Programmazione ad Oggetti - Relazione YouKalk

Davide Barasti

Febbraio 2018

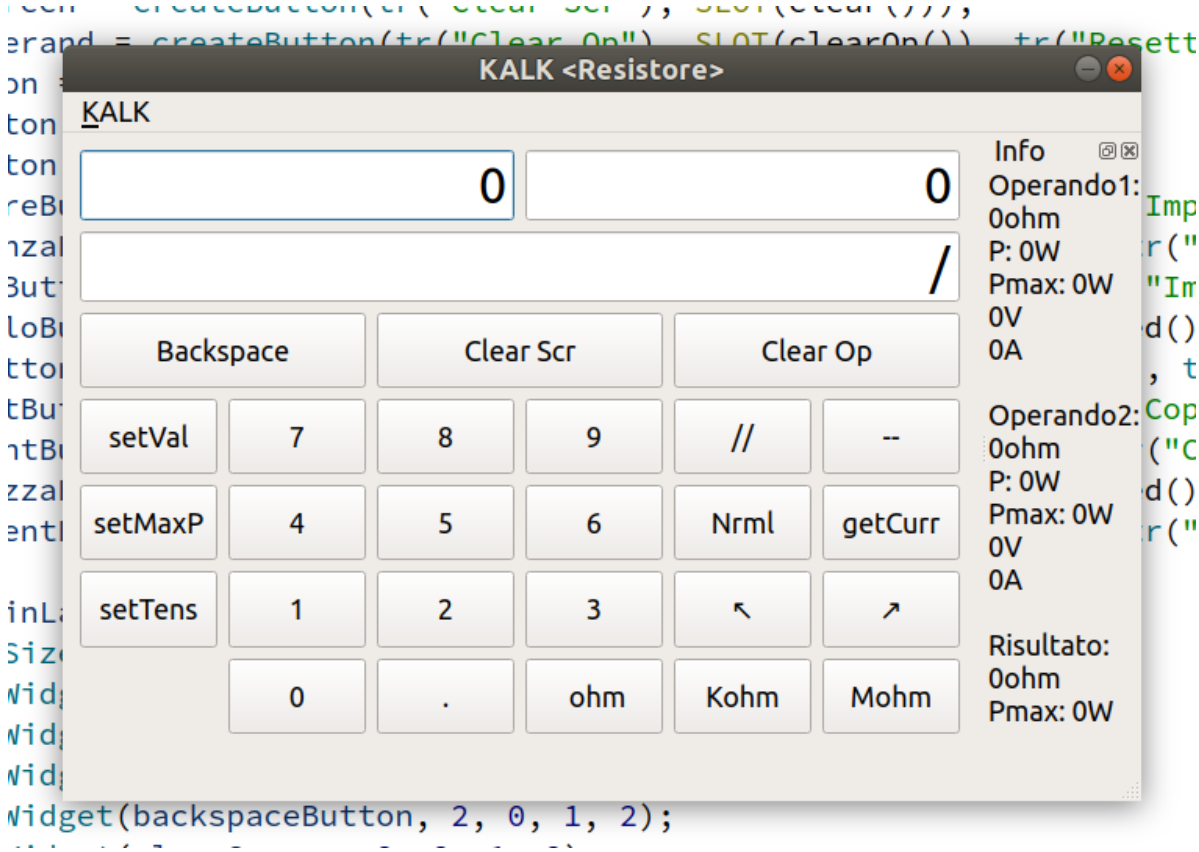

Figure 1: Schermata della calcolatrice in modalità *Resistore*

1 Introduzione

Con questo progetto si è voluta realizzare una calcolatrice multifunzione che possa esegure calcoli con i principali *componenti elettrici* più utilizzati.

2 Informazioni sullo sviluppo

Il progetto è stato realizzato su Ubuntu 17.10, Qt 5.9.2, GCC 5.3.1 (Red Hat 5.3.1-6), 64 bit.
Testato in ultimo via VM da termilale con comandi qmake->make.

3 Compilazione

Per la compilazione è sufficiente generare il Makefile tramite comando *qmake* e lanciare poi *make*.

4 Tipi di dato

Gli oggetti che l'utente può utilizzare e modificare durante una sessione di *YouKalk* sono: **Resistori**, **Condensatori** e **Induttori**.

