Programmation événementielle

- Présentation
- Licences
- Développement
- Structure générale de Qt
- Premier programme
- Principes
- Signaux et slots
- Gestion mémoire

Présentation

Qt est une bibliothèque logicielle orientée objet et développée en C++ par Qt Development Frameworks, filiale de Nokia.

Qt est une plateforme de développement d'interfaces graphiques (GUI - Graphical User Interface) fournie à l'origine par la société norvégienne Troll Tech, puis rachetée par Nokia en février 2008 (http://qt.nokia.com/).

Qt permet la portabilité des applications qui n'utilisent que ses composants par simple recompilation du code source.

Les environnements supportés sont les Unix (dont Linux) qui utilisent le système graphique X Window System, Windows et Mac OS X.

Présentation

Qt est principalement dédiée au développement d'interfaces graphiques en fournissant des éléments prédéfinis appelés widgets (pour windows gadgets) qui peuvent être utilisés pour créer ses propres fenêtres et des boîtes de dialogue complètement prédéfinies (ouverture / enregistrement de fichiers, progression d'opération, etc).

Qt fournit également un ensemble de classes décrivant des éléments non graphiques : accès aux données, connexions réseaux (*socket*), gestion des fils d'exécution (*thread*), analyse XML, etc.

Qt dispose d'un moteur de rendu graphique 2D performant.

Présentation

Depuis, Qt4 sépare la bibliothèque en modules :

- QtCore : pour les fonctionnalités non graphiques utilisées par les autres modules ;
- QtGui : pour les composants graphiques ;
- QtNetwork : pour la programmation réseau ;
- QtOpenGL : pour l'utilisation d'OpenGL ;
- QtSql : pour l'utilisation de base de données SQL ;
- QtXml : pour la manipulation et la génération de fichiers XML ;
- QtDesigner : pour étendre les fonctionnalités de Qt Designer, l'assistant de création d'interfaces graphiques ;
- QtAssistant : pour l'utilisation de l'aide de Qt ;
- Qt3Support : pour assurer la compatibilité avec Qt 3.
- et de nombreux autres modules, etc.

Qt utilise Unicode 4.0 pour la représentation de ses chaînes de caractères.

Présentation

Les interactions avec l'utilisateur sont gérées par un mécanisme appelé signal/slot. Ce mécanisme est la base de la programmation événementielle des applications basées sur Qt.

Qt est notamment connu pour être la bibliothèque sur laquelle repose l'environnement graphique KDE, l'un des environnements de bureau les plus utilisés dans le monde Linux.

De plus en plus de développeurs utilisent Qt, y compris parmi de grandes entreprises. On peut notamment citer : Google, Adobe Systems, Asus, Samsung, Philips, ou encore la NASA et bien évidemment Nokia.

Présentation

- L'environnement de Qt est aussi constitué de :
  - Qt Jambi : les possibilités de Qt pour le langage JAVA
  - Qtopia : une version de Qt destinée aux dispositifs portables (téléphones, PocketPC, etc..) et aux systèmes embarqués (ex Qt/Embedded)
  - QSA : Qt Script for Applications, ajout de scripts à ses applications.
  - Teambuilder : architecture de compilation distribuée pour les gros projets d'entreprise.

Licences

La société Trolltech mit tout d'abord la version Unix/Linux de Qt sous licence GNU GPL lorsque l'application développée était également sous GNU GPL. Pour le reste, c'est la licence commerciale qui entre en application. Cette politique de double licence est appliquée pour tous les systèmes depuis la version 4.0 de Qt.

Le 14 janvier 2009, Trolltech annonce qu'à partir de Qt 4.5, Qt sera également disponible sous licence LGPL v2.1 (Licence publique générale limitée GNU). Cette licence permet ainsi des développements de logiciels propriétaires, sans nécessiter l'achat d'une licence commerciale auprès de Qt Development Frameworks.

