un accesso di tipo *Protected;*
- metodi di supporto alle operazioni della calcolatrice e di controllo input utente, le quali permettono di prevenire errori di overflow o underflow normalizzando le statistiche o reimpostando valori numerici non corretti (esempio: rarità inferiore a 1 o superiore a 10 viene automaticamente impostata al valore valido più vicino);
- operazioni di tipo generico come *Crea, Trasforma* e *Combina*, le quali agiscono direttamente sulla mappa delle statistiche, in quanto non prevedono distinzioni di comportamento in base al tipo concreto;
- metodi virtuali puri di operazioni quali *Potenzia* e *Ricicla* che sono presenti in tutte le sottoclassi e che prevedono comportamenti specifici per ognuna di esse.

Alcuni di questi metodi utilizzano funzioni incapsulate nel namespace mathOp: la sua creazione è giustificata dal fatto che tali metodi permettono di effettuare operazioni su mappe e valori generici non specifici della gerarchia. Si è deciso di renderli disponibili in questo modo anche in un'ottica di estendibilità futura, rendendoli disponibili per eventuali altre classi esterne alla gerarchia.

- Tutte le altre classi che derivano da Oggetto sono concrete, ereditano i parametri di *Livello* e *Rarità* e la mappa delle statistiche. Inoltre possiedono, nella loro parte privata, campi di tipo stringa che contengono il nome del parametro caratteristico, come per *Spirito* in *Oggetto*.

  La differenziazione tra le varie classi, oltre ai metodi implementati, avviene grazie al costruttore, che richiama quello dell'eventuale superclasse e inserisce nella mappa le nuove statistiche.

  Alcune classi, in particolare le "foglie" della gerarchia, implementano dei metodi caratteristici legati a parametri specifici o al significato dell'oggetto in questione, ad esempio:

  - estraiDa è un metodo che può essere implementato solo nelle foglie in quanto richiede una superclasse concreta per avere successo, ed ha senso pensare all'estrazione da oggetti base a oggetti derivati (come specificato nella sezione Oggetti della Guida d'Uso);
  - distribuisci è un metodo caratteristico di Cristallo, che permette di distribuire la sua *magia* solo se la *durezza* è sufficiente;
  - ripara è specifica di Unguento, un metodo che permette di ripristinare attributi deboli di altri oggetti, solo se la sua energia è sufficiente;
  - duplica è caratteristico di Amuleto, che in base alla sua *fortuna* può creare una copia di un altro oggetto.

Implementazione metodi

Qui viene riportata la firma e la descrizione dei principali metodi utilizzati:

Metodi generici

```
void combina(Oggetto* object)
```

Il metodo fa side effect sull'oggetto di invocazione. Per entrambi gli oggetti, calcola una percentuale di distribuzione basata sul livello e sul mana. All'oggetto di invocazione vengono aumentate le statistiche in base a queste percentuali, distinguendo il caso di statistiche presenti in entrambi e statistiche peculiari di ogni oggetto.

```
void trasformaDa(const Oggetto *obj)
```

Il metodo fa side effect sull'oggetto di invocazione.
Viene controllato se il suo numero di parametri è maggiore di obj. In tal caso, viene lanciata un'eccezione. Altrimenti vi sono sufficienti statistiche per inizializzare tutto. A parametri uguali viene destinato metà dell'originale valore. Il rimanente, derivante da eventuali parametri caratteristici, viene usato per incrementare le statistiche rimanenti dell'oggetto di invocazione; in questo caso la rarità funge da moltiplicatore.

```
void crea(double mana, int livello, int rarita, string statistica = "")
```

Controlla l'input e imposta i parametri di livello e rarità.
Se è stato indicato un parametro, metà del mana verrà riservato ad esso. Altrimenti viene equamente diviso tra le statistiche.

