Progetto di Programmazione ad Oggetti

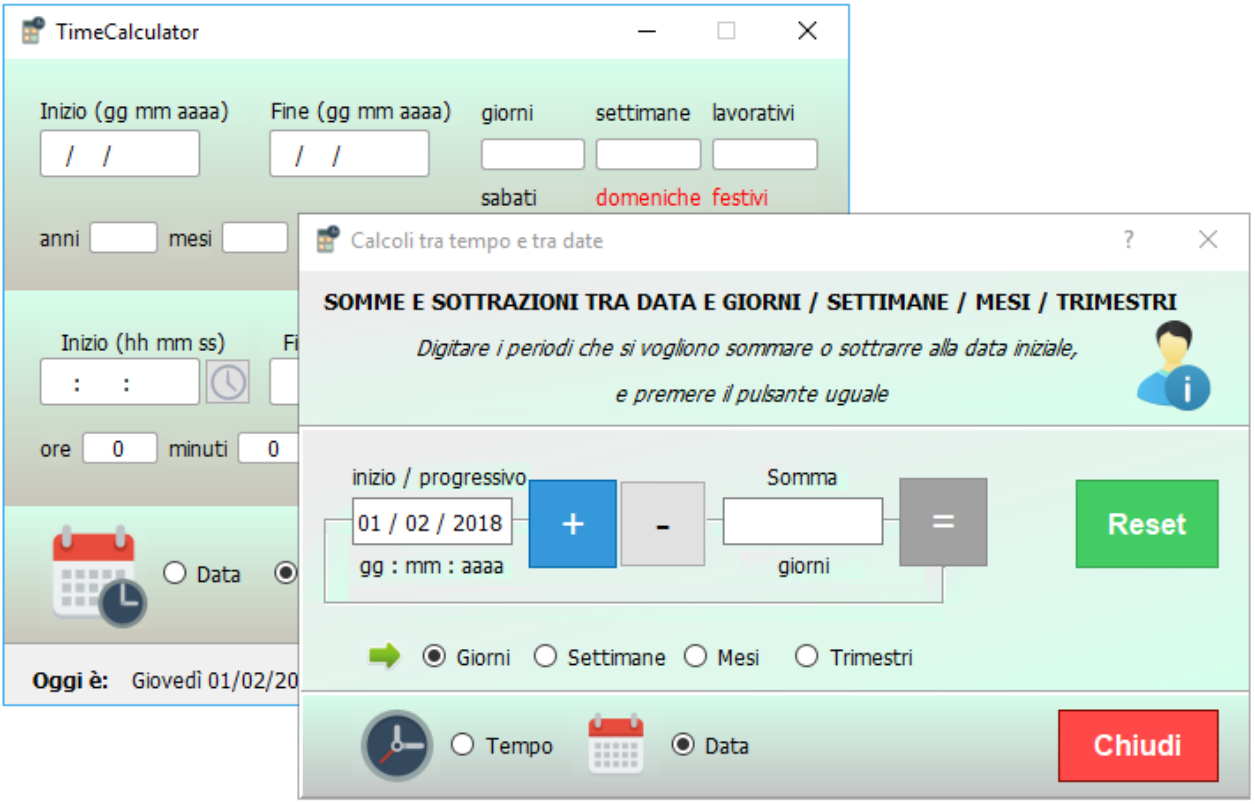
Calcolatrice estendibile in C++

Autore:	Dal Maso Daniele
OS di sviluppo:	Windows 10 Home
IDE utilizzato:	Qt Creator 3.5.1
Versione di Qt:	5.5.1
Compilatore:	GCC 5.1.0

Università degli studi di Padova

Sommario

1 Scopo del progetto



Lo scopo del progetto è di fornire un applicazione per calcolare la distanza tra due date e orari. Attraverso una semplice interfaccia siamo in grado di effettuare svariati tipi di calcolo tra date, ore e diversi momenti della giornata. TimeCalculator è una "calcolatrice temporale" che, inserite due date in ingresso, effettua il conteggio dei giorni totali, dei soli giorni lavorativi e festivi e delle settimane di mezzo, tenendo conto anche dei sabati e delle domeniche ricorrenti.