5 Set operazioni disponibili

Vi sono diversi tipi di operazioni disponibili, alcune possono riferirsi ad uno solo dei due operandi mentre altre producono un risultato dato da una computazione su due operandi.
Le operazioni su due operandi sono intese su coppie di operandi dello stesso tipo.
Di seguito vengono mostrate le operazioni in comune e quelle specifiche per **resistori**, **condensatori** e **induttori**;

5.1 Definizione degli operandi

Set di operazioni che vanno a definire le proprietà di un operando.

- **setVal**: imposta il valore relativo alla grandezza fisica, proprietà principale del componente:
  - per il resistore si tratta della *resistenza elettrica* che chiameremo semplicemente *resistenza* definita come

    > [...]una grandezza fisica che misura la tendenza di un corpo ad opporsi al passaggio di una corrente elettrica, quando sottoposto ad una tensione elettrica.

    che si misura in **ohm**;
  - per il condensatore si parla di *capacità elettrica* o semplicemente *capacità* definita come

    > [...]una grandezza fisica che quantifica l'attitudine di un corpo conduttore ad accumulare carica elettrica qualora sia dotato di un potenziale elettrico.

    che si misura in **farad**(F).
  - infine per l'induttore si intende l'*induttanza* definita come

    > [...]la proprietà dei circuiti elettrici tale per cui la corrente (intesa variabile nel tempo) che li attraversa induce una forza elettromotrice.

- **setTens**: imposta il valore della tensione ai capi dell'operando. Nel caso dell'induttanza si tratta di *tensione alternata* mentre per gli altri due componenti di *tensione continua*;
- **setFrq**: imposta il valore della frequenza di oscillazione dell'onda sinusoidale del generatore di tensione, solo nel caso in cui sia una tensione alternata;
- **Clear Op**: esegue un reset dell'operando azzerando tutti i relativi valori.

5.1.1 Resistore

- **Ohm**: imposta il prefisso dell'unità di misura con un fattore moltiplicativo pari a $10^0 = 1$;
- **Kohm**: imposta il prefisso dell'unità di misura a K(chiloohm) con fattore moltiplicativo $10^3 = 1000$;
- **Mohm**: imposta il prefisso dell'unità di misura a M(megaohm) con fattore moltiplicativo $10^6 = 1000000$;
- **setMaxP**: imposta la potenza massima dissipabile dal resistore senza che si danneggi irreparabilmente.

5.1.2 Condensatore

- **nF**: imposta il prefisso dell'unità di misura a n(nanofarad) con un fattore moltiplicativo pari a 10^{-9};
- **uF**: imposta il prefisso dell'unità di misura a u(microfarad) con fattore moltiplicativo 10^{-6};
- **mF**: imposta il prefisso dell'unità di misura a m(millifarad) con un fattore moltiplicativo pari a 10^{-3}.
- **setRes**: imposta il valore del resistore R necessario in un circuito di carica di un condensatore affinchè questa avvenga in un tempo tale da non danneggiare il condensatore.
  Solitamente questo valore è nell'ordine dei Kohm.

5.1.3 Induttore

- **nH** imposta il prefisso dell'unità di misura a n(nanoHenry) con un fattore moltiplicativo pari a 10^{-9};
- **uH** imposta il prefisso dell'unità di misura a u(microHenry) con fattore moltiplicativo 10^{-6};
- **mH** imposta il prefisso dell'unità di misura a m(milliHenry) con un fattore moltiplicativo pari a 10^{-3}.
- **setRes** imposta il valore del resistore R del circuito di riferimento R-L.

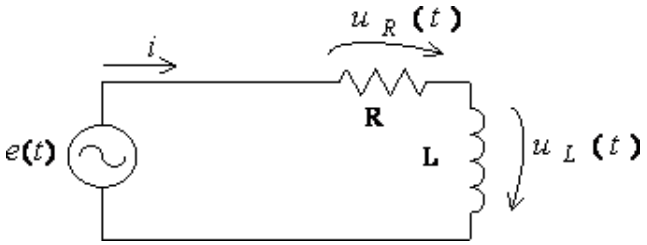

Figure 2: Circuito R-L di riferimento

5.2 Operazioni avanzate comuni

5.2.1 Binarie

- **Parallelo**: viene eseguita l'operazione del calcolo del valore equivalente tra due operandi posti in parallelo;

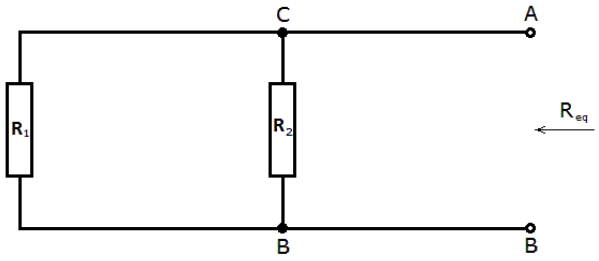

Figure 3: Parallelo tra due componenti, in questo caso tra *resistori*

- **Serie**: viene eseguita l'operazione del calcolo del valore equivalente tra due operandi posti in serie.

