



# Introduction à Qt

Gilles Bailly

[gilles.bailly@telecom-paristech.fr](mailto:gilles.bailly@telecom-paristech.fr)

# Crédits

- Eric Lecolinet

# Introduction





# What ?

Librairie graphique écrite en C++ par la société TrollTech.

- Mécanisme pour interagir :
  - avec l'utilisateur (bouton, liste déroulante, ..)
  - avec le système (OpenGL, Xml, SQL, sockets, plugin...)

## Multi-Plateforme

## Gratuit (GPL),

- mais dispose aussi d'une licence commerciale

## Approche :

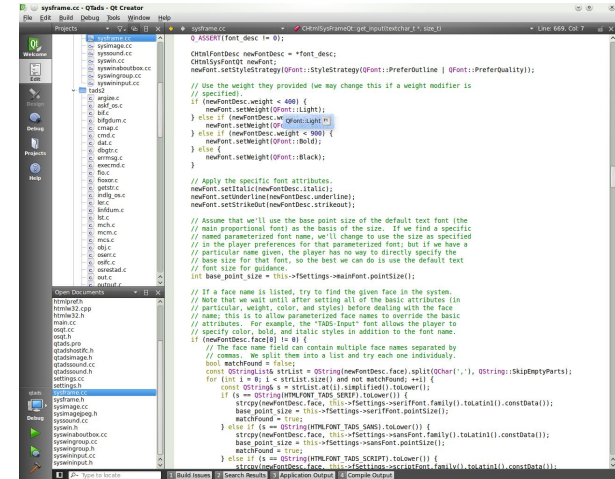
- Ecrire une fois, compiler n'importe où

# Historique

- 1988 : Haavard & Eirik ont l'idée de créer une librairie graphique orientée objet
- 1993 : Le noyau est terminé et ont pour objectif « The world's best C++ GUI framework »
  - Nom Qt, signaux et slots
- 2001 : Qt 3.0, 500 000 lignes de codes, Linux, Windows, Mac.
- 2008 : Qt 4.5 (racheté par Nokia; 250 employés)
  - la plateforme Symbian
- 2009 : Qt 4.6 : animation; QGraphicsScene; machine à état; gestures
- 2011 : Qt est racheté par Digia
  - objectif : Android, iOS et Windows 8
- 2012 : Qt 5.0 : Qt Quick

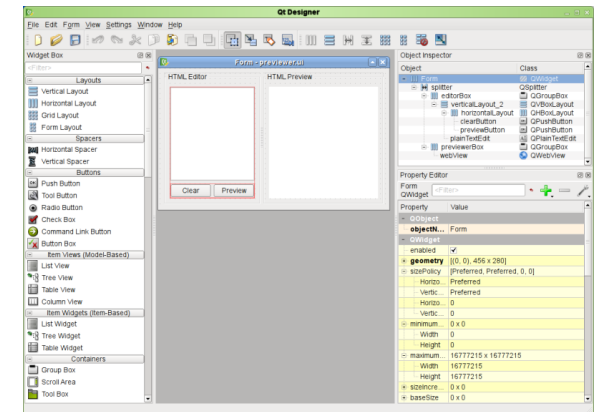
# Why?

- Performance (C++)
- Relativement Simple (proche de Swing)
- Gratuit (GPL) et code source
- Nombreux outils
  - Générateur d'interface : Qt Designer
  - Internationalisation : Qt Linguist
  - Documentation : Qt Assistant
  - Exemples : Qt Examples
  - Programmation : Qt Creator (eclipse)
- Multi-Plateformes
  - Linux, Windows, Mac OS X,
  - Android, iOS,
  - Look and feel simulé (comme swing)



```
classFontDesc newFontDesc = "font_desc";
constQString fontName = "font_desc";
newFontDesc.setFontStyleStrategy(QFont::StyleStrategy(QFont::PreferOutline | QFont::PreferQuality));
newFontDesc.setStyleStrategy(QFont::StyleStrategy(QFont::PreferOutline | QFont::PreferQuality));
// use the weight they provided (we may change this if a weight modifier is
// specified).
if (newFontDesc.weight == 400) {
    newFontDesc.setFontStyle(QFont::Light);
} else if (newFontDesc.weight == 300) {
    newFontDesc.setFontStyle(QFont::Normal);
} else if (newFontDesc.weight == 700) {
    newFontDesc.setFontStyle(QFont::Bold);
} else {
    newFontDesc.setFontStyle(QFont::Black);
}
// Apply the specific font attributes.
newFontDesc.setFontStyle(QFont::Normal);
newFontDesc.setFontStyle(QFont::Normal);
newFontDesc.setFontStyle(QFont::Normal);
newFontDesc.setFontStyle(QFont::Normal);
// Assume that we'll use the base point size of the default text font (the
// name proportional font) on the basis of the size. If we find a specific
// name, parameterized font name, we'll change to use the size as specified
// in the player preferences for that parameterized font; but if we have a
// particular name given, the player has no way to directly specify the
// base size for that font; so the best we can do is use the default text
// font size for guidance.
int base_point_size = this->Settings->mainFont_pointSize();
// If a face name is listed, try to find the given face in the system.
// Note that we wait until after setting all of the basic attributes (in
// particular, weight, color, and styles) before dealing with the face
// name; this is to allow parameterized face names to override the basic
// attributes. For example, the "TADS-SERP" font allows the player to
// specify color, bold, and italic styles in addition to the font name.
// The face name field can contain multiple face names separated by
// commas. We split them into a list and try each one individually.
bool matchFound = false;
const QStringList strList = QStringList(newFontDesc.faceName().split(','));
for (int i = 0; i < strList.size() and not matchFound; ++i) {
    const QStringList strList = QStringList(strList.at(i).split(' '));
    if (strList.size() > 0) {
        const QString fontFaceName = strList.at(0).lower();
        base_point_size = this->Settings->serifFont_pointSize();
        matchFound = true;
    } else if (strList.size() > 0) {
        const QString fontFaceName = strList.at(0).lower();
        base_point_size = this->Settings->sansFont_pointSize();
        matchFound = true;
    } else if (strList.size() > 0) {
        const QString fontFaceName = strList.at(0).lower();
        base_point_size = this->Settings->serifFont_pointSize();
        matchFound = true;
    } else if (strList.size() > 0) {
        const QString fontFaceName = strList.at(0).lower();
        base_point_size = this->Settings->sansFont_pointSize();
        matchFound = true;
    } else if (strList.size() > 0) {
        const QString fontFaceName = strList.at(0).lower();
        base_point_size = this->Settings->serifFont_pointSize();
        matchFound = true;
    } else if (strList.size() > 0) {
        const QString fontFaceName = strList.at(0).lower();
        base_point_size = this->Settings->sansFont_pointSize();
        matchFound = true;
    }
}
```

Qt creator



Designer

# Aujourd'hui

Augmentation de 250% des téléchargements de la GPL

Qt Downloads from qt.nokia.com



Qt 5 : 10 000 téléchargement par jour



# Aujourd'hui

- Utilisateurs de Qt :
  - Nokia, Nasa, Adobe, Motorola, Google, ...
- Bindings (java, python, c#)



**Qt in Automotive  
Infotainment**



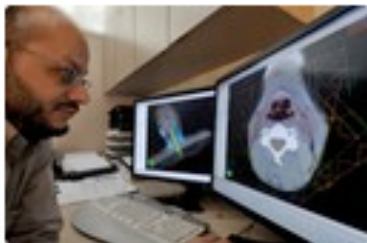
**Qt in Aerospace**



**Qt in Home Media**



**Qt in IP Communication**



**Qt in Medical**



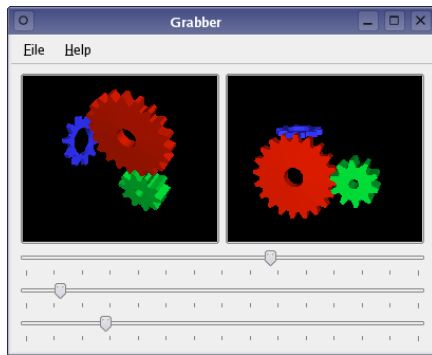
**Qt in Oil & Gas**



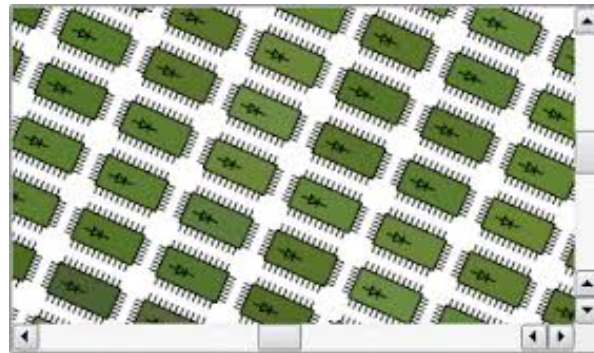
**Qt in Visual Effects**



# Aujourd'hui



Qt Widgets

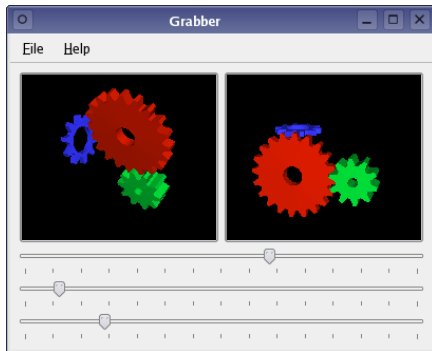


Qt Graphics view



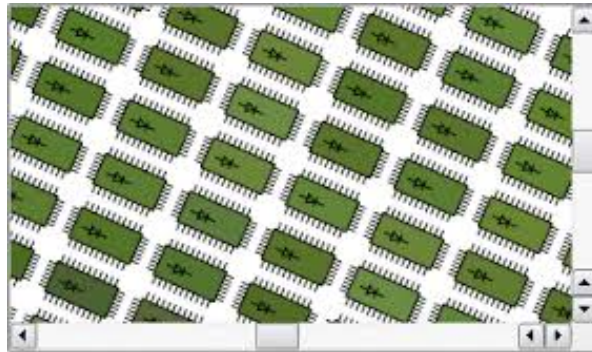
Qt quick

# Aujourd'hui



**Qt Widgets**

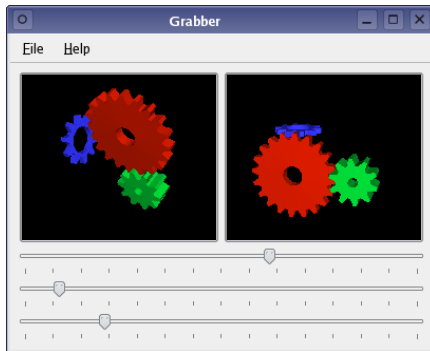
V.S.



**Qt Graphics view**

- Widgets *cannot* be scaled or rotated,
- Widgets can only appear once on the desktop, but several views can observe one graphicsitem.
- Widgets express their geometries in pixels, graphics items use logical units. (...int vs. double)
- Widgets support tons of features that graphics items don't understand.
- Widgets understand layouts,
- 4000000 widgets don't work that well, but **4000000 items works perfectly.** 10

# Aujourd'hui



Qt Widgets

V.S.



Qt quick

```
import QtQuick 2.0
```

```
Rectangle {  
    id: canvas  
    width: 200  
    height: 200  
    color: "blue"
```

```
Image {  
    id: logo  
    source: "pics/logo.png"  
    anchors.centerIn: parent  
    x: canvas.height / 5  
}  
}
```

Language déclaratif

- Widgets are more mature, flexible and provide rich features
- Qt Quick focuses on animation and transition
- Qt Quick is mainly for mobile devices (today)
- Qt Quick will replace Widgets tomorrow
- Qt Quick is maybe easier to use for designers

# Objectifs

## Introduction

- **Signaux et slots**
- Les principales **classes** Qt
- **Compiler** une application Qt

## Graphisme avancé

- **Création** de vos propres widgets
- Programmation **événementielle**
- Notion avancée de **graphisme** avec Qt

## Quelques autres notions avancées

- Machine à états
- Animation
- Qt Designer

Au delà de Qt ...

