Initiation à Qt

1 Qu'est-ce que Qt ?

Qt est une API orientée objet et développée en C++, conjointement par The Qt Company et Qt Project. Qt offre des composants d'interface graphique (widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc. ;

Qt permet la portabilité des applications qui n'utilisent que ses composants par simple recompilation du code source. Les environnements pris en charge sont les Unix et aussi les systèmes d'exploitation embarqués somme Android.

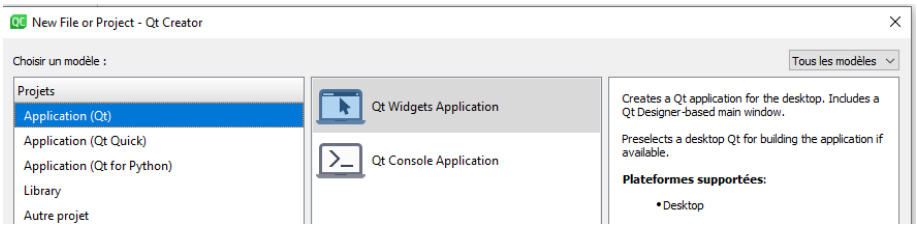
Qt intègre des bindings avec plus d'une dizaine de langages autres que le C++, comme Ada, C#, Java, Python, Ruby, Visual Basic, etc.

Qt est notamment connu pour être le Framework sur lequel repose l'environnement graphique KDE Plasma, l'un des environnements de bureau par défaut de plusieurs distributions GNU/Linux.

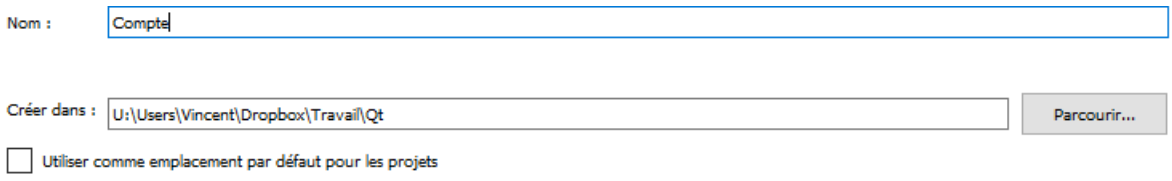
2 Création d'une application avec QtCreator

L'application que je vous propose d'écrire permet de gérer un compte bancaire de façon graphique. Cette application utilise la classe CCompte développée et testée en mode console.

- Lancer Qt Creator
- Créer un nouveau projet de type "Qt Widgets Application"



- Donner un nom à votre projet et choisir le répertoire

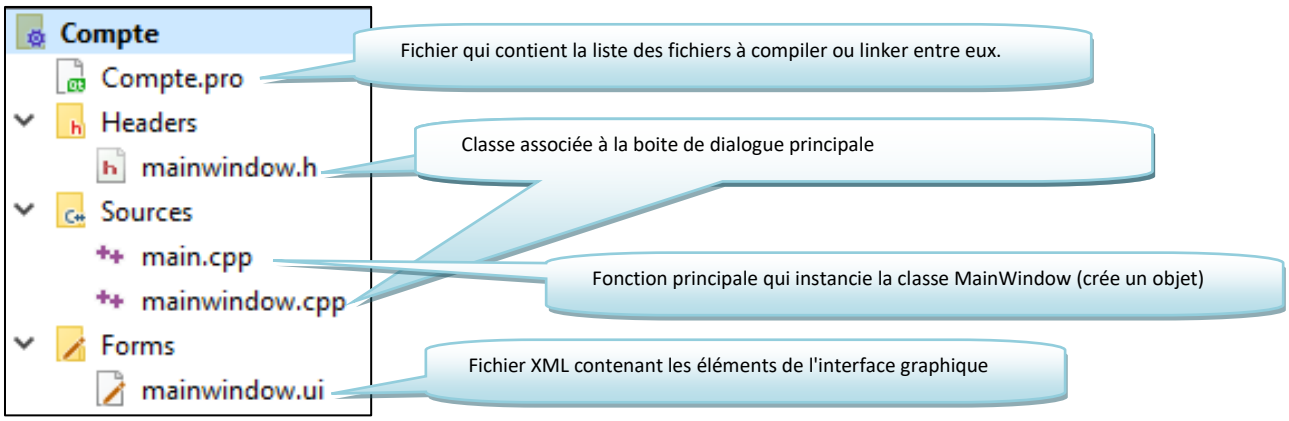


- Define Build System : qmake
- Class information : laisser ce qui est proposé
- Translation File : none (pas de traduction de l'IHM en plusieurs langues)
- Choisir le Kit de développement installé (version DesKtop Qt 6.2.0 MSVC2019 64 bits)

- Laissez les informations de base proposées par défaut.

- Terminer l'installation

2.1 Structure de l'application



2.2 Analyse de main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

crée une instance ***a*** de l'application

crée une instance ***w*** de la fenêtre principale

Affiche la fenêtre

exécute l'application qui va surveiller les interactions de l'utilisateur avec l'IHM (Interface Homme-Machine)

2.3 Analyse de la classe MainWindow

Mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Espace de nommage comportant la classe MainWindow

Notre classe perso (MainWindow) est une classe qui hérite de QMainWindow (appartenant au framework Qt)

Macro permettant d'utiliser des fonctionnalités spécifiques de Qt comme les signaux et les slots.