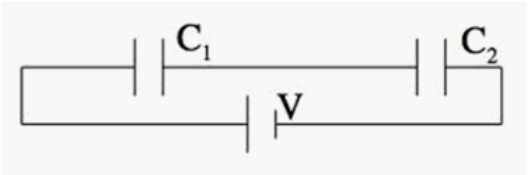

Figure 4: Serie tra due componenti, in questo caso tra *condensatori*, **V** è un *generatore di tensione*

Un tentativo di eseguire operazioni con valori non validi mostrerà un errore specifico.

Componente	Parallelo	Serie
Resistore	$\frac{R_1+R_2}{R_1\times R_2}$	R_1+R_2
Condensatore	C_1+C_2	$\frac{C_1+C_2}{C_1\times C_2}$
Induttore	$\frac{I_1+I_2}{I_1\times I_2}$	I_1+I_2

Table 1: Le operazioni di parallelo e serie in base al tipo di dato

5.2.2 Unarie

- **Nrml**: dato un valore che rappresenta la proprietà principale del componente , quando possibile viene ottimizzato modificando il valore stesso e il prefisso dell'unità di misura per migliorare la leggibilità.

Componente	Valore prima	Valore dopo
Resistore	1800Ω	1.8KΩ
Resistore	0.005MΩ	5KΩ
Condensatore	7500uF	7.5mF
Induttore	0.013mH	13uH

Table 2: Esempio funzione *Nrml*

5.3 Operazioni avanzate specifiche unarie

5.3.1 Resistore

- **getCurrent**: calcola la corrente in *Ampere* che scorre attraverso il resistore. La condizione necessaria è che sia stata applicata una Tensione $V > 0$.
  La formula applicata per il calcolo della corrente è: $I = \frac{V}{R}$;
- **setPotenza**: calcola la potenza dissipata dal resistore in *Watt* risultante dal passaggio di una corrente I. Implica una tensione V applicata ai capi del resistore[1] .
  La formula applicata per il calcolo della potenza è: $P = V \times I$.
  Qualora la **potenza dissipata** sia maggiore della *potenza massima* impostata(con $PMax > 0$) verrà segnalato con un *Warning* grazie all'eccezione *alert*.

5.3.2 Condensatore

- **getTcarica**: calcola il tempo che il condensatore impiega a caricarsi data la capacità del condensatore e il resistore utilizzato per la carica tramite la formula: $T_c = 5 \times R \times C$.
- **V(t)**: calcola la tensione ai capi del condensatore durante la carica in un istante di tempo t data la tensione ϵ applicata ai capi del circuito, la resistenza R e il condensatore[1].
  La formula utilizzata è: $V(t) = \epsilon \times (1 - e^{\frac{-t}{RC}})$

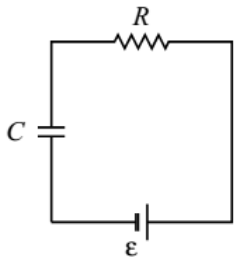

Figure 5: Circuito RC di carica del condensatore

5.3.3 Induttore

- **getCurr**: calcola il valore della corrente in Ampere che scorre attraverso il circuito R-L. La condizione necessaria è che sia stata applicata una Tensione $V > 0$.
  La formula applicata per il calcolo della corrente è: $I = V_{RMS}\sqrt{R^2 + (\omega \times L)}$ dove ω è la pulsazione in rad/s;

[1] Questa operazione è nascosta all'utente, avviene quindi in background. È comunque mostrata nei risultati.