- Les licences Qt : http://qt.nokia.com/products/licensing/
- Les licences GNU : http://www.gnu.org/licenses/licenses.fr.html

Environnement de Développement Intégré

Qt Creator est l'environnement de développement intégré dédié à Qt et facilite la gestion d'un projet Qt. Son éditeur de texte offre les principales fonctions que sont la coloration syntaxique, le complètement, l'indentation, etc... Qt Creator intègre en son sein les outils Qt Designer et Qt Assistant.

Même si Qt Creator est présenté comme l'environnement de développement de référence pour Qt, il existe des modules Qt pour les environnements de développement Eclipse et Visual Studio. Il existe d'autres EDI dédiés à Qt et développés indépendamment de Nokia, comme QDevelop et Monkey Studio.

Qt Designer est un logiciel qui permet de créer des interfaces graphiques Qt dans un environnement convivial. L'utilisateur, par glisser-déposer, place les composants d'interface graphique et y règle leurs propriétés facilement. Les fichiers d'interface graphique sont formatés en XML et portent l'extension .ui. Lors de la compilation, un fichier d'interface graphique est converti en classe C++ par l'utilitaire uic.

Environnement de Développement Intégré

QDevelop est un environnement de développement intégré libre pour Qt. Le but de QDevelop est de fournir dans les environnements les plus utilisés, Linux, Windows et Mac OS X d'un outil permettant de développer en Qt de la même manière avec un IDE unique. Il intègre également les outils Qt-Designer pour la création d'interface graphique et Qt-Linguist pour le support de l'internationalisation.

KDevelop est un environnement de développement intégré (IDE) pour KDE. Il intègre également les outils Qt-Designer pour la création d'interface graphique et Qt-Linguist pour la gestion de l'internationalisation.

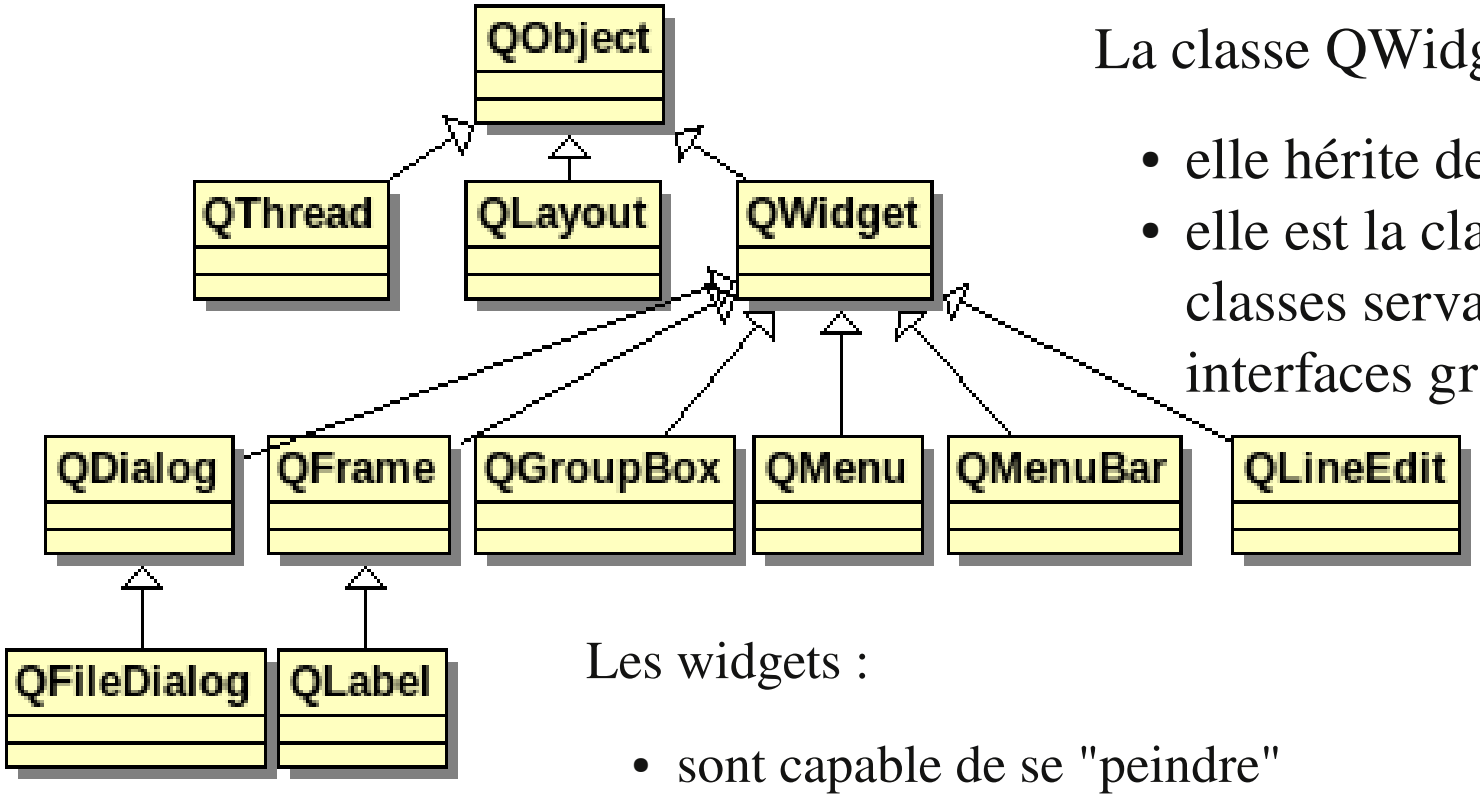
Autres bibliothèques généralistes multi-plateformes, parmi les plus connus :