L'interface graphique correspond à un pointeur nommé « ui »

Mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

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

Le constructeur : charge l'interface graphique

Le destructeur : ferme l'interface graphique et libère la mémoire

2.4 Première exécution

Observez l'existence d'un fichier Compte.pro qui contient les informations sur notre projet

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    mainwindow.h

FORMS += \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

indique les librairies devant être utilisées dans notre projet : core et gui (Graphic User Interface)

A partir de la version 4, les widgets sont chargés dans une librairie séparée

Les fichiers source

Les fichiers d'entête

Le fichier d'interface graphique

Emplacement de la cible (exécutable final)

- L'utilitaire qmake permet de créer le fichier MakeFile à partir de ce fichier .pro. Si vous exécutez qmake (menu compiler-exécuter qmake), vous devriez voir apparaître sur votre disque un dossier "build-Compte…" contenant un fichier Makefile

- Si vous exécutez ensuite une compilation, Le compilateur va interpréter le fichier **Makefile** et créer l'exécutable **.exe** .

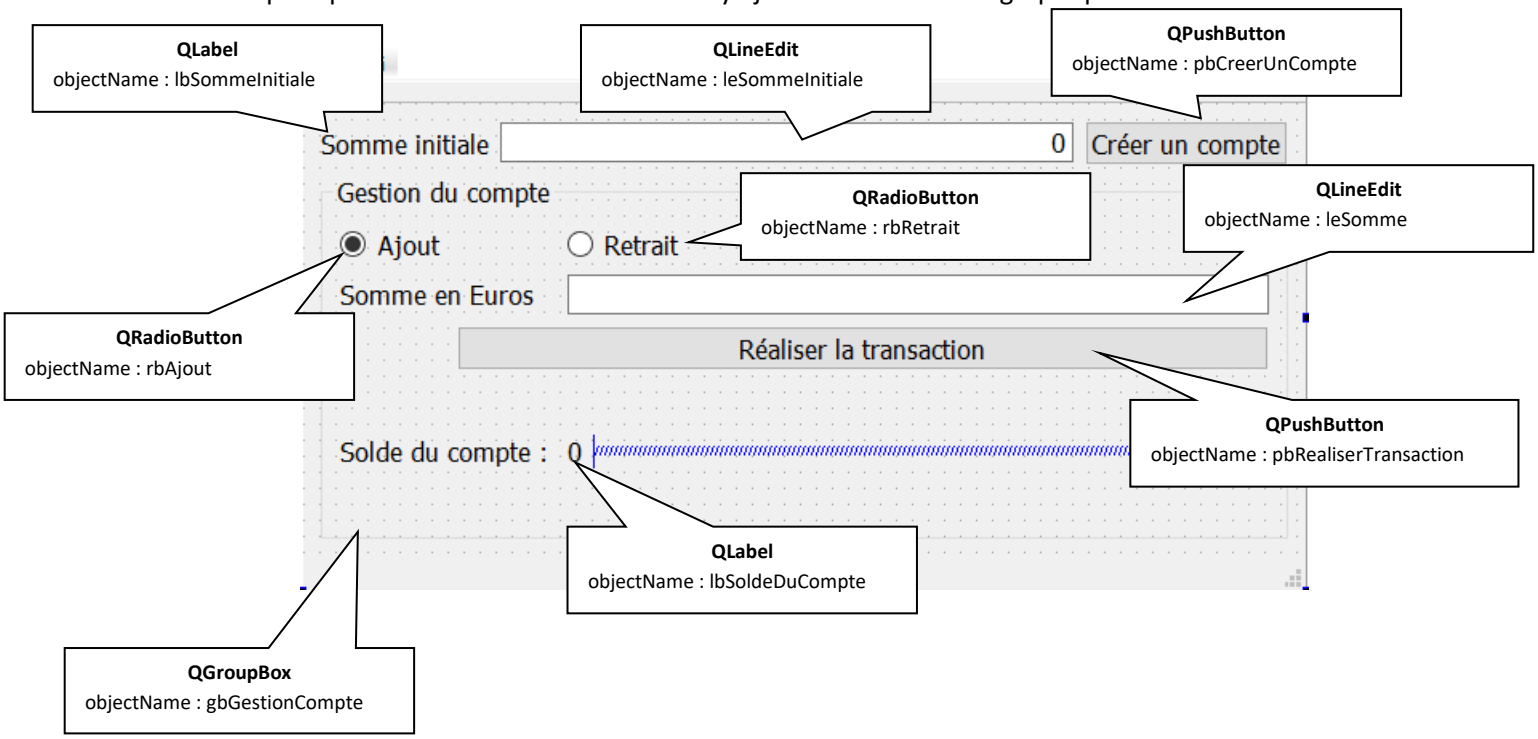
- Sous QtCreator, vous pourrez lancer votre application en mode **release** ou **debug**

- Ces flèches vont lancer qmake (la 1ère fois) puis faire la compilation et l'édition de liens de votre programme.

