Costruzione di Interfacce
Lezione 17
Qt Tutorial 1 parte 2

cignoni@isti.cnr.it
http://vcg.isti.cnr.it/~cignoni

Tutorial 8

- La prima classe che si ridisegna, 5 file
  - lcdrange.h, lcdrange.cpp
  - cannon.h, cannon.cpp
  - main.cpp
- lcdrange.h e lcdrange.cpp quasi uguali a quelli degli step precedenti
  - aggiunto setRange

```
void LCDRange::setRange( int minVal, int maxVal )
    {
        if ( minVal < 0 || maxVal > 99 || minVal > maxVal ) {
          qWarning( "LCDRange::setRange(%d,%d)\n"
                    "\tRange must be 0..99\n"
                    "\tand minVal must not be greater than maxVal",
                    minVal, maxVal );
         return;
        }
        slider->setRange( minVal, maxVal );
    }
```


Tutorial 8

- notare l'uso di qWarning
  - stampa messaggi di errore e di warning su stderr (o al debugger)
  - simili qDebug e qFatal
  - l'output puo essere gestito in maniera diretta
    - qInstallMsgHandler
  - come funge sotto win?

main.cpp

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qlodnumber.h>
#include <qfont.h>
#include <qlayout.h>

#include "lcdrange.h"
#include "cannon.h"


class MyWidget: public QWidget
{ public:
    MyWidget( QWidget *parent=0, const char *name=0 );
};

int main( int argc, char **argv )
{
    QApplication a( argc, argv );

    MyWidget w;
    w.setGeometry( 100, 100, 500, 355 );
    a.setMainWidget( &w );
    w.show();
    return a.exec();
}
```


main.cpp

```
MyWidget::MyWidget( QWidget *parent, const char *name )  : QWidget( parent, name )
{
    QPushButton *quit = new QPushButton( "Quit", this, "quit" );
    quit->setFont( QFont( "Times", 18, QFont::Bold ) );

    connect( quit, SIGNAL(clicked()), qApp, SLOT(quit()) );

    LCDRange *angle = new LCDRange( this, "angle" );
    angle->setRange( 5, 70 );

    CannonField *cannonField  = new CannonField( this, "cannonField" );

    connect( angle, SIGNAL(valueChanged(int)), cannonField, SLOT(setAngle(int)) );
    connect( cannonField, SIGNAL(angleChanged(int)), angle, SLOT(setValue(int)) );

    QGridLayout *grid = new QGridLayout( this, 2, 2, 10 ); //2x2, 10 pixel border

    grid->addWidget( quit, 0, 0 );
    grid->addWidget( angle, 1, 0, Qt::AlignTop );
    grid->addWidget( cannonField, 1, 1 );
    grid->setColStretch( 1, 10 );

    angle->setValue( 60 );
    angle->setFocus();
}
```


myWidget

- solito subclassing dove si fa una grid e si mette un lcdrange e un CannonField
- Notare:
  - il finto loop tra segnali e slot
    - non e' un loop perche il segnale del cannonfield viene sparato solo se cambia effettivamente il valore…
  - il meccanismo per fare la griglia resizabile solo in parte:
    - grid->setColStretch(1, 10);
    - specifica che la right column (la 1) e' stretchable. mentre the left column no (ha stretch factor 0, il default value);

Cannon.h

```
#ifndef CANNON_H
#define CANNON_H
#include <qwidget.h>

class CannonField : public QWidget
{
    Q_OBJECT
public:
    CannonField( QWidget *parent=0, const char *name=0 );

    int angle() const { return ang; }
    QSizePolicy sizePolicy() const;

public slots:
    void setAngle( int degrees );

signals:
    void angleChanged( int );

protected:
    void paintEvent( QPaintEvent * );

private:
    int ang;
};

#endif // CANNON_H
```


Cannon.h

- la classe deve rappresentare un pezzo di interfaccia dove un cannone 2d spara.
- l'unico dato e' l'angolo del cannon con signal e slot circa la sua modifica
  - segnala quando viene modificato
  - lo slot serve per fare modifiche
- gestione del paint event

protected:
    void paintEvent(QPaintEvent *);

cannon.cpp (1)

```
#include "cannon.h"
#include <qpainter.h>


CannonField::CannonField( QWidget *parent, const char *name )
        : QWidget( parent, name )
{
    ang = 45;
    setPalette( QPalette( QColor( 250, 250, 200) ) );
}


void CannonField::setAngle( int degrees )
{
    if ( degrees < 5 )          degrees = 5;
    if ( degrees > 70 )         degrees = 70;
    if ( ang == degrees )         return;
    ang = degrees;
    repaint();
    emit angleChanged( ang );
}
```


cannon.cpp

- la setAngle emette un segnale
- notare la keyword non c++
  - emit angleChanged(ang);
- ricordatevi che della angleChanged() noi non scriviamo il codice
- Al solito e' il moc a trasformare il codice in c++ puro e a riempire i le parti mancanti…

cannon.cpp (2)

```
void CannonField::paintEvent( QPaintEvent * )
{
    QString s = "Angle = " + QString::number( ang );
    QPainter p( this );
    p.drawText( 200, 200, s );
}