Metodi comuni con comportamento specializzato

```
virtual double ricicla() const
```

Restituisce il mana ottenuto dal riciclo dell'oggetto di invocazione. Il mana dipende dalla somma delle statistiche, dal loro numero, dal livello e dalla rarità. Si distingue in ogni classe concreta per via dei parametri caratteristici, che aumentano il mana in base alla rarità.

```
virtual void potenzia(double mana, string parametro ="")
```

incrementa le statistiche dell'oggetto di invocazione grazie al mana e al parametro. Se il parametro è errato o non esiste nella mappa, il comportamento eseguito è quello con parametro di default. Il comportamento tra i vari metodi può variare in base alla rarità dell'oggetto, al suo livello, a particolari parametri forniti in input. Generalmente i parametri caratteristici dell'oggetto ricevono sempre e comunque un incremento bonus.

Metodi specifici

```
void estraiDa(Oggetto* oggetto)
```

Il metodo appartiene solo alle classi foglia, in quanto necessita di una superclasse concreta, il cui tipo viene verificato con un typeid. Se il controllo fallisce viene sollevata un'eccezione. Il comportamento tra le varie classi varia inoltre per particolari attenzioni volte ai parametri caratteristici e possono essere previsti comportamenti diversi o concessioni di bonus maggiori per livelli o rarità alte.

```
void ripara(Oggetto* obj)
```

Specifico di unguento. Viene calcolata la differenza tra i parametri di obj con il valore più alto e più basso tramite la funzione *findMinMaxStat*. Se l'energia dell'unguento è sufficiente a coprire questo gap, in caso di rarità di unguento <= 7, la statistica più bassa di obj viene portata al valore della statistica più alta; se la rarità è >7 tutte le statistiche vengono portate al valore più alto. In entrambi i casi l'energia di unguento viene decrementata.

```
Oggetto* duplica(Oggetto * obj) {
```

Specifico di amuleto. Crea una copia perfetta o con statistiche dimezzate di obj, in base alla rarità (>= 4 oppure < 5). La fortuna deve però essere almeno pari alla media del valore delle statistiche, altrimenti la fortuna scenderà a 1 e la copia avrà tutti i parametri a 1.

```
void distribuisci(Oggetto* obj)
```

Specifica di cristallo. Se la durezza è almeno la metà della magia, dimezza la propria magia per potenziare obj.

Installazione gerarchia: class Model

La class Model fa da ponte tra la gerarchia e il controller. Include tutte le classi della gerarchia e tiene traccia degli oggetti creati in una lista di puntatori a Oggetti che funziona come una pila. Le operazioni selezionate dall'utente vengono eseguite generalmente sull'ultimo e penultimo elemento memorizzato.

L'uso di polimorfismo è reso quindi necessario per via di questa pila: avendo una lista di puntatori a Oggetto, è possibile accedere a tutti i metodi definiti in Oggetto, garantendo comunque la selezione del metodo corretto in base al tipo dinamico in caso di metodi virtuali, quali *Potenzia* e *Ricicla*, virtuali puri nella classe base ma che hanno un comportamento diverso per ogni tipo concreto derivato.

I metodi della classe model catturano le eccezioni sollevate dalla gerarchia, e solleva eccezioni di view, che verranno poi stampate sul display. Può emettere inoltre altre eccezioni causate da letture o scritture da memoria vuota, le quali sono catturate dal controller ma non gestite, in quanto la logica di questa particolare view non lo necessita.

Vengono memorizzate in una mappa anche le immagini relative agli oggetti, per poter essere recuperate successivamente e visualizzate come risultato nel display della view.

Controller

Il controller collega Model a View, fornisce una cache temporanea che contiene le informazioni relative all'impostazione di oggetti e operazioni.

Utilizza la classe *DisplayAndSlider*, la quale gestisce il pannello di espansione che viene visualizzato nella View quando viene cliccato un oggetto o un'operazione che richieda l'inserimento di dati in input.

Gli slider di questa classe sono collegati al controller, il quale preleva i loro valori e crea gli oggetti necessari quando selezionati. Alla conferma dell'operazione, il pannello di espansione viene nascosto ed eliminato, le informazioni passate alla memoria, e la "cache" del controller liberata.