# Mes premiers pas avec Qt

"Hello world"



# Hello Word !!!

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[]) {
    QApplication *app = new QApplication(argc, argv);
    QPushButton *hello = new QPushButton( "Hello world!" );

    hello->resize(100, 30);
    hello->show();

    return app->exec();
}
```



Définition des  
classes

Point d'entrée  
Du programme

Redimensionnement  
du bouton  
Affiche le bouton

Creation de  
L'application  
Creation du  
bouton

Passe le contrôle  
À QCoreApplication

- Remarques

- Attention aux -> et \*
- Les pointeurs ne sont pas des références

## Java [\[modifier\]](#)

---

```
import com.trolltech.qt.gui.QApplication;
import com.trolltech.qt.gui.QPushButton;

public class HelloWorld
{
    public static void main(String args[])
    {
        QApplication.initialize(args);
        QPushButton hello = new QPushButton("Hello World!");
        hello.show();
        QApplication.exec();
    }
}
```

## Python [\[modifier\]](#)

---

```
from PyQt4 import QtGui, QtCore
import sys

app = QtGui.QApplication(sys.argv)
hello = QtGui.QPushButton("Hello World!", None)
hello.show()
app.exec_()
```

# Hello Word !!!

```
#include <QApplication>
#include <QPushButton>
#include <QWidget>
```

```
int main(int argc, char *argv[]) {
```

```
    QApplication *app = new QApplication(argc, argv);
```

```
    QWidget *box = new QWidget();
```

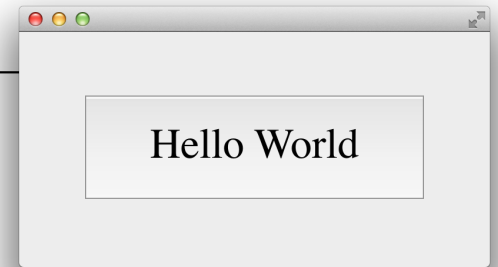
```
    QPushButton *hello = new QPushButton( "Hello world!", box );
```

```
    hello->resize(100, 30);
```

```
    box->show();
```

```
    return app->exec();
```

```
}
```



Conteneur

Parent



# Un peu plus loin...

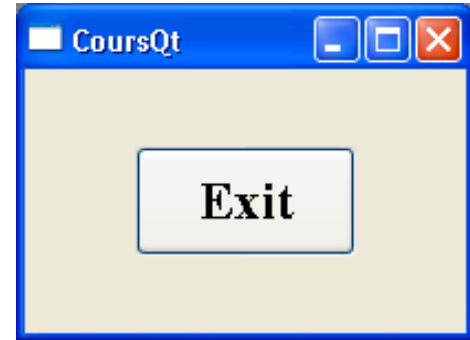
```
#include <QApplication>
#include <QPushButton>
#include <QWidget>
#include <QFont>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget box;
    box.resize( 200, 120 );

    QPushButton exitButton("Exit", &box);
    exitButton.resize( 100, 50 );
    exitButton.move( 50, 35 );
    exitButton.setFont( QFont( "Times", 18, QFont::Bold) );
    box.show();

    QObject::connect(&exitButton, SIGNAL(clicked()), &app, SLOT(quit()));

    return app.exec();
}
```



Quand je clic sur  
le bouton « Exit »  
Je quitte  
l'application

# Signaux et Slots



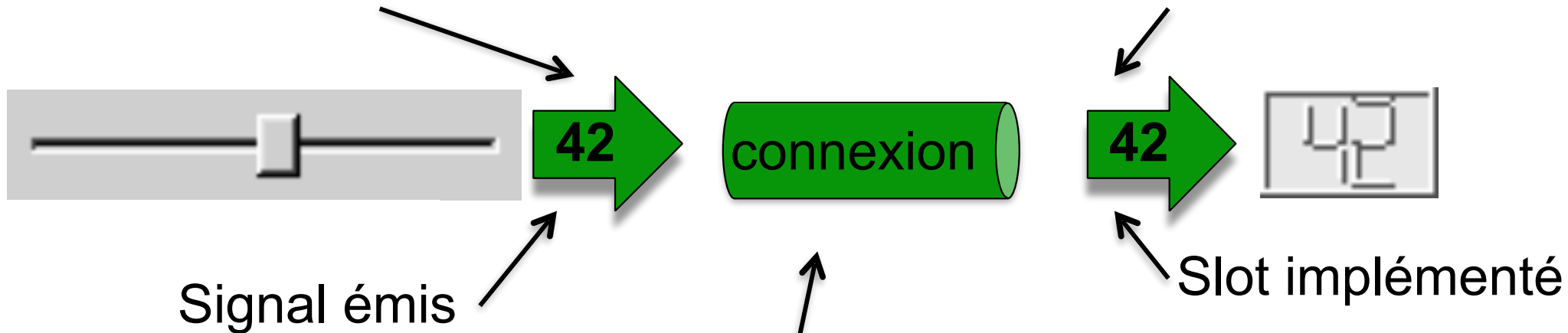
# Problématique

- Comment
  - à partir d'un « clic sur un bouton »
  - Je peux exécuter la partie correspondant à la logique de mon application ?
- Solutions
  - MFC (introduit un langage au dessus de C++)
  - Java (utilise des listeners)
  - **Qt (utilise principalement des signaux et slots)**

# Connecter Signaux et Slots

```
void Qslider::mouseMoveEvent(...)  
{  
...  
emit valueChanged( newValue);  
}
```

```
void QLCDNumber::display(int _num)  
{  
...  
m_value = num;  
}
```

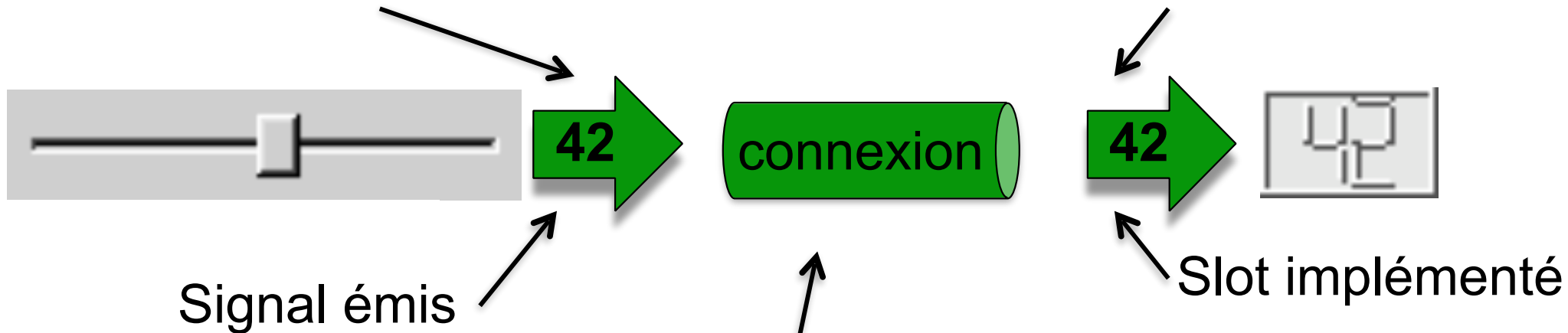


```
QObject::connect(slider, SIGNAL( valueChanged( int ) ),  
                 lcdNumber, SLOT( display( int ) ) );
```

# Connecter Signaux et Slots

```
void Qslider::mousePressEvent(...)  
{  
...  
emit valueChanged( newValue)  
}
```

```
void QLCDNumber: display(int _num)  
{  
...  
m_value = num;  
}
```



```
QObject::connect(slider, SIGNAL( valueChanged( int ) ),  
IcdNumber, SLOT( display( int ) ) );
```

# Une classe avec des signaux et des slots

*Ce n'est pas du C++*

Dans le fichier header (.h, .hh)

```
class MyClass : public QObject{
    Q_OBJECT
public:
    ...

signals:
void mySignal( int );

public slots:
void mySlot( int );

};
```

- Sous class de **QObject**
- 
- Mot-clés :
  - **Q\_OBJECT**,
  - **signals**,
  - **slots**
- Les **signaux** sont pas implémentés
- Les **slots** doivent être implémentés

# Signaux et Slots

- **Modularité, flexibilité**

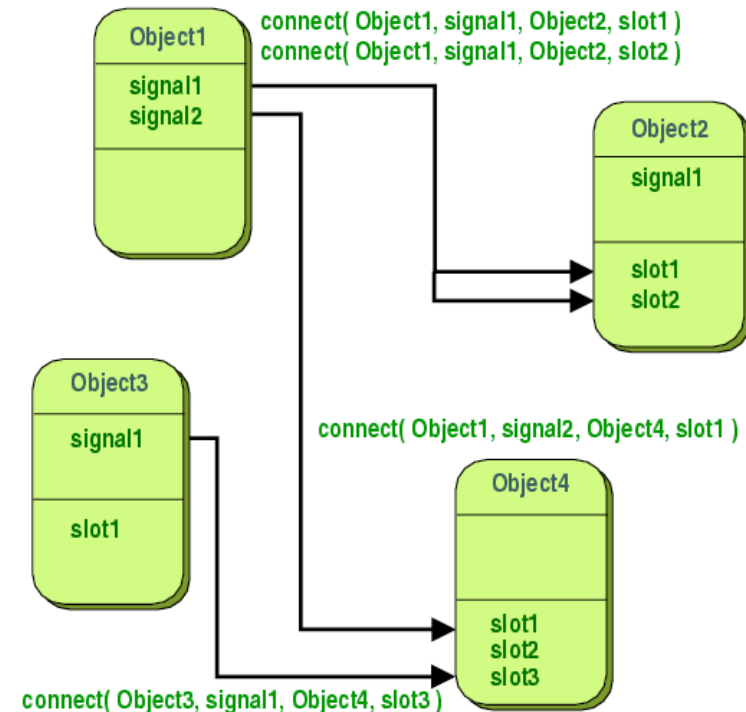
- Connecter **plusieurs** signaux à **un** slot
- Connecter **un** signal à **plusieurs** slots

- **Philosophie**

- L'émetteur n'a pas besoin de connaître le(s) récepteur(s)
- L'émetteur ne sait pas si le signal a été reçu
- Le récepteur ne connaît pas non plus l'émetteur
- Programmation par composant (indépendant, réutilisable)

- **Sécurité, typage fort**

- Les **types** des paramètres doivent être les **mêmes**
- Un slot peut avoir **moins** de paramètres qu'un signal



```
QObject::connect( &quitBtn, SIGNAL(clicked( )), &app, SLOT(quit( )) );
```

```
QObject::connect( &x, SIGNAL(balanceChanged(int)), &y, SLOT(setBalance(int)) );
```

```
QObject::connect( &x, SIGNAL(balanceChanged(int)), &app, SLOT(quit()) );
```

- **Sécurité, typage fort**

- Les **types** des paramètres doivent être les **mêmes**
- Un slot peut avoir **moins** de paramètres qu'un signal



# Exemple: Bank (.h)

transfert d'argent entre deux banques

---

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int curBalance;

public:
    BankAccount( ) { curBalance = 0; }
    int getBalance( ) const { return curBalance; }

public slots:
    void setBalance( int newBalance );

signals:
    void balanceChanged( int newBalance );
};
```

# Exemple: Bank (.cpp)

transfert d'argent entre deux banques

---

```
void BankAccount::setBalance(int newBalance)
{
    curBalance = newBalance;
    emit balanceChanged(curBalance);
}
```

## Problème?

- x <- 25
- x emit balanceChanged(25)
- y <- 25
- y emit balanceChanged(25)
- etc.