- GTK+, utilisée par l'environnement graphique GNOME
- wxWidgets (anciennement wxWindows)

Structure générale des classes Qt

- L'API Qt est constituée de classes aux noms préfixés par Q et dont chaque mot commence par une majuscule (QLineEdit, QLabel, ...).
- Arborescence des objets : Les objets Qt (ceux héritant de QObject) peuvent s'organiser d'eux-mêmes sous forme d'arbre. Ainsi, lorsqu'une classe est instanciée, on peut lui définir un objet parent.
- La classe QObject :
  - elle est la classe mère de toutes les classes Qt
  - tout objet dont la classe hérite (directement ou non) de la classe QObject peut émettre et recevoir un signal
  - elle permet une gestion simplifié de la mémoire

Arborescence des classes Qt



La classe QWidget :

- elle hérite de QObject
- elle est la classe mère de toutes les classes servant à réaliser des interfaces graphiques

Les widgets :

- sont capable de se "peindre"
- sont capable de recevoir les évènements souris, clavier
- sont les éléments de base des interfaces graphiques
- sont tous rectangulaires
- ils sont ordonnés suivant l'axe z (gestion de la profondeur)
- ils peuvent avoir un widget parent

Premier programme

Qt4 pose comme principe que pour utiliser une classe, il faut inclure un fichier *header* du nom de la classe.

```
#include <QApplication>
#include <QLabel>

int main( int argc, char* argv[] )
{
    QApplication MonAppli( argc,
argv );

    QLabel *pMonTexte = new
QLabel("<H1><center>Bonjour à
tous</center></H1>", NULL);
    pMonTexte->resize(200, 70);
    pMonTexte->show();
    return MonAppli.exec();
}
```

La classe QApplication gère les options standards d'un programme Qt (argc, argv) et les propriétés globales de l'application :

- le style des fenêtres, la couleur du fond, la taille de la fenêtre principale
- la boucle d'écoute des évènements
- la récupération d'information globale (taille écran)
- le langage de l'application internationalisation i18n (cf. L10n, g11n)

Un objet de type QApplication doit être créé avant celle des widgets

Utiliser QCoreApplication si l'application n'est pas graphique

Affichage du message

Boucle infinie d'attente des événements jusqu'à la fermeture du dernier widget

Principes de fonctionnement

- Instanciation de la classe QApplication
  - Elle doit être créée avant tout objet graphique et reçoit tous les paramètres transmis à la fonction main (argc, argv). Elle s'occupe de toute la gestion des événements (et les envoie aux widgets concernés). Cet objet est toujours accessible dans le programme grâce au pointeur global nommé qApp.
  - Les applications doivent se terminer proprement en appelant QApplication::quit()
  - Cette méthode est appelée automatiquement lors de la fermeture du dernier widget.
  - Un widget est toujours créé caché, il est donc nécessaire d'appeler la méthode show() pour l'afficher