Per il calcolo delle ore, è possibile memorizzare l'ora attuale di inizio attività con un click sul pulsante a forma di lancetta dell'orologio, e allo stesso modo sempre con un altro click memorizzare l'orario di fine. Ciò può essere utile per tenere conto del tempo speso in un qualsiasi progetto o impegno.

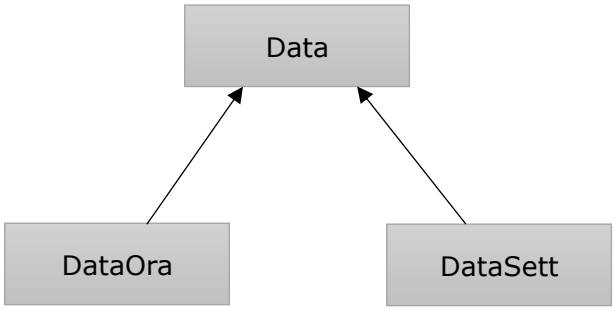
Cliccando sul pulsante *"Operazioni"* in fondo all'interfaccia, si apre una nuova finestra su cui è possibile eseguire operazioni più specifiche. Ad esempio, si possono sommare o sottrarre dei giorni a una data di inizio e sapere la data finale, oppure aggiungere o togliere ore minuti e secondi a un determinato orario.

Le funzionalità dell'applicazione sono quindi molteplici, si può ad esempio utilizzare per calcolare date di scadenza, sapere quanti giorni mancano ad un evento speciale, come un matrimonio o un esame, oppure sapere quante ore abbiamo dedicato a una specifica attività e così via.

L'idea del software è nata mentre stavo salvando sul calendario le date degli appelli d'esame. Non essendo presente sul calendario un promemoria più dettagliato, che mi dicesse ad esempio quanti giorni mancavano a un determinato appello, decisi di implementare questa feature nel mio progetto e da lì ho esteso l'idea che si è poi tramutata in una calcolatrice per l'esecuzione di calcoli tra date e orari.

2 Descrizione della gerarchia

Di seguito è illustrata la gerarchia dei tipi adottata:



Per sviluppare la logica del software, mi sono concentrato su pochi ma basilari tipi di dato, sufficienti per le funzionalità di calcolo richieste dal programma. Si è cercato di separare quanto più possibile il **modello logico** dal **codice della GUI**, consentendo inizialmente l'utilizzo della calcolatrice da riga di comando, senza il supporto di una GUI.

Per rappresentare i seguenti tipi di dato, non è stata usata alcuna classe o libreria particolare di Qt o di C++, ma si è invece cercato di costruire il codice a mano, per mettere meglio in evidenza i principi della OOP. L'unico utilizzo della classi *QDate* e *QTime* è servito puramente per il calcolo della data e dell'ora corrente.

La gerarchia è molto semplice, ed è caratterizzata da tre tipi di dato su cui TimeCalculator offre delle funzionalità di calcolo:

- Il tipo **Data** mi rappresenta solo una data, in formato gg/mm/aaaa. E' il tipo base della gerarchia da cui ereditano tutti gli altri tipi. La classe è **polimorfa**, in quanto prevede un metodo virtuale *toString(),* che assume significato diverso a run-time a seconda dell'oggetto su cui è invocato (*late binding*), e il relativo *distruttore polimorfo*.

- Il tipo **DataOra** mi rappresenta un preciso orario della giornata, in formato hh:mm:ss. Estende le funzionalità della classe Data aggiungendo nuove proprietà relative all'orario. Fornisce un implementazione (override) del metodo *toString()* della casse base.

- Per ultimo abbiamo il tipo **DataSett**, sempre derivato dalla classe Data, che aggiunge la possibilità di memorizzare il giorno della settimana. Anche qui vi è un implementazione propria del metodo *virtual* della superclasse.