6 Analisi della gerarchia

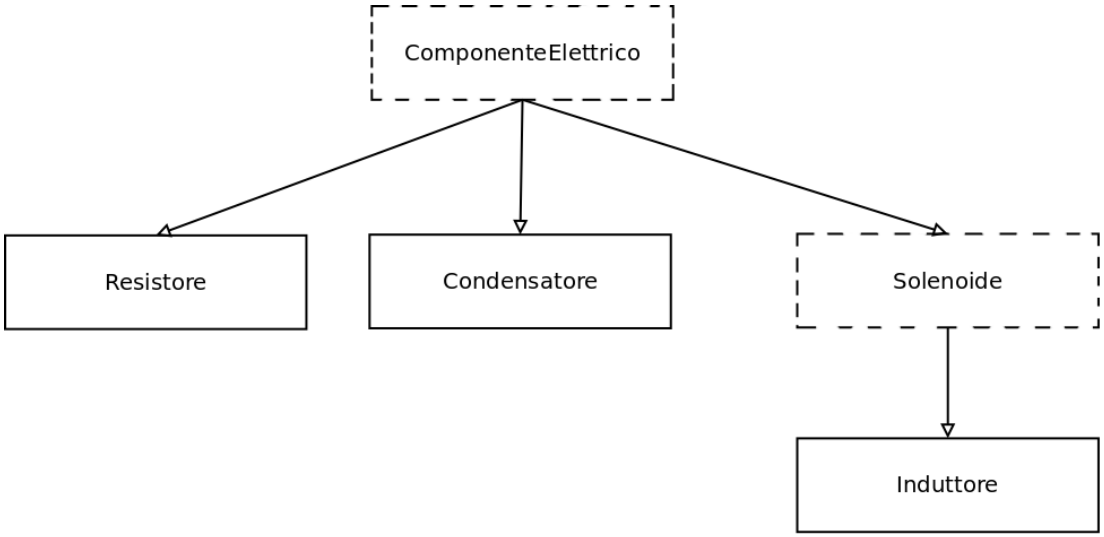

Figure 6: gerarchia principale su cui è basato il modello

ComponenteElettrico è una classe base astratta superclasse degli oggetti che vengono utilizzati nelle operazioni. Qualora un'operazione non sia possibile con gli operandi specificati viene lanciata l'eccezione *BadOperation* con un preciso messaggio di errore.
All'interno del file eccezioni(.h/.cpp) si trovano tutte le classi relative agli errori.

In ComponenteElettrico vengono definiti puri i due metodi che rappresentano le operazioni principali eseguibili: *Serie* e *Parallelo*.
Nei campi dati abbiamo il valore principale comune alle classi derivate (resistenza, capacità e induttanza) e la tensione applicata al singolo componente in AC(corrente alternata) e DC(corrente continua). **Valore_principale** e **tensioneDC** sono di di tipo *Misura*, una classe modellata appositamente per gestire un valore e l'unità di misura mentre *TensioneAC* è di tipo *MisuraTA*, una classe derivata da Misura che la espande aggiungendo i campi **frequenza** e **RMS** (entrambi di tipo Misura) necessari per registrare una tensione alternata.
Di seguito una rappresentazione dei campi dati Misura:

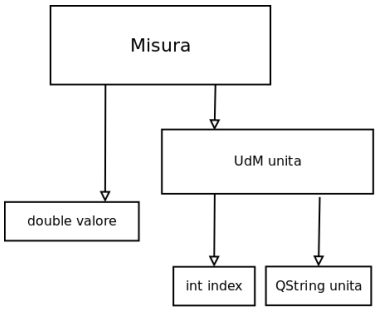

Figure 7: derivazione dei campi dati che rappresentano una Misura

Tra i capi dati di UdM si nota il campo index; questo rappresenta l'indice attraverso il quale si può risalire al carattere prefisso dell'unità di misura. Si riferisce ad un vettore statico di prefissi e permette anche di calcolare il fattore moltiplicativo del prefisso.

Resistore e **Condensatore** estendono ComponenteElettrico implementando i metodi puri e aggiungendo campi dati e metodi specifici: Resistore aggiunge tre campi dati di tipo Misura: *potenzaMax*, *corrente* e *potenza*, Condensatore aggiunge due campi: *t_ Carica* e *resistenza*

Solenoide estende ComponenteElettrico aggiungendo due capi dati non utilizzati in questa versione di YouKalk: uno rappresenta la lunghezza dell'avvolgimento mentre un secondo campo indica il numero di spire. Non implementando i metodi virtuali puri di ComponenteElettrico questa classe rimane **astratta**.

