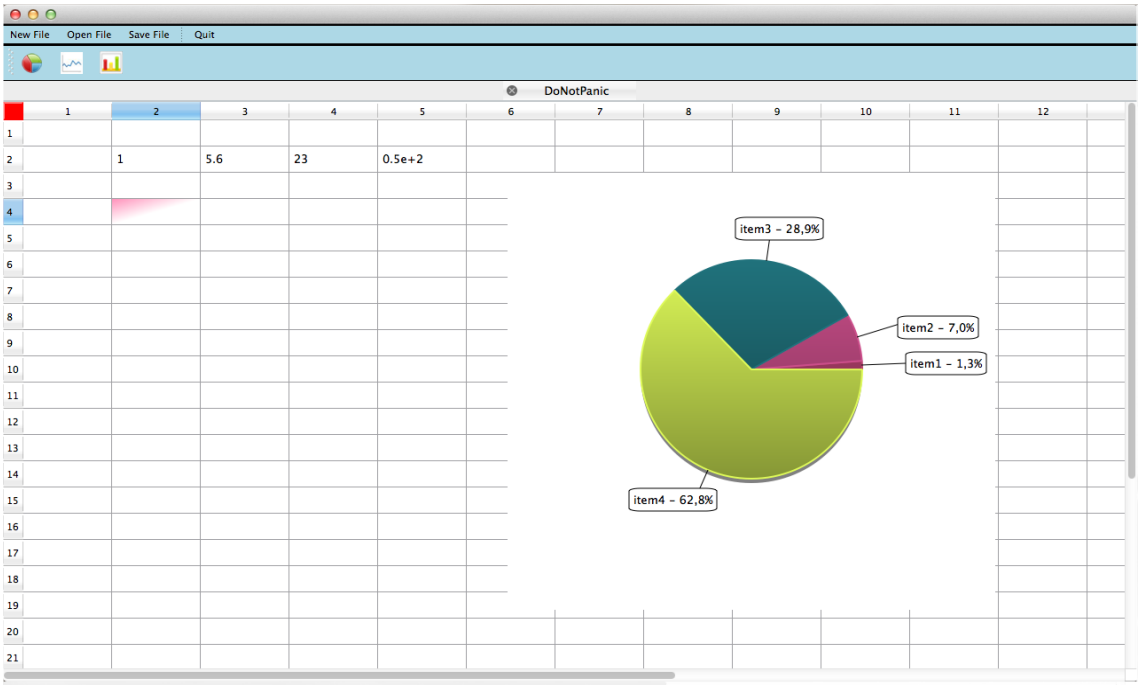
Programmazione ad oggetti

Relazione qCharts

Sartoretto Massimiliano
matricola n.1008168

Anno Accademico 2012/13



1 Introduzione

Il progetto si propone di fornire uno strumento capace di creare, modificare, visualizzare e archiviare diagrammi che mettono in relazione i dati inseriti dall'utente.

2 Informazioni su sviluppo e compilazione

Il progetto è stato sviluppato su MacOS con compilatore GCC (x86 64bit) e versione 4.8.4 di Qt. Per compilare si esegua il comando *qmake Charts.pro* dalla directory del progetto generando il Makefile, eseguire quindi il comando *make* per creare l'eseguibile.