QSizePolicy CannonField::sizePolicy() const
{
   return QSizePolicy( QSizePolicy::Expanding,QSizePolicy::Expanding );
}
```


cannon.cpp

- il repaint e' fatto tramite la classe Qpainter
  - The painter provides highly optimized functions to do most of the drawing GUI programs require.
  - QPainter can draw everything from simple lines to complex shapes like pies and aligned text and pixmaps
  - The typical use of a painter is:
    - Construct a painter.
    - Set a pen, a brush etc.
    - Draw.
    - Destroy the painter.
  - Invocare il painter

Gestire gli eventi di paint

- Si reimplementa in una sottoclasse il metodo `paintEvent`
- `void QWidget::paintEvent ( QPaintEvent * ) [virtual protected]`
- L'oggetto in arrivo contiene info su quali porzioni del widget si deve ridisegnare
  - spesso si ignora…
- Il widget arriva gia (di solito) cleared con il colore di bakground settato

```
setPalette( QPalette( QColor( 250, 250, 200) ) );
```

- avremmo potuto farlo anche con

```
setPaletteBackgroundColor(QColor( 250, 250, 200) );
```

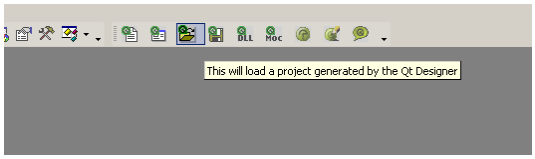

Invocare gli eventi di paint

- 2 modi
- **update**
  - it does not cause an immediate repaint; instead it schedules a paint event for processing when Qt returns to the main event loop
  - Calling update() several times normally results in just one paintEvent() call.
- **repaint**
  - Repaints the widget directly by calling paintEvent() immediately, unless updates are disabled or the widget is hidden.

```
void QWidget::repaint ( const QRect & r, bool erase=TRUE )
```

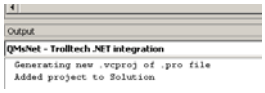

Integrazione sotto .net

- proviamo ad importare il progetto sotto .net e ad abbandonare, per un po', la console
  - aprire il .net
  - controllare che ci sia la toolbar di qt
  - aprire il .pro generato da qmake



.net e qt

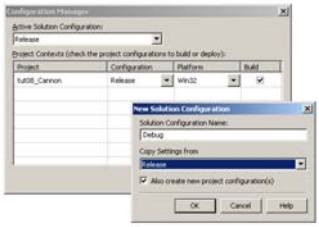
- se tutto va bene (controllate nella finestra dell'output)



- dovrebbe compilare direttamente
- Notare che:
  - fa solo la configurazione di release e non quella di debug

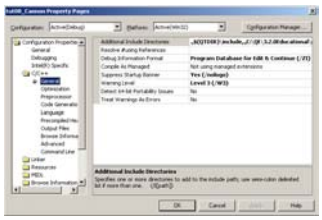
Creiamo la configurazione debug

- 1) far partire il configuration manager e fare una nuova config basata su quella esistente, chiamata debug



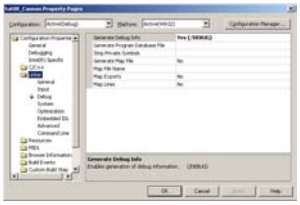
Creiamo la configurazione debug

- 2) Settare le opz del compilatore e del linker per il debug



Creiamo la configurazione debug

- 2) Settare le opz del compilatore e del linker per il debug



Scrivere nella win di debug

- aggiungere al main
- qWarning("Starting the app\n");
- durante il run in debug mode (F5) la stringa appare nella windows di output
- e se volevo fare le scritte che apparivano nella console (stile vecchie printf)?

scrivere nella win di debug

- scrivere un nuovo handler dei messaggi

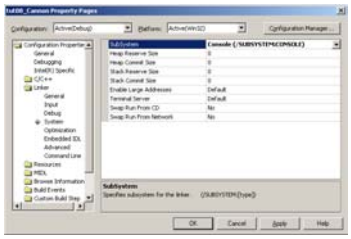
```
#include <stdio.h>
#include <stdlib.h>

void myMessageOutput( QtMsgType type, const char *msg )
{
        switch ( type ) {
           case QtDebugMsg:
               fprintf( stderr, "Debug: %s\n", msg );
               break;
           case QtWarningMsg:
               fprintf( stderr, "Warning: %s\n", msg );
               break;
           case QtFatalMsg:
               fprintf( stderr, "Fatal: %s\n", msg );
               abort();            // deliberately core dump
        }
}
```

- aggiungerlo nel main

```
...
 qInstallMsgHandler( myMessageOutput );
...
```


debug nella console

- e compilare in modalita' console



Cambiamo la paint

```
void CannonField::paintEvent( QPaintEvent * )
{
    QPainter p( this );

    p.setBrush( blue );
    p.setPen( NoPen );

    p.translate( 0, rect().bottom() );
    p.drawPie( QRect(-35, -35, 70, 70), 0, 90*16 );
    p.rotate( -ang );
    p.drawRect( QRect(33, -4, 15, 8) );
}
```

- Sistemi di riferimento in QT
  - Uguali a opengl ☺

Riferimenti

- nell'assistant
  - tutorial #1
  - Signal and slots
  - moc
  - qmake
- il codice prendetelo direttamente dall'assistant