A questi tre tipi, inoltre, si aggiungono due tipi classe di eccezioni (**TimeException** e **DateException**) da me definiti per la gestione dei problemi di input. Ogni tipo di errore viene gestito con opportune chiamate che riportano un chiaro messaggio di errore all'utente (mediante *QMessageBox*), senza corrompere lo stato del programma o provocare errori run-time.

3 Interfaccia grafica

Per la realizzazione dell'interfaccia grafica si è fatto uso del *Qt Designer*, lo strumento visuale messo a disposizione dalla libreria Qt per la creazione di elementi grafici. Le form costruite nel nostro caso sono due, quella principale per il conteggio dei tempi di distanza tra due date e/o ore, e quella secondaria per le operazioni vere e proprie. Nel primo caso si è utilizzato un *QMainWindow*, mentre nel secondo caso un *QDialog*. I files generati dal Qt Designer sono rispettivamente: *timecalculator.ui* e *calcdialog.ui*.

Per integrare il codice prodotto dal Qt Designer nell'applicazione, si possono utilizzare tre metodi: *The Direct Approach, The Single Inheritance Approach e Automatic Connections.*

In questo caso si è deciso di adottare l'approccio **Single Inheritance** usando un variabile puntatore, con il vantaggio che l'oggetto dell'interfaccia utente può essere dichiarato più avanti, il che significa che non dobbiamo includere il file generato ui_timecalculator.h nell' header timecalculator.h. Il form può quindi essere modificato senza la ricompilazione dei file sorgente dipendenti.

Per maggiori informazioni ci si può riferire all'esauriente documentazione accessibile con il comodo Qt Assistant.

Terminata l'interfaccia grafica, per far comunicare tra loro i widgets viene utilizzato il meccanismo dei **Signals and Slots**, offerto da Qt. Vediamo un esempio d'uso di una connessione signal-slot per il pulsante "=" del form calcdialog.ui, risiedente nel file calcdialog.cpp del progetto.

```
1. connect(ui->btnEq, SIGNAL(clicked(bool)), calcManager, SLOT(setResult()));
```

In questo caso l'oggetto *btnEq* (il pulsante uguale) emette il segnale *clicked()* che è ricevuto dallo slot *setResult()* dell'oggetto *calcManager* (istanza della classe **CalcManager** da me creata per gestire le operazioni di calcolo). Nel file calcmanager.cpp, lo slot *setResult()* è definito come segue:

```
1.  void CalcManager::setResult() {
2.      if(ui->radioTime->isChecked()) {
3.          QString orario = ui->textOra2->text();
4.          QString ore = orario.left(2);
5.          QString min = orario.section(':', 1, 1);
6.          QString sec = orario.right(2);
7.
8.          time_operand = Orario(ore.toInt(), min.toInt(), sec.toInt());
9.          doOperation();
10.
11.         QString res = time_res.toString();
12.         ui->textOra1->setText(res);
13.     }
14.     else {
15.         date_operand = ui->textData2->text().toInt();
16.         doOperation();
17.
18.         QString res = date_res.toString();
19.         ui->textData1->setText(res);
20.     }
21. }
```

Senza addentrarci troppo nei dettagli, possiamo vedere che in *setResult(),* viene eseguita l'operazione *doOperation()* e viene stampato a video nella casella di testo *textOra1* o *textData1* (a seconda di quale opzione abbiamo selezionato nel radioButton) il risultato *res*.

3.1 Guida all'utilizzo della GUI

Istruzioni per l'utilizzo dell'interfaccia utente.

Finestra principale

Calcoli tra date

Inserire nelle caselle 'Inizio' e 'Fine' le rispettive date di cui si vogliono effettuare i calcoli e cliccare il tasto 'Calcola'. Vengono mostrati i giorni, i mesi e gli anni che intercorrono tra le due date immesse. In più vengono visualizzati i giorni complessivi, il numero di settimane, il numero di giorni lavorativi (calcolati sottraendo le domeniche e i giorni festivi dai giorni complessivi), il numero di sabati, il numero delle domeniche e il numero dei giorni festivi presenti nel periodo indicato.