---

```
BankAccount x, y;
```

```
connect( &x, SIGNAL(balanceChanged(int)), &y, SLOT(setBalance(int)) );
```

```
connect( &y, SIGNAL(balanceChanged(int)), &x, SLOT(setBalance(int)) );
```

```
x.setBalance( 2450 );
```



# Retour sur les signaux et les slots

---

**Comment différencier des actions dans un même slot ?**

# Solution 1: QObject::sender()

---

// Dans le .h en variables d'instance de **MaClasse** :

```
QObject * action1, * action2, ...;
```

// Dans le .cpp:

```
void MaClasse::createGUI() {
```

```
    action1 = new QObject(tr("Action 1"), this);
```

```
    connect(action1, SIGNAL(triggered()), this, SLOT(dolt()));
```

```
    action2 = new QObject(tr("Action 2"), this);
```

```
    connect(action2, SIGNAL(triggered()), this, SLOT(dolt()));
```

```
    ...
```

```
}
```

```
void MaClasse::dolt() {
```

```
    QObject * sender = QObject::sender();
```

```
    if (sender == action1) ....;
```

```
    else if (sender == action2) .... ;
```

```
    ....
```

```
}
```

# Solution 2: QActionGroup

---

// Dans le .h en variables d'instance de **MaClasse** :

```
QAction * action1, * action2, ...;
```

// Dans le .cpp:

```
void MaClasse::createGUI( ) {
```

```
    QActionGroup *group = new QActionGroup(this);
```

```
    connect(group, SIGNAL(triggered(QAction*)), this, SLOT(dolt(QAction*))); // un seul connect !
```

```
    action1 = group->addAction(tr("Action 1"));
```

```
    action2 = group->addAction(tr("Action 2"));
```

```
    ...
```

```
}
```

```
void MaClasse::dolt(QAction* sender ) { // l'action est récupérée via le paramètre
```

```
    if (sender == action1) ....;
```

```
    else if (sender == action2) .... ;
```

```
    ....
```

```
}
```

# Solution 2: QActionGroup

---

Par défaut le groupe est exclusif, sinon faire :

```
QActionGroup *group = new QActionGroup(this);  
group->setExclusive(false);
```

On peut faire de même pour les boutons (QPushButton, QRadioButton, QCheckBox ...):

```
QButtonGroup *group = new QButtonGroup(this);
```

En utilisant le signal :

```
buttonClicked ( QAbstractButton * button )
```

ou :

```
buttonClicked ( int id )
```

# Solution 3: QSignalMapper

---

```
void MaClasse::createGUI() {
    QSignalMapper* mapper = new QSignalMapper (this) ;
    connect(mapper, SIGNAL(mapped(int)), this, SLOT(dolt(int))) ;

    QPushButton * btn1 = new QPushButton("Action 1", this);
    connect (btn1, SIGNAL(triggered( )), mapper, SLOT(map( )) ;
    mapper -> setMapping (btn1, 1) ;

    QPushButton * btn2 = new QPushButton("Action 1", this);
    connect (btn2, SIGNAL(triggered( )), mapper, SLOT(map( )) ;
    mapper -> setMapping (btn2, 2) ;
    ...
}

void MaClasse::dolt(int value) {           // l'action est récupérée via le paramètre
    ....
}
```

Possible pour les types : **int**, **QString&**, **QWidget\*** et **QObject\***



# Conclusion

- Aspect central de Qt
- Diffère de l'habituel mécanisme des Listeners (java)
- Avantages/inconvénients
  - Slot et Signal sont des MACROS
  - Nécessite une phase de pré-compilation
  - Modulaire

# Conclusion

- Aspect central de Qt
- Diffère de l'habituel mécanisme des Listeners (java)
- Avantages/inconvénients
  - Slot et Signal sont des **MACROS**
  - **Nécessite une phase de précompilation**
  - Modulaire

# Questions

- Comment connecter un signal à un slot ?
  - `QObject::connect()`
- Quel code pour déclarer / implémenter un slot ?
  - `public slots: (.h)`
- Est ce qu'un slot peut retourner une valeur ?
  - Oui
- Quel code pour déclarer / implémenter un signal ?
  - `signals (.h)`
  - `emit (.cpp)`
- Où et Quand j'utilise `Q_OBJECT` ?
  - Première élément dans la class (.h)
  - signals et/ou slots

# Compilation & QMake



# Une classe avec des signaux et des slots

*Ce n'est pas du C++*

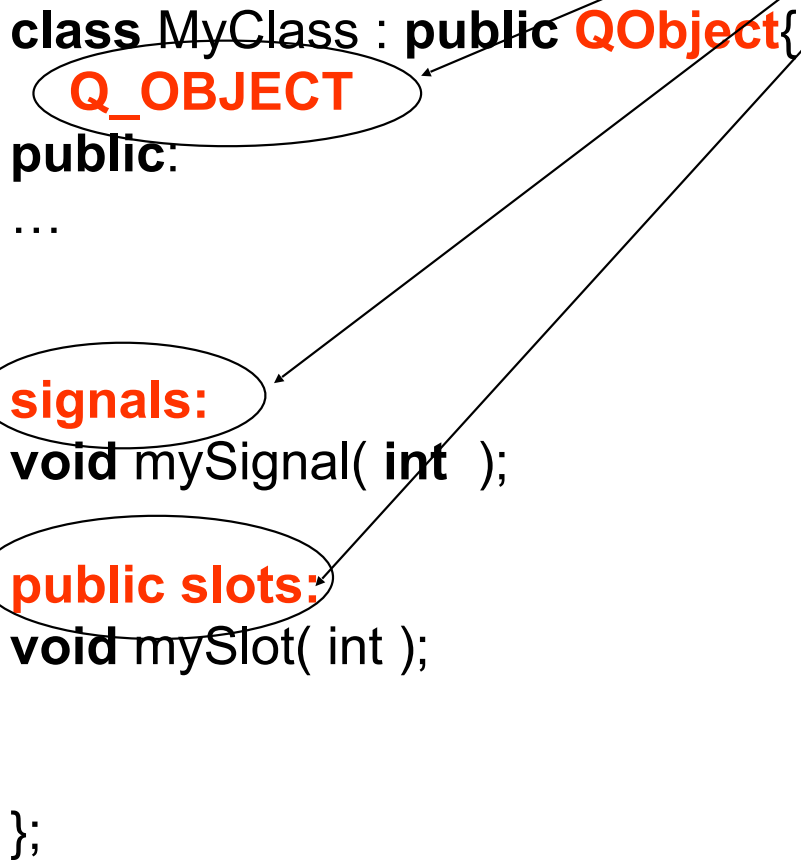
Dans le fichier header (.h, .hh)

```
class MyClass : public QObject{
    Q_OBJECT
public:
    ...

signals:
    void mySignal( int );

public slots:
    void mySlot( int );

};
```



# Comment compiler si ce n'est pas du C++ ?




Meta Object Compiler (Moc)

- Pré-processeur C++
  - `Moc -o moc_myClass.cpp myClass.h`
- Génère du code supplémentaire
  - Tables de signaux/slots
- Permet aussi des informations sur méta-information sur la classe courante (nom de la classe, test d'héritage, ...)
- **Attention**: ne pas oublier le mot clé **Q\_OBJECT**

**Pénible ?**

# QMake

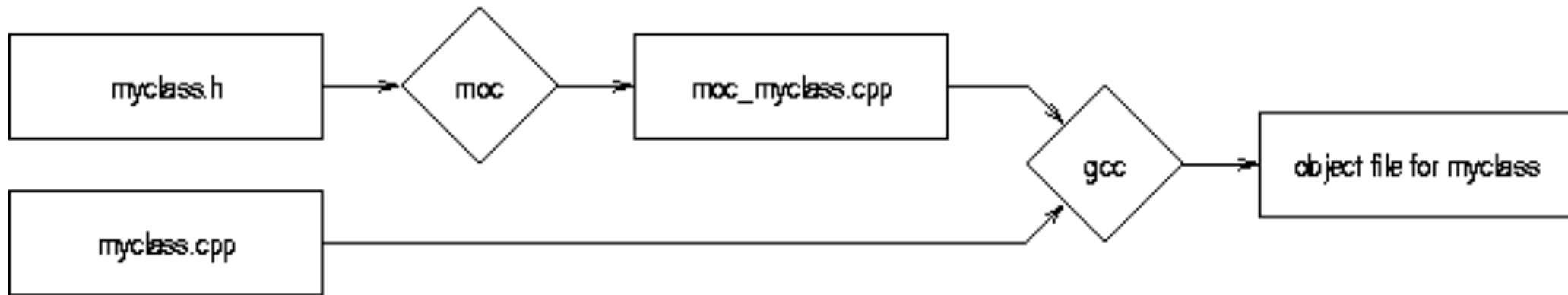
## Utilisation de QMake

- 1) qmake – project  Crée le fichier nomRepertoire.pro
- 2) qmake  Crée le fichier makefile  
Crée les fichiers moc\_\*
- 3) make  .o et executable

## Multi-plateforme

- Nouvelle plateforme : qmake pour recréer le makefile

# QMake





# Questions/Réponses (2/2)

- Quand est ce que je fais make ?
  - à chaque changement dans le code
- Quand est ce que je fais qmake ?
  - à chaque changement concernant les signaux et les slots
- Quand est ce que je fais qmake -project
  - quand je rajoute / supprime un fichier

# Les principaux widgets



# Modules

- **QtCore**
- **QtWidgets**
- QtBluetooth
- QtOpenGL
- QtSript/QtScriptTools
- QtSql
- QtSvg
- QtWebKit
- QtXml/QtXmlPatterns
- QtMultimedia
- QtSensors

# QtCore

- QObject
- Type de base : QChar, QDate, QString, **QStringList**, QTime, ...
- File systems : QDir, QFile, ...
- Container : QList, QMap, QPair, QSet, QVector, ...
- Graphique : QLine, QPoint, QRect, QSize ...
- Thread : QThread, QMutex, QSemaphore, ...
- Autres : QTimer, **QTimeline**, ...

