Usare un file *.ui

Il file *.pro di un progetto, contiene, tra le altre cose, anche il nome di un file *.ui, come nel seguente esempio:

app.pro

```
TEMPLATE  = app
FORMS     = form.ui
SOURCES   = main.cpp
```

I file *.ui contengono le informazioni sulla forma dell'interfaccia grafica dei form e possono essere usati:
(1)durante la compilazione, oppure,
(2)durante l'esecuzione di un'applicazione.

(1)Durante la compilazione

Se usato durante la compilazione, il file form.ui serve per generare automaticamente (tramite qmake e uic) un file di intestazione (ui_form.h) che può essere usato in un'applicazione in tre modi diversi:

descrizione	relazione tra classi
A. con approccio diretto	nessuna relazione
B. con approccio ereditarietà singola con variabile membro	aggregazione
C. con approccio ereditarietà singola con variabile dinamica	associazione
D. con approccio ereditarietà multipla	

1-A) Approccio diretto

Il file sorgente main.cpp diventa:

main.cpp

```
#include "ui_form.h"

 int main(int argc, char *argv[])
 {
     QApplication app(argc, argv);
     QWidget* widget = new QWidget;    // ospiterà il form
     Ui::Form ui;                      // crea l'interfaccia ut.
     ui.setupUi(widget);               // inizializzato l'oggetto

     widget->show();
     return app.exec();
 }
```

- ui è un oggetto della classe Form apparttenente al namespace Ui
- widget p un puntatore alla classe QWidget

Non esiste nessuna relazione tra queste due classi (associazione, aggregazione, ecc..), quindi usando questo approccio gli oggetti del form ui **non** si integrano

con il resto del codice...

1-B) Approccio ereditarietà singola con variabile membro

Questa tecnica deriva una nuova classe W dalla classe base QWidget e quindi è necessario un nuovo file sorgente w.cpp

il file app.pro diventa

app.pro

```
TEMPLATE  = app
FORMS     = form.ui
SOURCES   = main.cpp\ w.cpp
HEADERS   = w.h
```

w.h

```
#include "ui_form.h"

 class W : public QWidget
{
     Q_OBJECT

 public:
     W(QWidget *parent = 0); //costruttore

 private:
     Ui::Form ui;            // var. membro
};
```

w.cpp

```
W::W(QWidget* parent):
   QWidget(parent)
{
  ui.setupUi(this);  // prefisso ui.ogniMetodo()
}
```

ui è un oggetto della classe Form, usato dalla classe W, quindi si ha una relazione di composizione.

Come vantaggio si ha l'incapsulamento dei dati dell'interfaccia utente (ui), che si può costruire usando anche più di un form, ognuno nella sua classe.

Nota pratica

Nella realtà, la classe Form e la classe W appartengono a due namespace diversi quindi si potrebbe usare per entrambe lo stesso nome...

Purtroppo Qt Creator fa esattamente in questo modo, anche se questo genera un po' di confusione in chi legge il codice...

1-C) Approccio ereditarietà singola con variabile membro puntatore

Ecco come si modificano i diversi file sorgente

w.h

```
#include <QWidget>

namespace Ui {class Form;} //forward class reference (forw.decl.)

class W:public QWidget
{
  virtual ~W();
  private:
        Ui::Form* ui;
}
```

w.cpp

```
#include "ui_form.h"

W::W(QWidget* parent):
    QWidget(parent), ui(new Ui::Form)
{
  ui->setupUi(this); // prefisso ui->ogniMetodo()
}

W::~W()
{
  delete ui;
}
```

ui è un puntatore ad un oggetto della classe Form, usato dalla classe W, quindi si ha una relazione di associazione, più debole del caso precedente.
In questo caso si può evitare di scrivere #include "ui_form.h" dentro w.h ed è sufficiente la forward declaration: namespace Ui {class Form;}

Per questo motivo non è necessario ricompilare tutto il codice dopo eventuali modifiche fatte esclusivamente alla classe Form dell'interfaccia utente (ui).

Consigliato per grandi applicazioni e per librerie

1-D) Approccio ereditarietà multipla

Nell'ereditarietà multipla la classe W eredita tutti i metodi di QWidget e di Form, quindi non è più necessario creare l'oggetto ui.

Ecco come si modificano i diversi file sorgente

w.h

```
#include <QWidget>

#include "ui_form.h" // scompare la forward declaration

class W: public QWidget, private Ui::Form
{
  Q_OBJECT
  public:
       W(QWidget* parent = 0);
  private slots:
       void on_input_value(int v);
};
```

w.cpp

```
W::W(QWidget* p):
   QWidget(parent)
{
  setupUi(this); // non serve il prefisso ui....
}
```

Nel caso in cui l'interfaccia utente si modifichi in runtime esiste l'evento QEvent::LanguageChange....

http://doc.trolltech.com/main-snapshot/designer-using-a-ui-file.html