Calcoli con le ore

Inserire nelle caselle 'Inizio' e 'Fine' i rispettivi orari di cui si vuole effettuare il calcolo e premere il tasto 'Calcola'. Vengono mostrate le ore, i minuti e i secondi che intercorrono tra i due orari immessi; inoltre vengono calcolati gli intervalli di tempo complessivo, espressi in ore, minuti e secondi tra 'Inizio' e 'Fine'. Se l'orario immesso nella casella 'Inizio' è maggiore dell'orario di 'Fine', il secondo orario viene considerato come successivo (cioè riferito al giorno seguente).

Finestra secondaria

Calcoli tra unità di tempo

Inserire nella casella 'inizio/progressivo' un orario, scegliere il tipo di operazione (+/-) cliccando sul tasto corrispondente, inserire nella seconda casella quante ore, minuti e/o secondi si vogliono addizionare/sottrarre all'orario di inizio e cliccare il pulsante '='. La casella 'inizio/progressivo' viene aggiornata col nuovo orario, ottenuto con la precedente operazione.

Calcoli tra date e giorni

Anche qui valgono le stesse regole esposte sopra. L'utente può scegliere se aggiungere/togliere giorni oppure settimane, mesi o trimestri alla data di inizio, specificando la quantità nelle rispettive caselle di testo.

Con il tasto 'Reset' si azzerano i dati presenti nelle caselle di testo. Nel caso delle date, la casella relativa alla data di inizio viene impostata di default alla data corrente.

4 Tempo richiesto

- **Analisi preliminare del problema:** 2 ore
- **Progettazione modello e GUI:** 4 ore
- **Apprendimento libreria QT:** 6 ore
- **Codifica modello e GUI:** 48 ore
- **Debugging:** 5 ore
- **Testing:** 4 ore

5 Considerazioni generali

- **Gestione degli errori**: gli errori sono stati gestiti mediante due classi opportune, illustrate in precedenza.
- **Overloading di operatori**: dove possibile è stato effettuato l'overloading per gli operatori interessati. Gli operatori per i quali non era possibile definire l'overloading sono stati sostituiti da analoghe funzioni definite esternamente alla classe.
- **Gestione della memoria**: l'applicazione non fa un uso eccessivo della memoria. La definizione dei metodi costruttore di copia, assegnazione e distruttore è stata lasciata dove possibile al compilatore per motivi di efficienza, mediante la keyword *default*. Nessuna copia profonda è stata fatta, non essendoci condivisione di memoria tra gli oggetti.
- **Polimorfismo**: è stato fatto uso di codice polimorfo, mediante chiamate a metodi virtuali. Ad esempio, nella definizione del metodo *diffGiorni(…)* all'interno del file *Data.cpp* definiamo un puntatore *dt1* se cui eseguiamo ad ogni iterazione del ciclo while il metodo *toString()* del tipo dinamico associato. Essendo *DataSett* il tipo dinamico di dt1, a run-time viene invocato il metodo *DataSett::toString()*. Un altro uso del polimorfismo viene fatto nella funzione *showcurretnDate()* presente nel file *timecalculator.cpp*. In questo caso abbiamo sempre un puntatore polimorfo se cui viene invocato il metodo *toString()* ridefinito (*override*) nella sottoclasse *DataSett* per ottenere il giorno della settimana, mentre per avere solo l'informazione sull'orario (metodo *toString()* della classe base), si è utilizzato l'operatore di scoping. Anche nella funzione *showcurrentTime()* per il calcolo dell'ora corrente, viene fatto uso di polimorfismo. In questo caso viene selezionato il metodo *toString()* ridefinito in *DataOra*.
- **Correttezza**
  - Reso il codice più estensibile ed evolvibile, mediante polimorfismo.
  - Rispettata la specifica richiesta sulla gerarchia, introducendo un nuovo tipo C_2 (*DataSett*) sottotipo di *B* (*Data*). Rimosso il tipo separato *DiffData* per ridondanza con il tipo *Data*.
  - Risolti alcuni bug sui messaggi di errore utente e sugli output del programma.