Induttore estende Solenoide e implementa i metodi puri di ComponenteElettrico rendendo la classe concreta e aggiunge due campi: *resistenza* e *corrente*.

Sfruttando il polimorfismo diventa quindi possibile eseguire le operazioni in comune tra tipi diversi in maniera trasparente (*serie e parallelo*).

Oltre a questi particolari vengono ovviamente aggiunte in ogni classe derivata i metodi specifici necessari per svolgere le operazioni elencate sopra.

7 GUI

I file relativi a Controllo e View sono i seguenti(.h e .cpp):

- Application;
- kalk;
- kalkC;
- kalkI;
- kalkR;
- button_mod;
- display_mod;

I file kalk* sono una gerarchia di classi, utilizzate per costruire la view. Ognuna si occupa di creare e disporre i componenti necessari alla singola modalità.
Kalk costruisce tutto quello che è in comune nell'interfaccia come gli schermi degli operandi e alcuni bottoni. Gli altri tre file costruiscono le parti specifiche dell'interfaccia.
In base alla modalià di esecuzione selezionata verrà istanziato un puntatore polimorfo alla classe desiderata.
In generale le quattro classi possono essere riassunte come segue:
viene creata ed allineata la vista e viene effettuato il collegamento con una *connect* dei segnali dei bottoni con gli slot della classe Application che si occupa della parte di *controllo*.
Nelle classi *Button*, derivata da *QPushButton* e *Display*, derivata da QLineEdit si sono reimplementate le classi base aggiungedo alcuni dettagli utili per realizzazione dell'applicazione.

Per l'interfaccia utente è stata derivata da QMainWindow una classe Application che si occupa di creare:

- Menù;
- DockWidget;
- Status Bar;

e di istanziare a comando una delle tre classi disponibili per la creazione della calcolatrice di cui si è scritto sopra.

La scelta di lasciare alla classe Application la gestione degli eventi scatenati dai Widget istanziati deriva dal voler isolare la costruzione del widget dalla gestione del sistema.

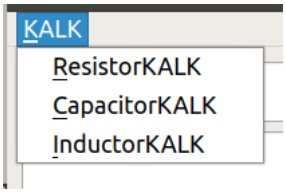

Figure 8: Menù di scelta

Attraverso il *menù* è possibile cambiare la modalità di Kalk scegliendo tra le tre disponibili.

La *StatusBar* rende facile l'utilizzo della GUI e in caso di bisogno viene utilizzata per comunicare piccoli messaggi di errore o di avviso già menzionati in precedenza.

Il DockWidget posto in posizione laterale permette di avere un quadro chiaro della situazione degli operandi utilizzando una QLabel per stampare i campi dati.

Figure 9: Messaggio di errore nella *status bar* conseguente al throw dell'eccezione badOperation

È stata rispettata la condizione di separare totalmente il modello logico dal codice della GUI. L'unico collegamento con le librerie di QT presente nel modello è l'utilizzo di *QString*, una scelta che ha evitato di implementare funzioni di elaborazioni di stringhe già comodamente presenti in questa calsse.

8 Manuale utente

La GUI è sviluppata in modo da permette una semplice interazione con la calcolatrice. Di seguito vengono elencati i punti principali dell'interazione con *YouKalk*:
Nelle sessioni **CapacitorKALK**, **ResistorKALK** e **InductorKALK** gli operandi sono rappresentati da i due schermi superiori mentre il risultato delle operazioni verrà mostrato nello schermo inferiore.
Per interagire con l'operando è sufficiente selezionare lo schermo corrispondente.
I valori vengono inseriti tramite i tasti numerici e ad ogni operazione è associato un bottone.
Ulteriori informazioni sono sempre visualizzabili nella *barra di stato* inferiore passando il puntatore sopra la funzione desiderata.

9 Osservazioni

Sono state eseguite delle prove di correttezza delle classi del modello tramite l'utilizzo di **Assert**. Questo ha permesso di procedere con una certa sicurezza nella creazione delle gerarchie perchè qualora si presentasse un problema, questo non poteva venire dallo strato inferiore poichè era stato solidamente testato.

10 Ore impiegate

Ore totali: 55
Analisi preliminare: 10%
Progettazione modello: 20%
Progettazione GUI: 35%
Libreria QT: 10%
Codifica modello e GUI: 20%
Debugging/testing: 5%