# QString

---

## Codage Unicode 16 bits

- Suite de **QChars**
  - 1 caractère = 1 **QChar** de 16 bits (cas usuel)
  - 1 caractère = 2 **QChars** de 16 bits (pour valeurs > 65535)
- **Conversions** d'une **QString** :
  - **toAscii()** : ASCII 8 bits
  - **toLatin1()** : Latin-1 (ISO 8859-1) 8 bits
  - **toUtf8()** : UTF-8 Unicode multibyte (1 caractère = 1 à 4 octets)
  - **toLocal8Bit()** : codage local 8 bits
- **qPrintable** ( const QString & str )
  - équivalent à : `str.toLocal8Bit().constData()`

# QFile

---

## QFile

- lecture, écriture de fichiers
- exemples :
  - **QFile** file( fileName );
  - if ( file.open( QIODevice::ReadOnly | QIODevice::Text ) ...;
  - if ( file.open( QIODevice::WriteOnly ) ) ...;

# QTextStream

---

## QTextStream

- lecture ou écriture de **texte** depuis un **QFile** :
  - **QTextStream** stream( &file );
- Améliorent les **iostream** du C++
  - compatibles avec **QString** et **Unicode** (et d'autres codecs de caractères)

# QTextStream

---

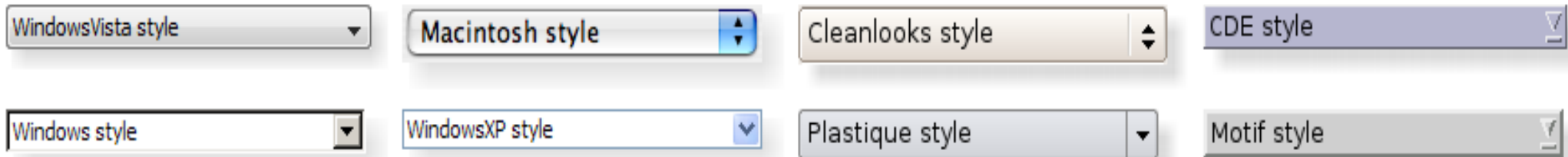
## QTextStream

- lecture ou écriture de **texte** depuis un **QFile** :
  - **QTextStream** stream( &file );
- opérateurs **<<** et **>>** :
  - outputStream **<<** string;
  - inputStream **>>** string; *// attention : s'arrête au premier espace !*
- méthodes utiles :
  - QString **readLine**( taillemax = 0 ); *// pas de limite de taille si = 0*
  - QString **readAll**( ); *// pratique mais à n'utiliser que pour des petits fichiers*
- codecs :
  - **setCodec**( codec ), **setAutoDetectUnicode**( bool );



# QtGUI

# QStyle



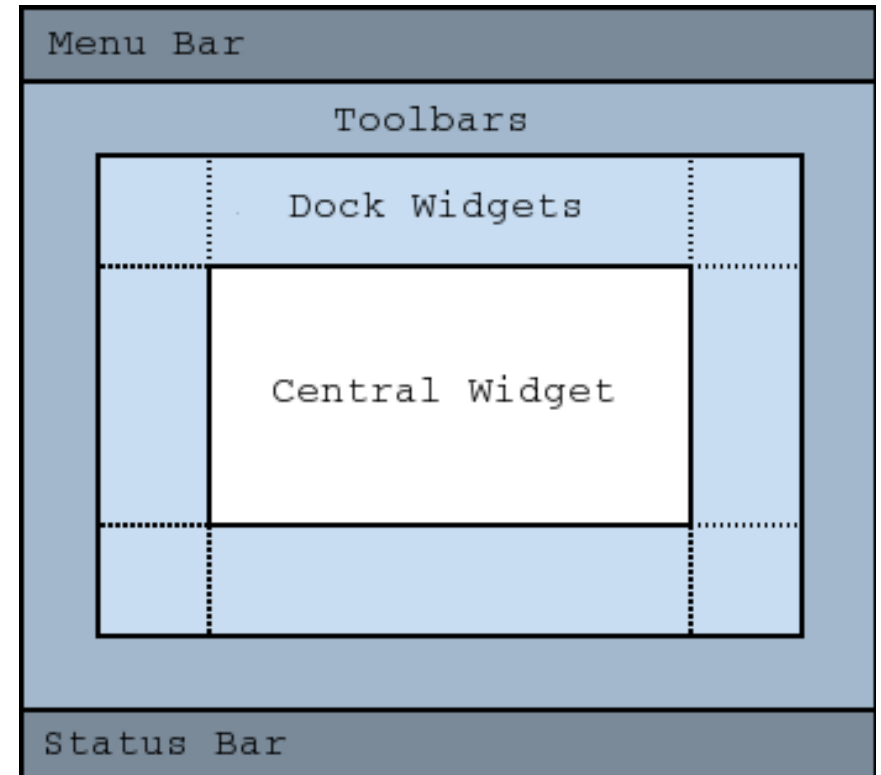
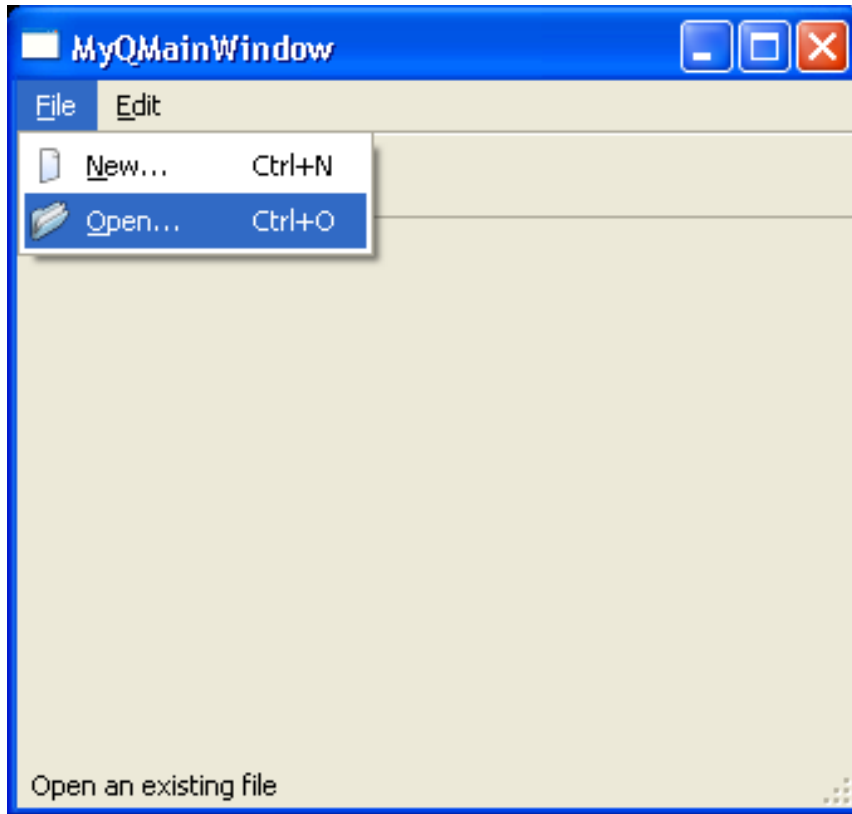
- **int main(int argc, char \*argv[])**  
    **QApplication app( argc, argv);**

—————→ **./monAppli -style plastique**

Ex: windows, motif, platinum,...

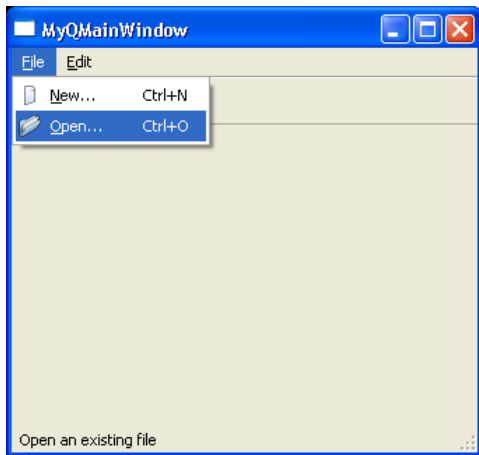
**QApplication::setStyle( new QPlastiqueStyle() );**

# QMainWindow

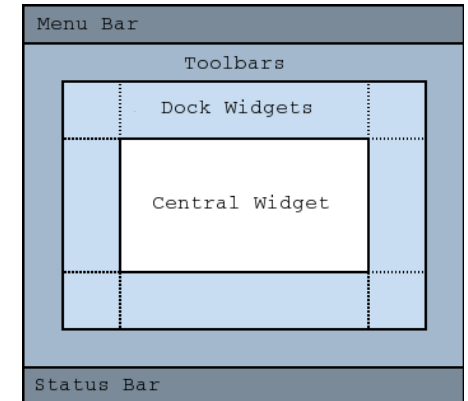


## Utilisation:

- créer une sous-classe de **QMainWindow**
- dont le constructeur crée / ajoute des objets graphiques à cette fenêtre

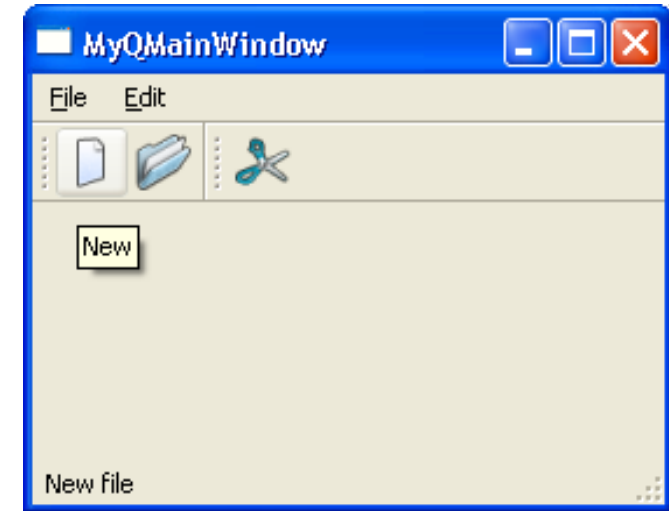


# QMainWindow : Menus



- **QMenuBar\*** myMenuBar = **menuBar()**;
- **QMenu\*** fileMenu = myMenuBar->**addMenu**( *tr*("&File") );
- **QAction\*** newAct = **new QAction**(**QIcon**(":/images/new.png"), *tr*("&New..."), **this**);
- newAct->**setShortcut**(*tr*("Ctrl+N"));
- newAct->**setToolTip**(*tr*("New File"));
- newAct->**setStatusTip**(*tr*("New file"));
- fileMenu->**addAction**(newAct);
- **connect**(openAct, **SIGNAL**(triggered()), **this**, **SLOT**(open()));

# QMainWindow



- *QMenuBar, QMenu, QAction*

- **QToolBar**

- `QToolBar* fileToolBar = addToolBar(tr("File"));`
- `fileToolBar->addAction(newAct);` ←

- `newAct->setEnabled( false)` ←

Grise la commande  
Dans la barre des  
Menus et toolbar

- **QToolTip, QWhatsThis**

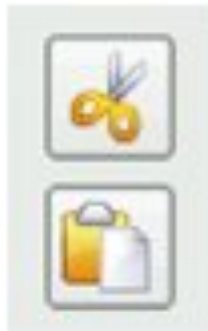
- **Composant central**

```
QTextEdit* textEdit = new QTextEdit( this );  
setCentralWidget( textEdit );
```