View

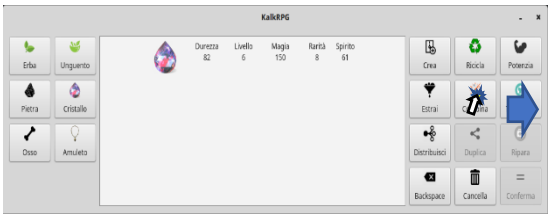
La View è implementata nella classe KalkRpg derivata da QWidget. Fa uso a sua volta di una classe Display ereditata da QTextEdit e da una classe Button derivata da QToolButton

KalkRpg

Il costruttore di KalkRpg crea tutti i pulsanti, connettendo i vari slot che permettono alla grafica di aggiornarsi in base allo stato in cui si trova la calcolatrice. Ha un layout impostato con un QGridLayout e al suo interno vengono disposti i vari pulsanti, raggruppati in QGridLayout proprie, e il display.

Per agevolare l'esperienza utente, la calcolatrice è stata organizzata a stati, e in ogni momento vengono disattivati tutti i pulsanti che non possono essere cliccati. Lo stato iniziale della calcolatrice tiene attivi solo i pulsanti degli oggetti e di due operazioni, *Crea* e *Ricicla.*

La pressione di ogni pulsante causa l'emissione di segnali che modificano apposite variabili booleane che portano alla disabilitazione o abilitazione di pulsanti, in modo da rendere possibile sempre e solo le scelte corrette.



Ogni volta che l'utente deve inserire dei dati in input viene fatto apparire, sotto al display e ai pulsanti, un pannello di espansione contente degli slider e delle caselle di testo su cui può inserire i valori, per i quali sono previsti un limite inferiore o superiore. Per la selezione dei parametri, se previsto, è reso disponibile un menu a tendina. Ogni input deve essere confermato con l'apposito pulsante di conferma. Ogni selettore è collegato tramite un puntatore alla cache del controller, il quale avrà poi il compito di aggiornare la memoria statica della calcolatrice una volta confermato l'inserimento.

Viene fornita anche una classe di eccezioni che permette di fare il catch delle eccezioni sollevate dal model, e visualizzare l'errore sul display.

Button

E' una classe che deriva da QToolButton e che ne ridefinisce le dimensioni consigliate. La scelta di derivarla da QToolButton è giustificata da una migliore gestione di testo sotto l'icona, rendendo il pulsante più esplicativo. La classe salva il percorso dell'immagine usata e il testo del pulsante per permetterne un facile riutilizzo qualora

fosse necessario mostrare tali informazioni nel display o nel pannello di espansione. Oggetti, operazioni e pulsanti di gestione memoria sono di tipo Button.

Display

Deriva da QTextEdit, impostato in sola lettura. Nella parte privata mantiene una lista di interi che usa per memorizzare la posizione dei cursori prima dell'inserimento di ogni informazione, in modo da agevolare l'utilizzo del pulsante backspace e mantenere la coerenza tra display e memoria.

Ridefinisce il metodo *clear()* per pulire il display e la lista di posizioni di cursori, e implementa, sovraccaricandolo, il metodo *show()* che permette di stampare l'immagine e le informazioni degli oggetti, l'immagine dell'operazione e i diversi valori dati in input.

Offre un metodo *back()* che cancella l'ultima informazione stampata a display, recuperando e distruggendo l'informazione più recente dalla lista precedentemente descritta.

Ambiente di Sviluppo

Mirko		Giovanni	
SO:	Linux Mint 18.3	SO:	Windows 10
C++		C++	
Editor:	Qt Creator 4.4.1	Editor:	Qt Creator 4.4.1
Compilatore:	gcc 5.4.0	Compilatore:	gcc 5.4.0
Qt:	5.5.1	Qt:	5.5.1
Java		Java	
Compilatore:	jdk 1.8.3	Compilatore:	jdk 1.8.3
IntelliJ	2017.3.4	IntelliJ	2017.3.4

Suddivisione per fasi delle ore di lavoro

	Giovanni	**Mirko**
Studio	8	8
Idealizzazione e progettazione	7	8
Scrittura codice	37	31
Debugging	8	7
TOTALE	**60**	**54**

Le ore in eccesso sono dovute a piccoli problemi riscontrati con la libreria Qt, la cui soluzione non è stata immediata per via della non conoscenze del programma.