3 Descrizione della parte logica

- **chart** è la classe base astratta della gerarchia dalla quale derivano *linechart, histogramm e piechart*. Contiene due metodi virtuali puri: `draw(QPainter*)` e `drawLegend(QPainter*)` che vengono concretizzati nelle sottoclassi. Contiene una classe *chartPiece* interna e annidata nella parte protetta, che rappresenta un "pezzo" del chart caratterizzato da nome, colore e valore percentuale rispetto all'insieme cui appartiene. Un oggetto chart è quindi rappresentato da un QVector di chartPiece
- **lineChart** Deriva pubblicamente da Chart e la concretizza definendo i metodi virtuali *draw* e *drawLegend*.
  - **draw** disegna un grafico composto da linee che collegano ogni chartPiece, rappresentato con un punto.
  - **drawLegend** disegna la legenda in un rettangolo sulla parte superiore destra.
- **pieChart** Deriva pubblicamente da Chart e la concretizza definendo i metodi virtuali *draw* e *drawLegend*.
  - **draw** disegna un grafico a torta nel quale ogni chartPiece è rappresentato con uno spicchio.
  - **drawLegend** disegna la legenda composta da dei rettangoli posti circolarmente attorno al grafico che contengono il nome ed il valore associato al relativo dato.
- **histogramm** Deriva pubblicamente da Chart e la concretizza definendo i metodi virtuali *draw* e *drawLegend*.
  - **draw** disegna un istogramma in cui ogni chartPiece è rappresentato da una colonna del grafico.
  - **drawLegend** disegna la legenda in un rettangolo sulla parte superiore destra.
- **myData** si occupa di di gestire i dati inseriti dall'utente memorizzandoli in un $QVector < double >$. Implementa i metodi `loadXML` e `writeXML` che si occupano rispettivamente di caricare e salvare i dati da un file xml selezionato dall'utente
- **myNormalized** Il costruttore di myNormalized riceve un puntatore a myData e popola un $QVector < double >$ con la versione normalizzata dei dati, al fine di poterli rappresentare graficamente.
- **myError** è una classe costruita appositamente per gestire gli errori. Riceve una stringa e la stampa in una *information QMessageBox*.

4 Descrizione della parte grafica

- **myMainWindow** deriva da *QMainWindow* e rappresenta la finestra principale, ad ogni istanza dell'applicazione uno solo oggetto myMainWindow viene allocato. È composta da un *QTabWidget** che permette di lavorare su più fogli, e una *QToolbar* sulla parte superiore che permette la gestione di sheets e grafici.
- **chartSheet** deriva da *QMainWindow* e fornisce il singolo foglio di lavoro, si tratta di una griglia, dove l'utente può inserire i dati che vuole analizzare. È composto da un *QTableWidget** e una *QList* $<$ *canvas*$*$ $>$ che tiene traccia dei singoli grafici creati.
- **Canvas** deriva da *QWidget* e rappresenta la finestra di un singolo grafico. È stato ridefinito il metodo virtuale *paintEvent(QPaintEvent*)* fornito dalla classe basedisegnare gli oggetti di tipo *graph*.

5 Scelte progettuali di maggiore rilevanza

Gestione della memoria

- Per la parte logica la maggior parte degli oggetti, principalmente array, viene distrutta in breve tempo dopo l'allocazione.
- Per la parte grafica i distruttori sono tracciati con QDebug per rendere visibile l'ordine di distruzione degli oggetti.

Gestione degli errori

- Per la parte logica gli errori vengono evitati con controlli preventivi a livello di codice. Generalmente si tratta di verificare che l'ispezione ad un Vector non vada "Out of Range"
- Per la parte grafica tutti i metodi critici, che possono causare errori, sono dotati di una clausola throw che viene catturata solo da una catch del chiamante sul tipo *myError*.

RTTI

- La classe *chart* è polimorfa, dichiara due metodi virtuali puri e il distrutture virtuale. Nel *paintEvent(QPaintEvent*)* della classe *Canvas* ci sono le chiamate polimorfe a **draw** e **drawLegend** sull'oggetto *graph* di tipo *chart*. Viene fatto type checking dinamico sempre all'interno della classe *Canvas* anche nel metodo *setSize()* che in base al tipo di grafico setta la dimensione del canvas.

Operazioni consentite

- La grafica è excel-like, una griglia permetterà all'utente di inserire valori numerici decimali, anche in notazione esponenziale (a.e. 1.e+2). I dati inseriti vengono considerati in ordine top/left - bottom/right all'interno della griglia. Le stringhe non sono valori consentiti e di conseguenza non sono immissibili.
  È possibile eliminare un grafico una volta creato, facendo click destro sul canvas comparirà un QMenu dedicato.
  I documenti creati possono essere aperti e/o salvati attraverso gli appositi bottoni sulla toolbar, si è scelto il formato xml per memorizzarli.