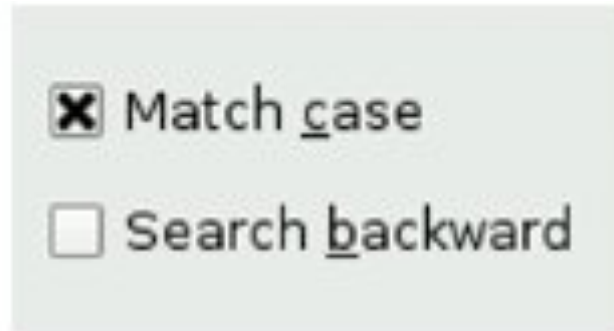
# Buttons



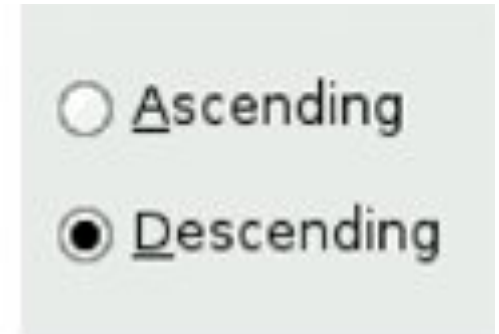
QPushButton



QToolButton



QCheckBox



QRadioButton

# Input Widgets



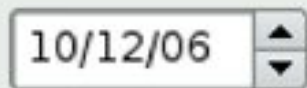
QSpinBox



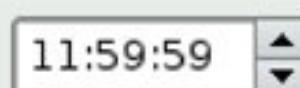
QDoubleSpinBox



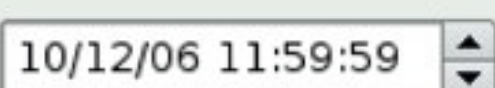
QComboBox



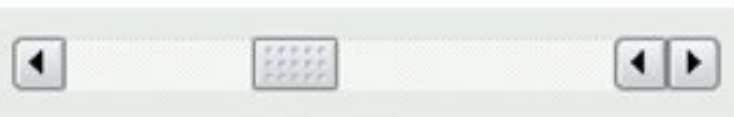
QDateEdit



QTimeEdit



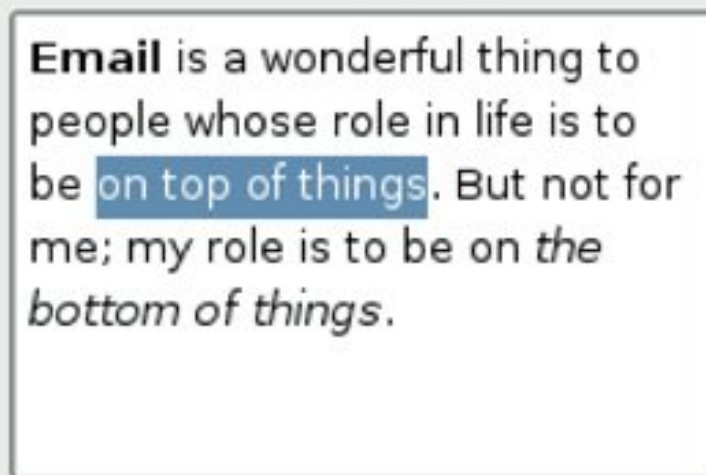
QDateTimeEdit



QScrollBar



QSlider



QTextEdit



QLineEdit

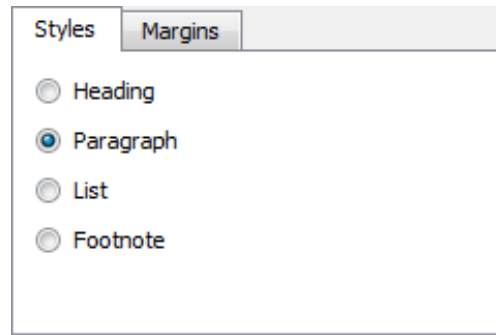


QDial

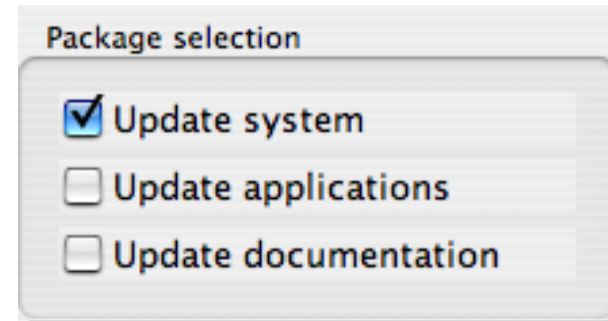
# Containers



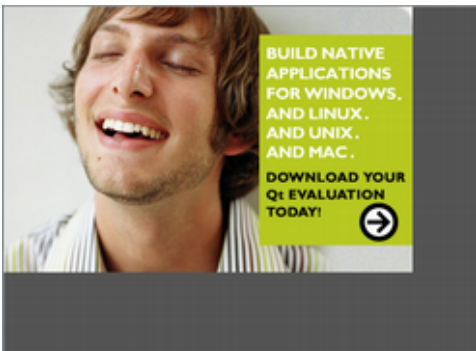
**QMidArea**



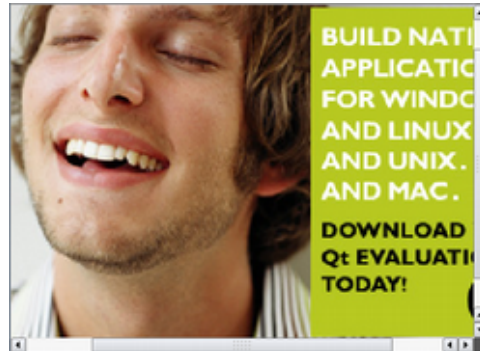
**QTabWidget**



**QGroupBox**



**QScrollArea**



**QToolBox**

**QWidget; QFrame; QDockWidget; QStackedWidget**



# Views



QListView (as list)



QTreeView



QListView (as icons)

A screenshot of a QTableView widget showing a table with three columns (A, B, C) and four rows (1-4). The table has a scrollbar on the right and a header with 'A', 'B', and 'C'. The data in the table is as follows:

	A	B	C
1	1043.23	250	
2	1037.39	178	
3	970.77		
4	1008.32		

QTableView

# Display Widgets

**Warning:** All unsaved information will be lost!

QLabel (text)



QLCDNumber



QProgressBar

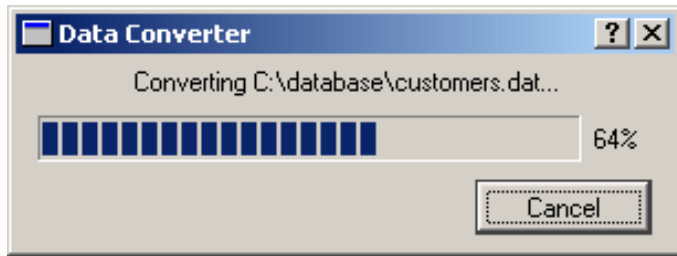


QLabel (image)

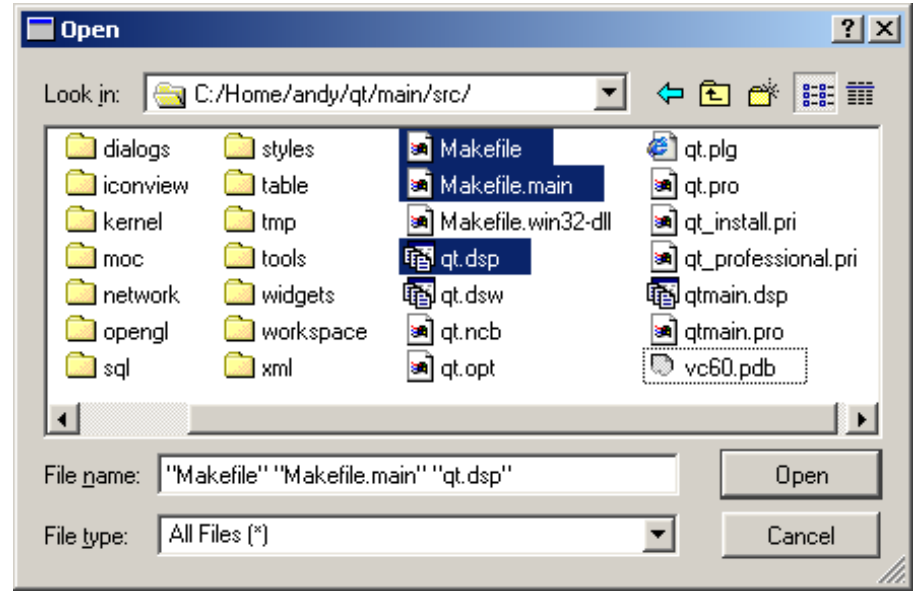


QTextBrowser

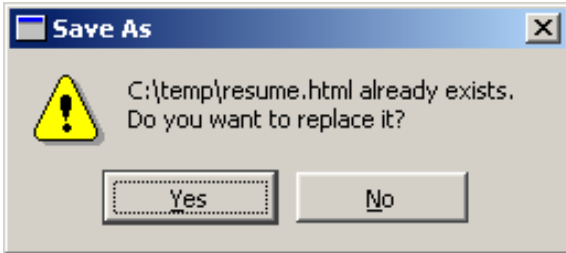
# Boite de dialogue



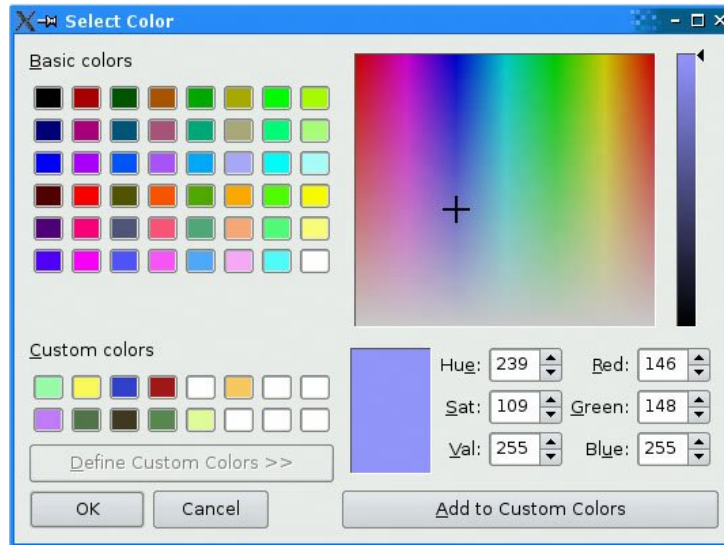
**QProgressDialog**



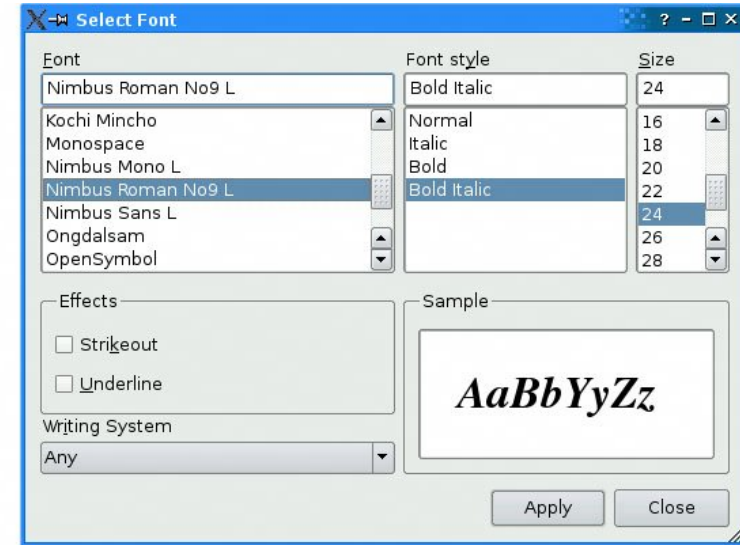
**QFileDialog**



**QMessageBox**



**QColorDialog**



**QFontDialog**

# Boîte de dialogue modale

## Solution générale

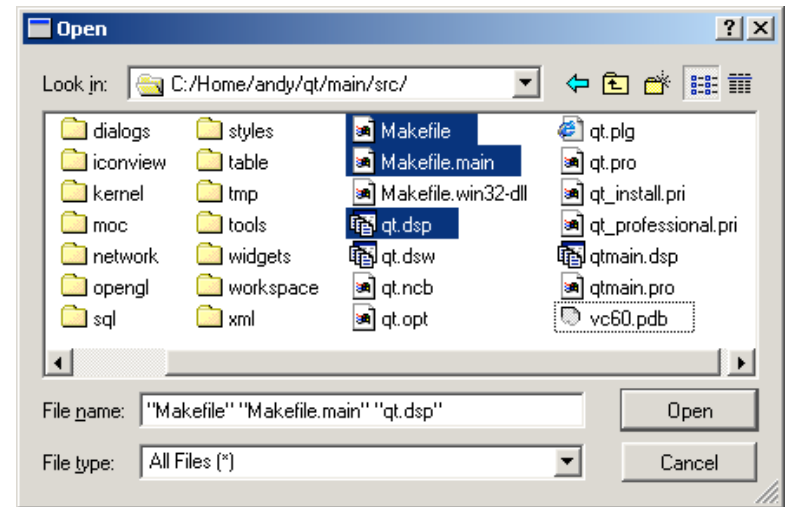
```
QFileDialog dialog (parent);
dialog.setFilter("Text files (*.txt)");
QStringList fileNames;

if (dialog.exec() == QDialog::Accepted) {
    fileNames = dialog.selectedFiles();
    QString firstName = fileNames[0];
    ...
}
```