Compilation : ligne de commande

- La génération d'une application se fait en plusieurs étapes :
  - création d'un répertoire et des sources (dépend de l'EDI utilisé)
    le nom initial du répertoire détermine le nom du projet et donc de l'exécutable qui sera produit.
  - qmake –project
    génère un fichier .pro qui décrit comment générer un Makefile pour compiler ce qui est présent dans le dossier courant.
  - qmake [-makefile]
    génère un Makefile à partir des informations du fichier .pro
  - make
    appel classique à l'outil make, par défaut il utilise un fichier appelé Makefile

QMAKE

- Qt se voulant un environnement de développement portable et ayant le MOC comme étape intermédiaire avant la phase de compilation/édition de liens, il a été nécessaire de concevoir un moteur de production spécifique. C'est ainsi qu'est conçu le programme qmake.
- Ce dernier prend en entrée un fichier (avec l'extension .pro) décrivant le projet (liste des fichiers sources, dépendances, paramètres passés au compilateur, etc...) et génère un fichier de projet spécifique à la plateforme. Ainsi, sous les systèmes UNIX/Linux, qmake produit un Makefile.
- Le fichier de projet est fait pour être très facilement éditable par un développeur. Il consiste en une série d'affectations de variables.
- Manuel : http://doc.trolltech.com/4.x/qmake-manual.html

Exemple Fichier Projet

```
# -------------------------------------------------
# Project created by QtCreator 2010-10-03T09:33:45
# -------------------------------------------------
TARGET = Chronometre
TEMPLATE = app
SOURCES += main.cpp \
    chronometreihm.cpp \
    cchronometre.cpp
HEADERS += chronometreihm.h \
    cchronometre.h
FORMS += chronometreihm.ui
```

- Hiérarchie
  - D'une manière générale, les widgets sont hiérarchiquement inclus les uns dans les autres. Le principal avantage est que si le parent est déplacé, les enfants le sont aussi.
  - On va donc ajouter un bouton à notre application pour obtenir ceci (les deux éléments seront inclus dans un même widget) :

Widgets parent et enfant

```
#include <QApplication>
#include <QLabel>
#include <QPushbutton>

int main( int argc, char* argv[] )
{
    QApplication MonAppli( argc, argv );
    QWidget *pMaFenetre = new QWidget();
    //position horizontale et verticale de la fenetre, puis dimensions de la fenetre
    pMaFenetre->setGeometry(100,300,200,80);
    //pour forcer l'affichage avec les accents-> QString::fromUtf8
    QLabel* pMonTexte = new QLabel(QString::fromUtf8("<H1><center>Bonjour à tous
     </center></H1>"),pMaFenetre);
    QPushButton *pMonBouton = new QPushButton("Quitter",pMaFenetre);
    pMonBouton->setGeometry(40,40,pMonTexte->width(),40);
    pMaFenetre->show();
    return MonAppli.exec();
}
```

- Lors de la création d'un widget on indique le widget parent (ici pMaFenetre). Les widgets sont alors dessinés à l'intérieur de l'espace attribué à leur parent.
- Bien sur ce bouton n'a aucune action car une pression dessus ne crée pas d'événements, il faut utiliser le mécanisme des signaux et slots.