## Solution simplifiée

```
QString fileName =
    QFileDialog::getOpenFileName( this,
                                tr("Open Image"),
                                "/home/jana",
                                tr("Image Files (*.png *.jpg *.bmp)")
                                );
```

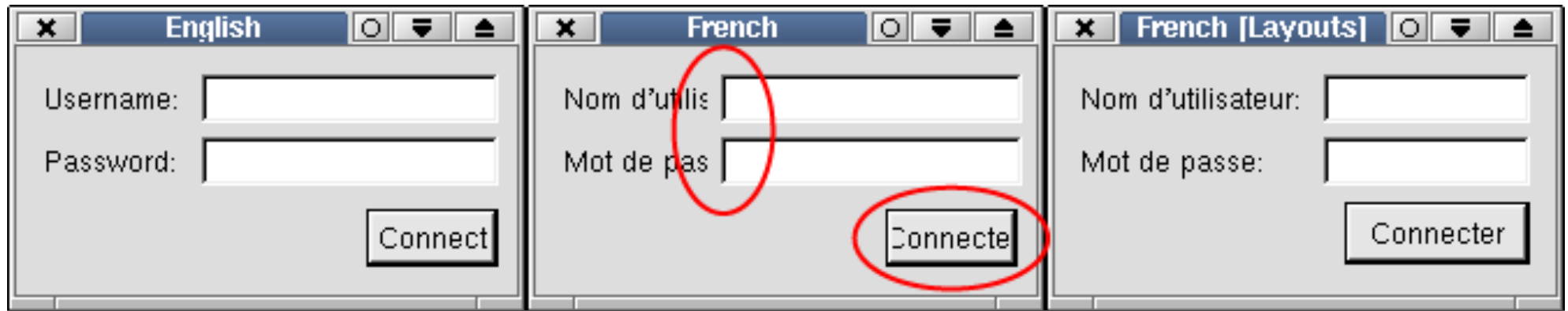
// titre  
// répertoire initial  
// filtre



### Note :

**dialog.exec()** lance une boucle de gestion des événements secondaire !

# Layout



## Problèmes

- internationalisation
- redimensionnement
- complexité du code

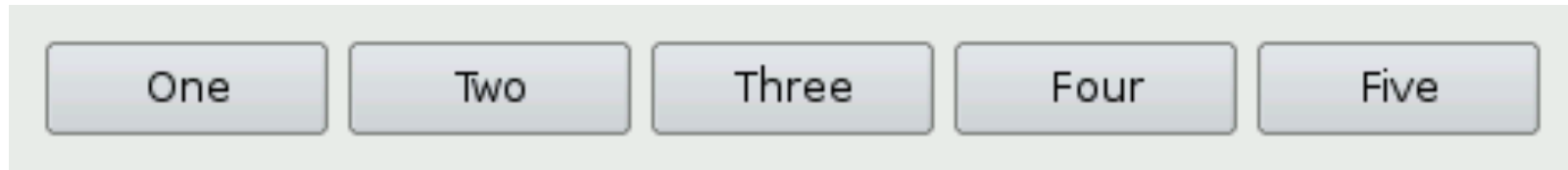
# Layout




A screenshot of a Qt window titled "Windows" with a blue title bar. The window contains a form with three input fields arranged vertically. The first field is labeled "Name:" and contains the text "Gandalf". The second field is labeled "Email address:" and contains the text "gg@troll.no". The third field is labeled "Age:" and contains the text "4000". The "Age" field has a small spinner control on its right side.

**QFormLayout**

**QHBoxLayout**



A screenshot of a Qt widget with a light gray background. It displays five buttons arranged horizontally in a single row. The buttons are labeled "One", "Two", "Three", "Four", and "Five" from left to right.



A screenshot of a Qt widget with a light gray background. It displays five buttons arranged vertically in a single column. The buttons are labeled "One", "Two", "Three", "Four", and "Five" from top to bottom.

**QVBoxLayout**



A screenshot of a Qt widget with a light gray background. It displays five buttons arranged in a grid. The first row contains two buttons labeled "One" and "Two". The second row contains a single wide button labeled "Three". The third row contains two buttons labeled "Four" and "Five".

**QGridLayout**

# Layout : exemple

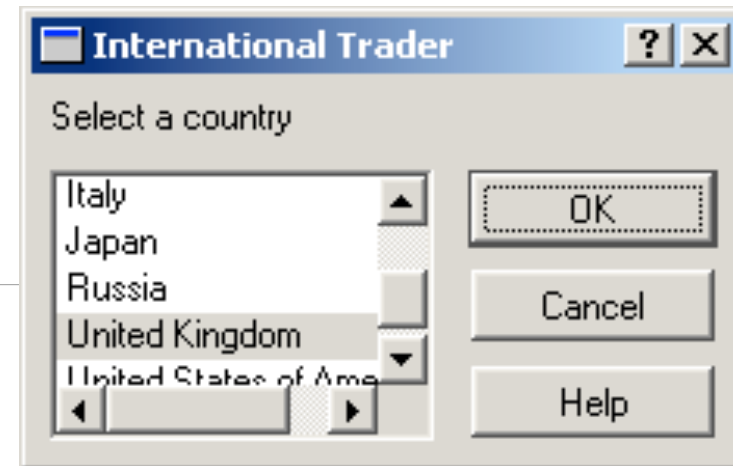
```
QVBoxLayout * v_layout = new QVBoxLayout();  
v_layout->addWidget( new QPushButton( "OK" ) );  
v_layout->addWidget( new QPushButton( "Cancel" ) );  
v_layout->addStretch();  
v_layout->addWidget( new QPushButton( "Help" ) );
```

```
QListBox * country_list = new QListBox( this );  
countryList->insertItem( "Canada" );  
...etc...
```

```
QHBoxLayout * h_layout = new QHBoxLayout();  
h_layout->addWidget( country_list );  
h_layout->addLayout( v_layout );
```

```
QVBoxLayout * top_layout = new QVBoxLayout();  
top_layout->addWidget( new QLabel( "Select a country", this ) );  
top_layout->addLayout( h_layout );
```

```
window->setLayout( top_layout );  
window->show();
```



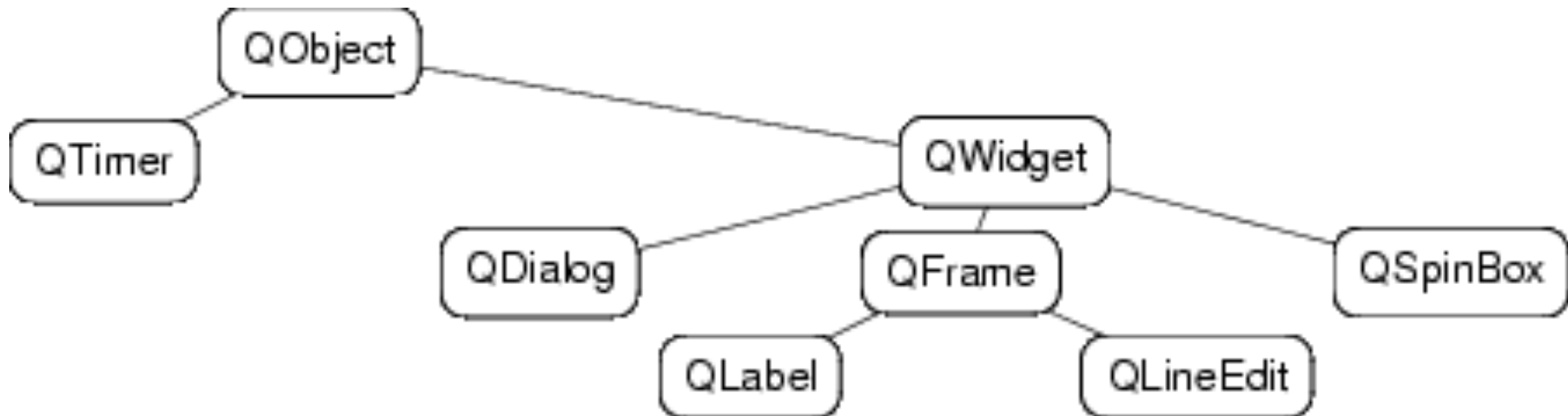
## Notes sur layouts :

- peuvent être emboîtés
- pas liés à une hiérarchie de conteneurs comme Java
- cf. le « stretch »

**Arbre d'héritage vs.  
arbre d'instanciation**

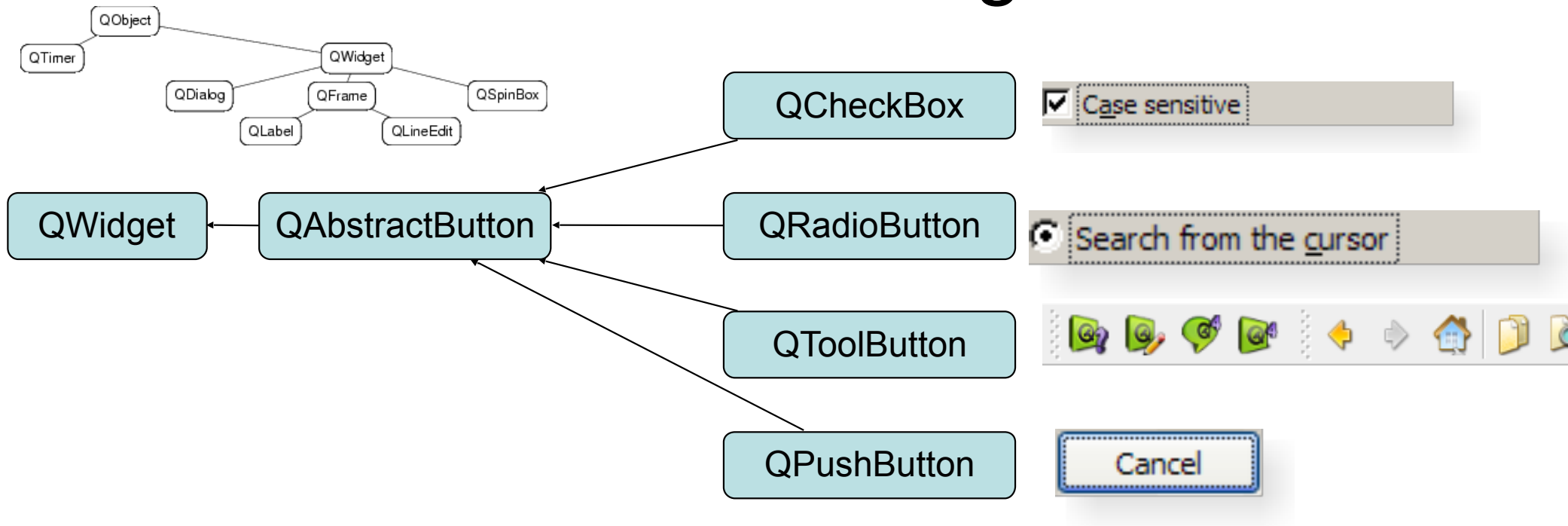


# Arbre d'héritage



# Principaux widgets :

## Arbre d'héritage

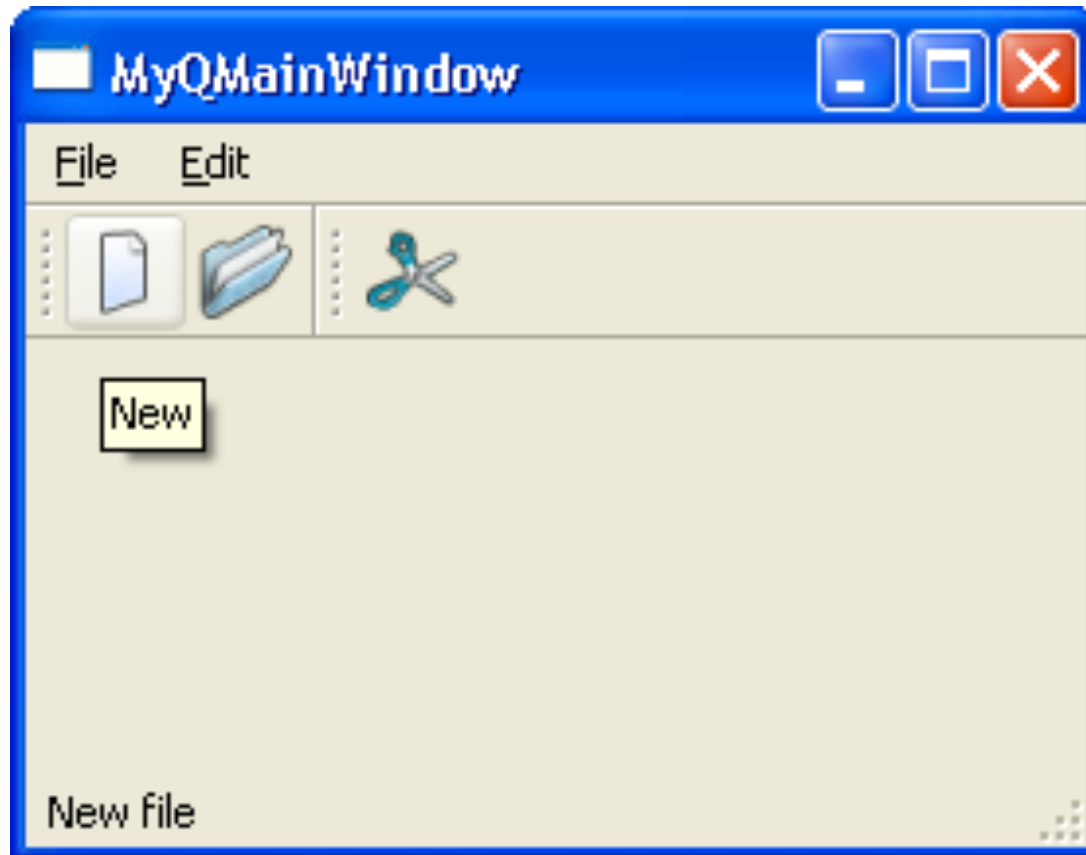


## Hierarchie de classes (type)

- chaque sous-classe hérite des variables et méthodes de sa superclasse
- du plus général au plus particulier
- héritage simple

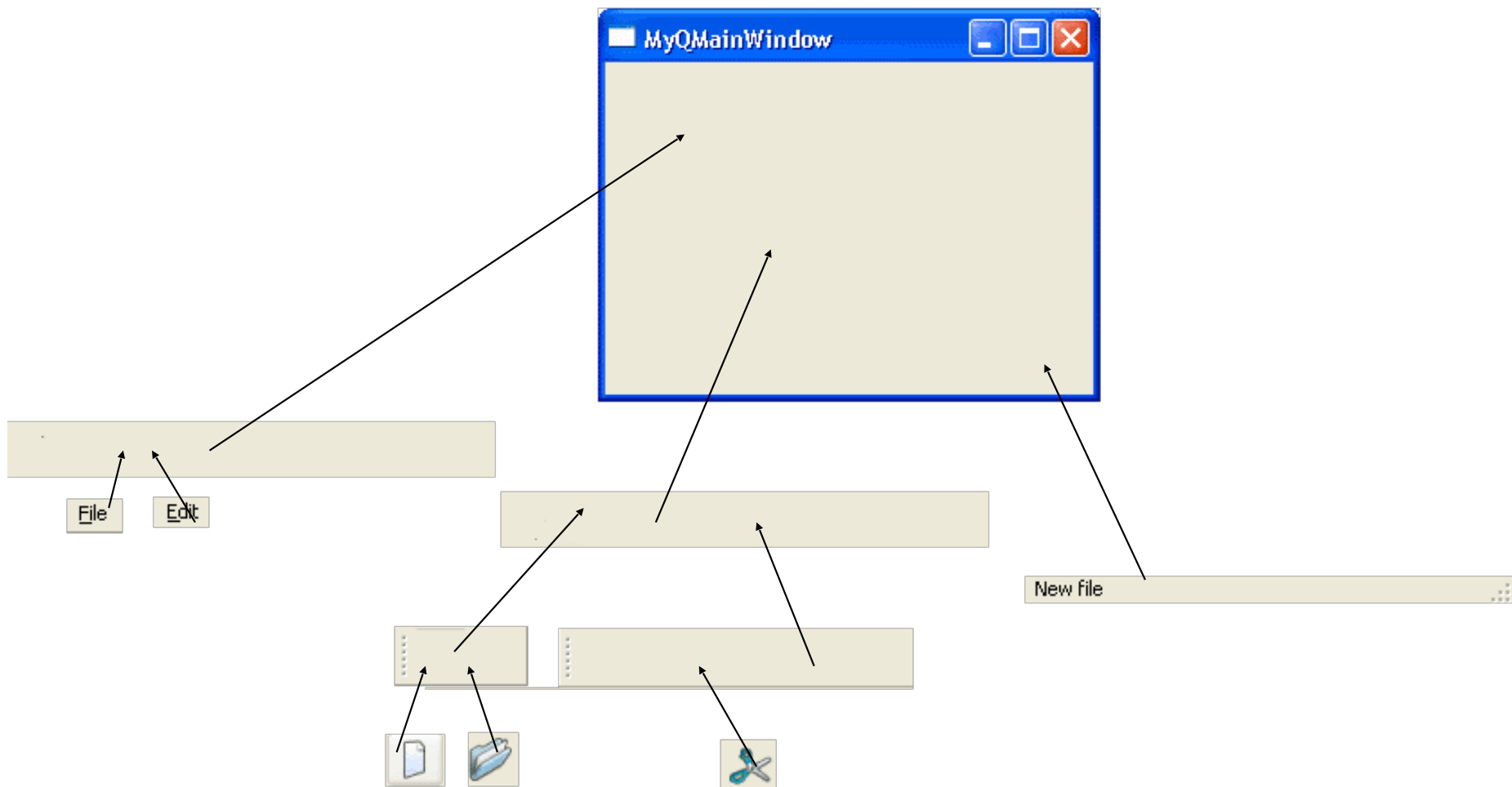
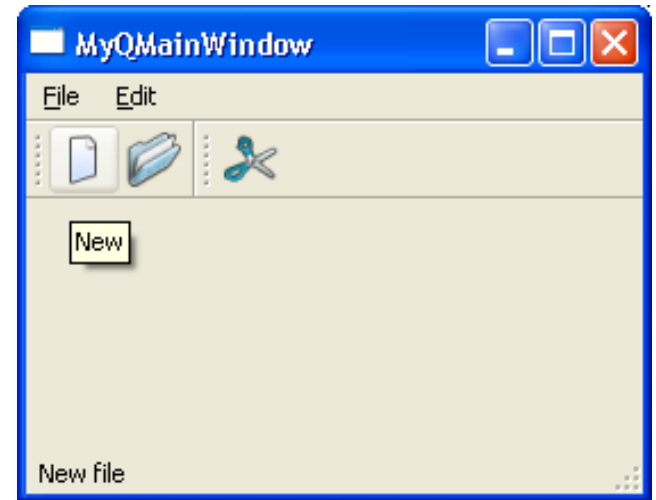
# Arbre d'instanciation

- Hiérarchie d'instance (=objets)
  - Arbre de filiation des objets



# Arbre d'instanciation

- Hiérarchie d'instance (=objets)
  - Arbre de filiation des objets

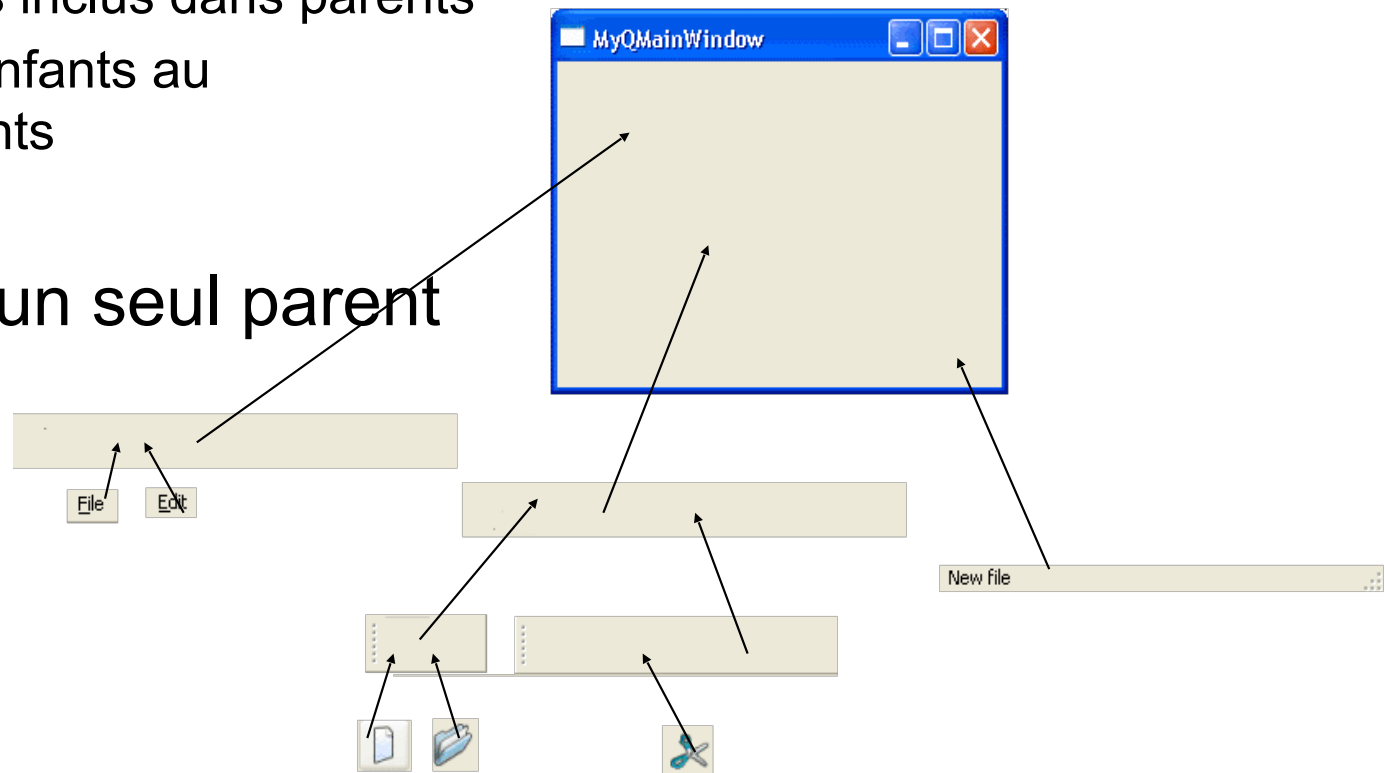


# Arbre d'instanciation

- Les enfants se déclarent auprès de son parent (≠ java)
  - **Qlabel \*label = new Qlabel("Hello", parent);**
  - Exceptions
    - QFile, QApplication...
- Si le parent d'un Widget est nul, le Widget est une fenêtre (Window).
- Que font les parents ?
  - Ils ont une liste des enfants
  - Ils détruisent automatiquement les enfants quand ils sont détruits
  - Enable/disable les enfants quand ils enable/disable eux memes
  - Pareil pour Show/Hide

# Arbre d'instanciation

- Hiérarchie d'instance (=objets)
  - Arbre de filiation des objets
- Chaque objet contient ses enfants
  - Clipping : enfants inclus dans parents
  - Superposition : enfants au dessus des parents
- Un objet n'a qu'un seul parent



# Modules

- QtCore
- **QtWidgets**
- QtBluetooth
- QtOpenGL
- QtSript/QtScriptTools
- QtSql
- QtSvg
- QtWebKit
- QtXml/QtXmlPatterns
- QtMultimedia
- QtSensors