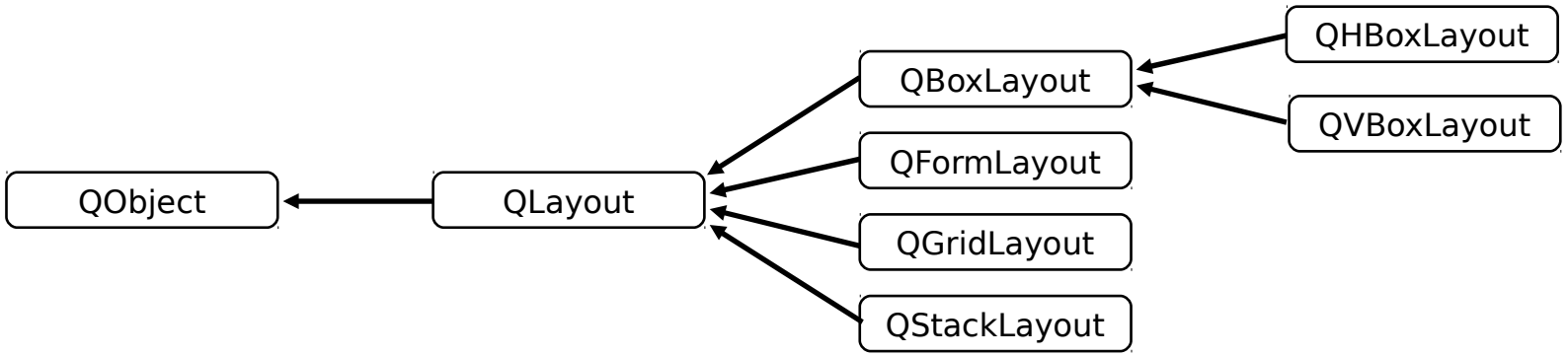
Hiérarchie des widgets

- D'une manière générale, les widgets sont hiérarchiquement inclus les uns dans les autres. Le principal avantage est que si le parent est déplacé, les enfants le sont aussi.
- Les gestionnaires de disposition, les classes de Layout, simplifient ce travail :
  - On peut ajouter des widgets dans un layout

    ```
    void QLayout::addWidget(QWidget *widget)
    ```

  - On peut associer un layout à un widget qui devient alors le propriétaire du layout et parent des widgets inclus dans le layout

    ```
    void QWidget::setLayout (QLayout *layout)
    ```



- Événements
  - Pour la gestion des événements, Qt utilise un mécanisme de communication d'objets faiblement couplés appelé signal / slot. Faiblement couplé signifie que l'émetteur d'un signal ne sait pas quel objet va le prendre en compte (d'ailleurs, il sera peut être ignoré). De la même façon, un objet interceptant un signal ne sait pas quel objet a émis le signal.
  - Tout objet dont la classe hérite (directement ou non) de la classe QObject peut émettre et recevoir un signal.

- Signaux
  - Ces signaux n'ont rien à voir avec les signaux UNIX ou Linux (de communication inter-processus), ce sont des événements qui surviennent suite à des actions de l'utilisateur.
  - Un signal peut être connecté à un autre signal. Dans ce cas, lorsque le premier signal est émis, il entraîne l'émission du second.
  - Émission d'un signal peut être "automatique" par exemple lorsqu'on appui sur un bouton, le signal existe dans la classe utilisée.

- Signaux
  - Mais on peut aussi forcer l'émission à l'aide de la méthode

```
emit nomSignal(parametreSignal);
```

  - La déclaration d'un signal personnalisé est possible, il faut savoir qu'il n'a pas de valeur de retour (void) et pas de corps, le mot clé signals apparait dans le fichier d'entête comme public, private...

```
signals :
   void nomSignal(parametreSignal);
```

- Slots
  - Les slots étant des méthodes, ils peuvent être appelés explicitement (sans présence d'un événement) comme toute méthode. Ils peuvent donc être private, protected ou public avec les conséquences habituelles pour leur connexion (private pour la classe ...). De même, ils peuvent être virtuels et/ou surchargés. Ils ont seulement la propriété d'être connectés à un signal et dans ce cas, il sera automatiquement appelé lorsque le signal sera émis.

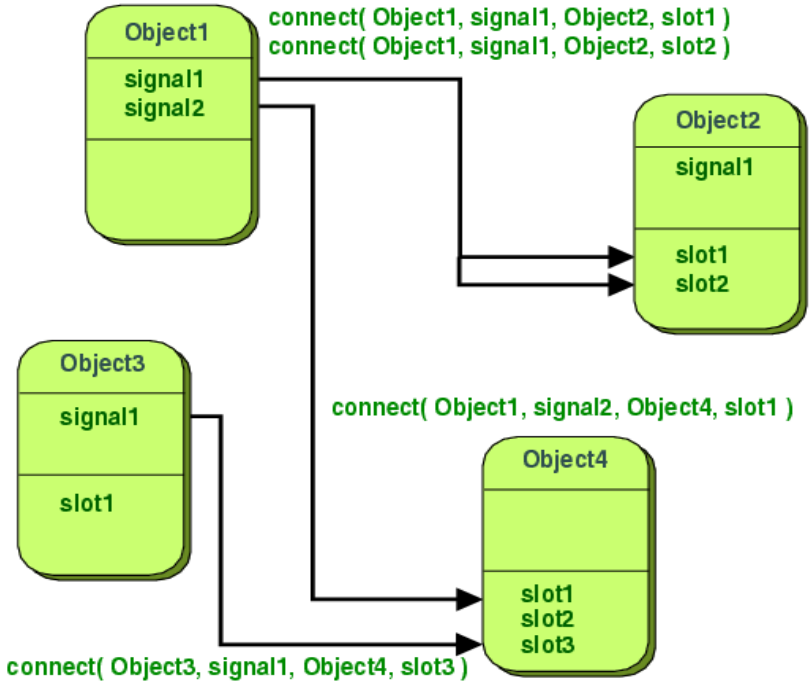
- Connexion
  - signal et slot doivent être compatibles (avoir la même signature) pour être connectés.
  - un signal peut être connecté à plusieurs slots. Attention : les slots sont activés dans un ordre arbitraire.
  - plusieurs signaux peuvent être connectés à un seul slot.
  - une connexion signal / slot peut être réalisée, par la méthode :

```
bool QObject::connect ( const QObject * sender, const char *
signal, const char * method, Qt::ConnectionType type =
Qt::AutoConnection ) const
```

- Déconnexion
  - une connexion signal/slot peut être supprimée, par la méthode :

```
bool QObject::disconnect ( const QObject * sender, const char *
signal, const QObject * receiver, const char * method )
```

- Vue générale



- Exemple

```
#include <qapplication.h>
#include <qlabel.h>
#include <qpushbutton.h>

int main( int argc, char* argv[] )
{
    QApplication MonAppli( argc, argv );
    QWidget *pMaFenetre = new QWidget();
    //position hor,vert fenetre, dimensions fenetre
    pMaFenetre->setGeometry(100,300,200,80);
    // pour forcer l'affichage avec les accents-> QString::fromUtf8
    QLabel* pMonTexte = new QLabel(QString::fromUtf8("<H1><center>Bonjour à tous
    </center></H1>"),pMaFenetre);
    QPushButton *pMonBouton = new QPushButton("Quitter",pMaFenetre);
    pMonBouton->setGeometry(40,40,pMonTexte->width(),40);
    QObject::connect(pMonBouton,SIGNAL(clicked()),&MonAppli, SLOT(quit()));
    pMaFenetre->show();
    return MonAppli.exec();
}
```

L'ajout de cette ligne permet de rendre actif le bouton

Gestion mémoire

- Qt est basé autour du modèle d'objet de Qt. Cette architecture est ce qui rend Qt puissant et facile à employer. Elle est entièrement basée autour de la classe QObject et de l'outil moc. Les classes graphiques dérivent toutes de QObject. Les QObject s'organisent eux mêmes en arbre d'objets. Quand on crée un QObject (un objet graphique par exemple) avec un autre QObject comme parent, l'objet parent ajoute le nouvel objet créé à sa liste d'enfants.
- Si le parent est détruit (appel de son destructeur), il détruira automatiquement tous ses enfants.
- On ne détruira (opérateur delete) donc que les QObject créés par l'opérateur new qui n'ont pas de parent.