# QtNetwork

- QFtp, QHttp, QTcpSocket, QUdpSocket



# OpenGL : Box3D.h

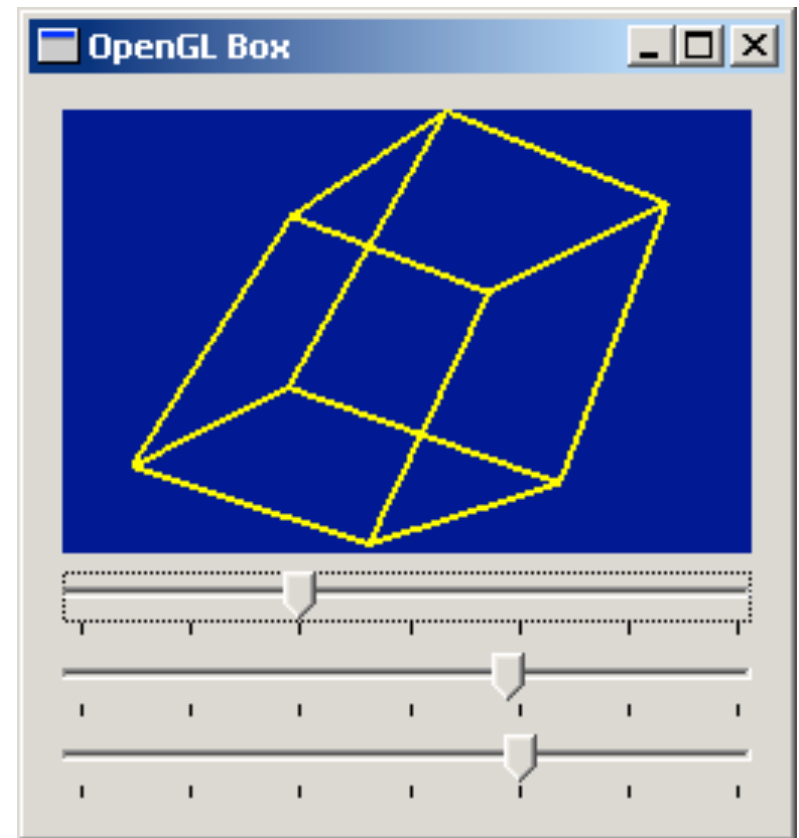
```
#include <QGLWidget>

class Box3D : public QGLWidget {
    Q_OBJECT
    GLuint object;
    GLfloat rotX, rotY, rotZ;

public:
    Box3D( QWidget *parent = 0);
    virtual ~Box3D();

protected:
    virtual void initializeGL();
    virtual void paintGL();
    virtual void resizeGL( int w, int h );
    virtual GLuint makeObject();

public slots:
    void setRotationX(int deg) { rotX = deg; updateGL( ); }
    void setRotationY(int deg) { rotY = deg; updateGL( ); }
    void setRotationZ(int deg) { rotZ = deg; updateGL( ); }
};
```



# OpenGL : Box3D.cpp

---

```
#include "Box3D.h"
```

```
Box3D::Box3D( QWidget *parent )  
: QGLWidget( parent ) {  
    object = 0;  
    rotX = rotY = rotZ = 0.0;  
}
```

```
Box3D::~Box3D() {  
    makeCurrent();  
    glDeleteLists(object, 1);  
}
```

```
void Box3D::initializeGL() {  
    qglClearColor( darkBlue );  
    object = makeObject();  
    glShadeModel(GL_FLAT);  
}
```

```
void Box3D::paintGL() {  
    glClear(GL_COLOR_BUFFER_BIT);  
    glLoadIdentity();  
    glTranslatef(0.0, 0.0, -10.0);  
    glRotatef(rotX, 1.0, 0.0, 0.0);  
    glRotatef(rotY, 0.0, 1.0, 0.0);  
    glRotatef(rotZ, 0.0, 0.0, 1.0);  
    glCallList(object);  
}
```

```
void Box3D::resizeGL( int w, int h ) {  
    glViewport(0, 0, w, h);  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
    glFrustum(-1.0,1.0,-1.0,1.0,5.0,15.0);  
    glMatrixMode( GL_MODELVIEW );  
}
```

# OpenGL : Box3D.cpp

---

```
GLuint Box3D::makeObject() {
    GLuint list = glGenLists( 1 );
    glNewList( list, GL_COMPILE );
    glColor( yellow );
    glLineWidth( 2.0 );

    glBegin( GL_LINE_LOOP );
    glVertex3f( +1.5, +1.0, +0.8 );
    glVertex3f( +1.5, +1.0, -0.8 );
    /* ... */
    glEnd();

    glEndList();
    return list;
}
```

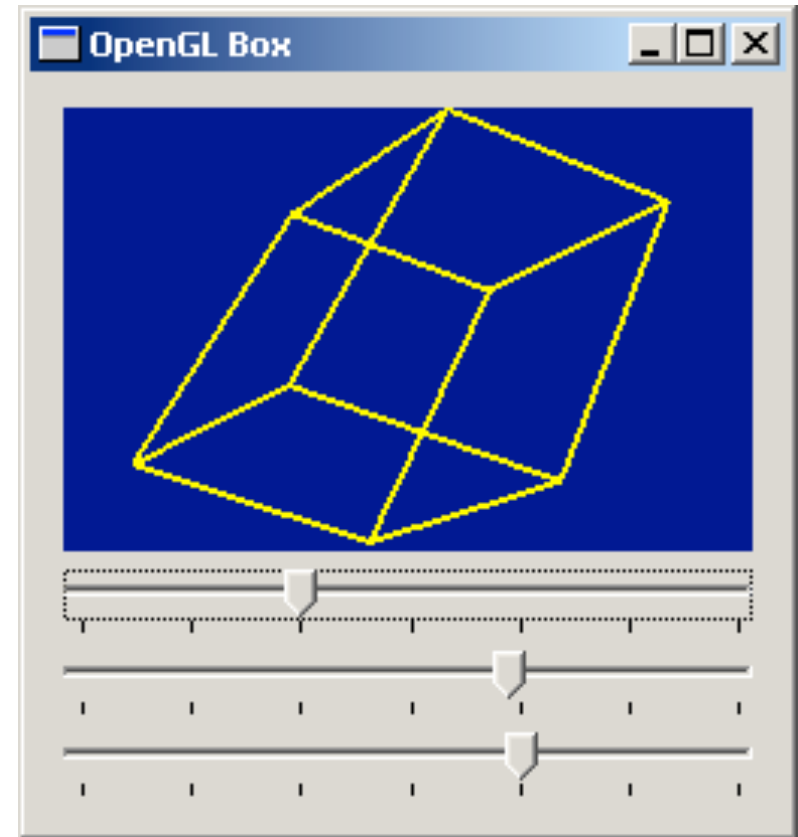
# OpenGL : main

```
#include <qapplication.h>
#include <qslider.h>
#include <qvbox.h>
#include "box3d.h"

void createSlider( QWidget * parent,
                  Box3D * box3d,
                  const char * slot ) // cf. le type de slot !
{
    QSlider *slider =
        new QSlider(QSlider::Horizontal, parent);

    slider->setTickPosition(QSlider::TicksBelow);

    QObject::connect( slider, SIGNAL(valueChanged(int)),
                     box3d, slot);
}
```



# OpenGL : main

```
int main(int argc, char **argv)
{
    QApplication::setColorSpec(QApplication::CustomColor);
    QApplication app(argc, argv);

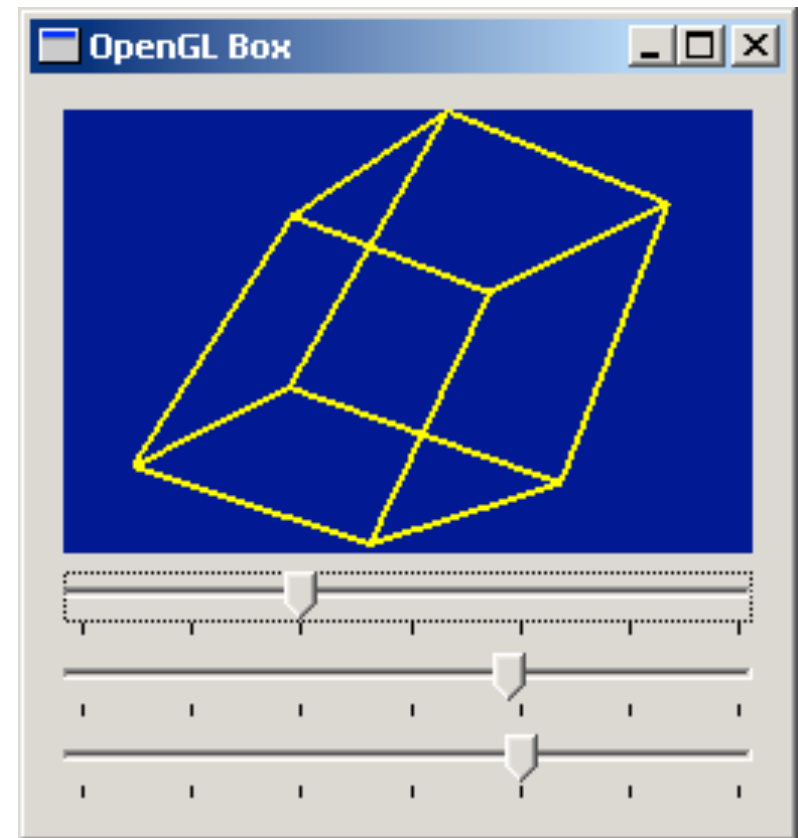
    if (!QGLFormat::hasOpenGL())
        qFatal("This system has no OpenGL support");

    QVBox * parent = new QVBox();
    parent->setCaption("OpenGL Box");
    parent->setMargin(11);
    parent->setSpacing(6);

    Box3D * box3d = new Box3D(parent);

    createSlider( parent, box3d, SLOT(setRotationX(int)) );
    createSlider( parent, box3d, SLOT(setRotationY(int)) );
    createSlider( parent, box3d, SLOT(setRotationZ(int)) );

    parent->resize( 250, 250 );
    parent->show();
    return app.exec();
}
```



# Utile

- Afficher des Traces
  - `#include <QDebug>`
  - `QDebug() << "trace";`
  - Sous Windows, rajouter dans le `.pro`
    - `Qt += console`

# Questions

- Quelle est la différence entre l'arbre d'héritage et l'arbre d'instanciation?
  - héritage de classe (A hérite de B)
  - héritage d'instances (A contient B)
- Quelle classe utiliser pour créer une fenêtre
  - QMainWindow
- Quels sont les différents layouts?
  - QVBoxLayout; QHBoxLayout; QGridLayout; QFormLayout
- Comment afficher une trace?
  - `#include <QDebug>`
  - `qdebug() << "ma trace";`

# Outils Qt





# Les outils Qt

- Qt Creator
- Qt Assistant
- Qt Examples
- QtDemo
- **Qt resources**
- **Qt Designer**

# Ressources

## Fichier source

```
QAction * newAction =  
    new QAction( QIcon(":/images/new.png"),  
                tr("&New..."),  
                this );  
  
newAction->setShortcut( tr("Ctrl+N") );
```

// : signifie: relatif au programme  
// tr() pour éventuelle traduction  
  
// l'accélérateur clavier peut être traduit

## Fichier .qrc

- créé à la main ou par **QtCreator**

```
<!DOCTYPE RCC><RCC version="1.0">  
<qresource>  
    <file>images/copy.png</file>  
    <file>images/cut.png</file>  
    <file>images/new.png</file>  
    <file>images/open.png</file>  
    <file>images/paste.png</file>  
    <file>images/save.png</file>  
</qresource>  
</RCC>
```