Gestion mémoire

- Exemple :

Chose.h

```
#include <QObject>
#include <iostream>

using namespace std;

class Chose : public QObject {
public:
 Chose(QObject *parent = 0, char *nom = 0);
 ~Chose();
 void Existe();
};
```

Chose.cpp

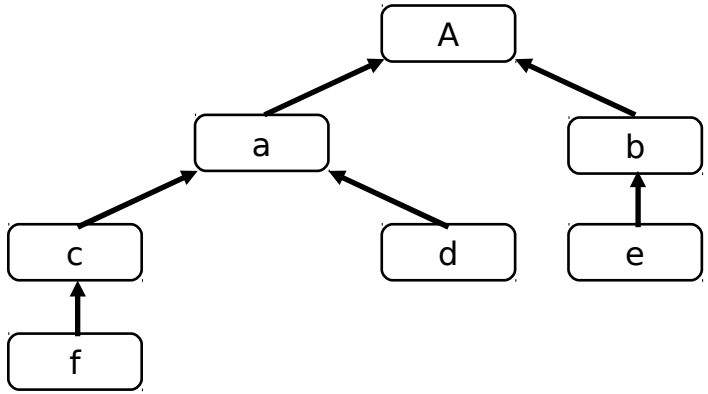
```
#include "Chose.h"

Chose::Chose(QObject *parent, char *nom):
QObject(parent) {
   setObjectName(nom);
   cout << "Constructeur : " <<
objectName().toStdString() << endl; }

Chose::~Chose() {  cout << "Destructeur : "
<< objectName().toStdString() << endl; }

void Chose::Existe()   {
   cout << "En exécution : " <<
objectName().toStdString() << endl; }
```

Gestion mémoire

- Exemple (suite) :

main.cpp

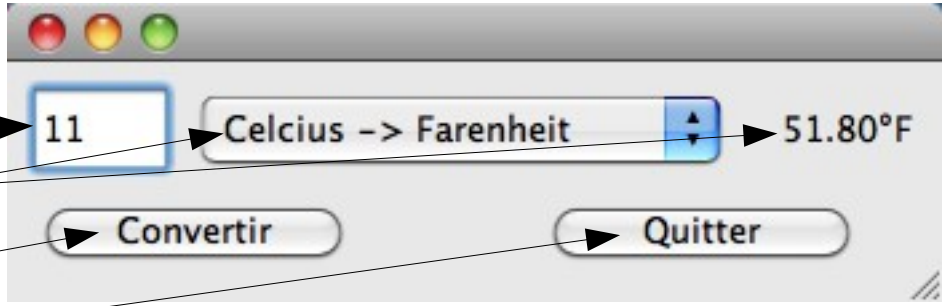
```
#include "Chose.h"

int main( int argc, char* argv[] ) {
Chose A(NULL, "A" );
Chose *aA   = new Chose(&A, "a");
Chose *bA   = new Chose(&A, "b");
Chose *aaA  = new Chose(aA, "c");
Chose *baA  = new Chose(aA, "d");
Chose *abA  = new Chose(bA, "e");
Chose *aaaA = new Chose(aaA, "f");

A.Existe(); aA->Existe();  bA->Existe();
aaA->Existe(); baA->Existe(); abA-
>Existe(); aaaA->Existe();

return 0;
}
```



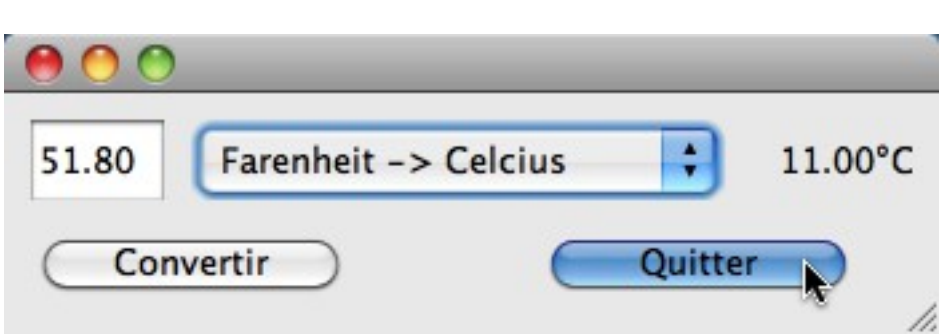

```
Constructeur : A Constructeur : a Constructeur : b
Constructeur : c Constructeur : d Constructeur : e
Constructeur : f

En exécution : A En exécution : a En exécution : b
En exécution : c En exécution : d En exécution : e
En exécution : f

Destructeur : A Destructeur : a Destructeur : c
Destructeur : f Destructeur : d Destructeur : b
Destructeur : e
```

Exemple

```
#include <QtGui>

class Convertisseur : public QWidget {
Q_OBJECT
private:
 QLineEdit   *valeur;
 QLabel      *resultat;
 QComboBox   *choix;
 QPushButton *bConvertir;
 QPushButton *bQuitter;
public:
 Convertisseur(QWidget *parent = NULL);
 ~Convertisseur();
signals:
 void actualiser();
private slots:
 void convertir();
 void permuter(int index);
};
```



Exemple




```
#include <QApplication>
#include "Convertisseur.h"
int main(int argc, char* argv[]) {
   QApplication app(argc, argv);
   Convertisseur convertisseur;
   convertisseur.adjustSize();
   convertisseur.show();
   return app.exec(); }
```

Exercice

```
#include "Convertisseur.h"
Convertisseur::Convertisseur(QWidget *parent) : QWidget(parent) {
valeur     = new QLineEdit(this);
resultat   = new QLabel(this);
choix      = new QComboBox(this);
bConvertir = new QPushButton("Convertir", this);
bQuitter   = new QPushButton("Quitter", this);

valeur->setGeometry(10, 10, 50, 30); resultat->setGeometry(290, 10, 50, 30);
choix->setGeometry(70, 10, 200, 30); bConvertir->setGeometry(10, 50, 120, 30);
bQuitter->setGeometry(200, 50, 120, 30);
choix->addItem("Celcius -> Farenheit"); choix->addItem("Farenheit -> Celcius");

connect(bConvertir, SIGNAL(clicked()), this, SLOT(convertir()));

connect(this,

connect(choix,

connect(bQuitter,
}
```

Exercice

```
Convertisseur::~Convertisseur() {}

void Convertisseur::convertir() {
 switch (choix->currentIndex()) {
  case 0: resultat->setText(QString::fromUtf8("%1°F").arg(9 * valeur->text().toDouble() / 5
+ 32, 0, 'f', 2)); break;
  case 1: resultat->setText(QString::fromUtf8("%1°C").arg(5 * (valeur->text().toDouble() -
32 )  / 9, 0, 'f', 2)); break;
 }
}

void Convertisseur::permuter(int index) {
 valeur->setText(resultat->text().left(resultat->text().size() - 2));


}
```

Documentation (http://doc.qt.nokia.com/)

- Tutoriel : http://doc.trolltech.com/4.7/tutorials.html

- La référence Qt4 : http://doc.trolltech.com/4.7/index.html, et plus particulièrement les concepts de base (http://doc.trolltech.com/4.7/qt-basic-concepts.html) :
  - Signals and Slots ;
  - Main Classes ;
  - Main Window Architecture ;
  - Internationalization (pensez à écrire votre programme en anglais puis à le traduire en français) ;
  - OpenGL Module.

- Les documentations des outils indispensables :
  - qmake (http://doc.trolltech.com/4.7/qmake-tutorial.html) ;
  - Qt Designer (http://doc.trolltech.com/4.7/designer-manual.html) ;
  - Qt Linguist (http://doc.trolltech.com/4.7/linguist-manual.html) ;
  - Qt Creator (http://doc.qt.nokia.com/qtcreator-2.0/index.html).

Documentation

- Quelques autres sources d'informations méritant le détour :
  - Qt Centre : http://www.qtcentre.org/
  - Qt Forum (en anglais) : http://www.qtforum.org/
  - QtFr, un site français, http://www.qtfr.org/
  - L'Independant Qt Tutorial qui est plein d'exemples (et disponible en français) : http://www.digitalfanatics.org/projects/qt_tutorial/
  - Qt-Apps.org propose un annuaire de programmes libres construits sur Qt : http://www.qt-apps.org/
  - Et Qt-Prop.org, l'équivalent pour les programmes non-libres (comme Skype et GoogleEarth) : http://www.qt-prop.org/
- Liste des Tps : http://pluton.up.univ-mrs.fr/eremy/Ens/LPSIL.Qt/