  Attention, si vous modifiez le fichier .pro, réexécuter qmake avant de lancer l'exécution.

2.5 Dessin de l'IHM

En faisant un double clic sur le fichier « mainWindow.ui », Qt Designer s'ouvre et présente un ensemble d'outils pour pouvoir créer l'IHM. Nous allons y ajouter nos éléments graphiques.



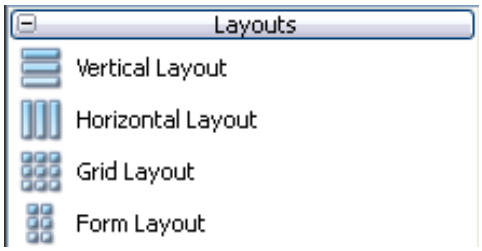
2.6 Organisation de l'IHM

Il ne suffit pas de placer les éléments graphiques sur l'écran pour que l'IHM soit terminée. Il faut aussi veiller à plusieurs choses :

- Vérifier le focus des éléments et l'ordre dans lequel la tabulation les sélectionne.
- Vérifier que les éléments ont des dimensions homogènes et que le redimensionnement de la fenêtre s'accompagne d'un redimensionnement des éléments graphiques.

Si vous testez la fenêtre (Menu Outil - Form Editor – Previsualisation), vous voyez un aperçu de la fenêtre. Vous pouvez constater que lorsque l'on redimensionne la fenêtre, les éléments graphiques ne changent pas de taille. Pour résoudre ce problème, nous allons attacher à la zone principale un gestionnaire de présentation. Il en existe de différents types.

- verticaux
- horizontaux
- sur une grille
- de type formulaire
  (label à gauche et zone d'édition à droite)



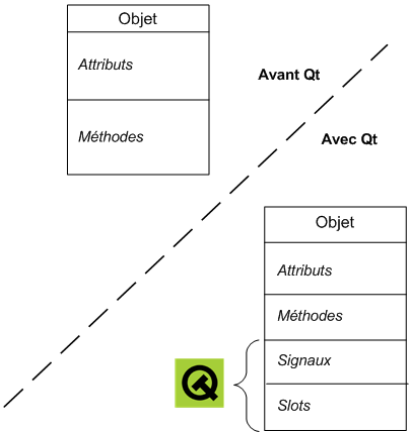
Nous allons choisir des « layouts » de type Grid Layout. Pour cela, sélectionner le "GroupBox" "Gestion du Compte" puis cliquer sur l'icône « Grid Layout » ou en Français "Mettre en page dans une grille". Sélectionnez ensuite la fenêtre principale et faire de même. Si vous testez de nouveau l'IHM, vous devriez voir que les éléments graphiques s'adaptent maintenant à la taille de la fenêtre.

2.7 Les signaux et les slots

Une fenêtre graphique est composée d'éléments qui doivent pouvoir réagir à des évènements (clic ou déplacement souris, touche clavier, joystick, etc…).

Pour assurer ceci, Qt dispose d'un mécanisme qui lui est propre : les signaux et les slots.

- **Un signal** : c'est un message envoyé par un widget lorsqu'un évènement se produit.
  Exemple : on a cliqué sur un bouton.
- **Un slot** : c'est la fonction qui est appelée lorsqu'un évènement s'est produit. On dit que le signal appelle le slot. Concrètement, un slot est une méthode d'une classe.
  Exemple : le slot quit() de la classe QApplication, qui provoque l'arrêt du programme.



Les signaux et les slots sont considérés par Qt comme des éléments d'une classe à part entière, en plus des attributs et des méthodes.

Voici un schéma qui montre ce qu'un objet pouvait contenir avant Qt, ainsi que ce qu'il peut contenir maintenant qu'on utilise Qt :

Pour cette première application graphique, nous allons utiliser les slots prédéfinis de Qt associés au clic sur un bouton.

- Dans l'environnement graphique, sélectionner le bouton "Créer un compte" puis choisir le menu contextuel "Aller au slot".
- Choisir ensuite le signal "clicked()" puis OK
- Qt vous ouvre maintenant l'éditeur de code et vous place le curseur sur la méthode "`on_pbCreerUnCompte_clicked()`" qui sera exécutée chaque fois que l'on cliquera sur le bouton. Remarquez aussi la déclaration de cette méthode dans le fichier de déclaration "`mainwindow.h`" dans une zone "private slots:"
- Ajouter ligne `#include <QtWidgets>` en haut du fichier "`mainWindow.h`". Ceci nous permettra d'utiliser les Widgets dans notre projet
- Compléter la méthode `on_pbCreerUnCompte_clicked()`

```
void MainWindow::on_pbCreerUnCompte_clicked()
{
    QMessageBox::about(this,"Test","Appui sur le bouton créer compte");
}
```

- Tester votre application. Vérifiez que le clic sur le bouton "Créer un compte" affiche le message « Appui sur le bouton créer compte ».
- Rajoutez un slot pour la gestion du bouton "Réaliser la transaction".

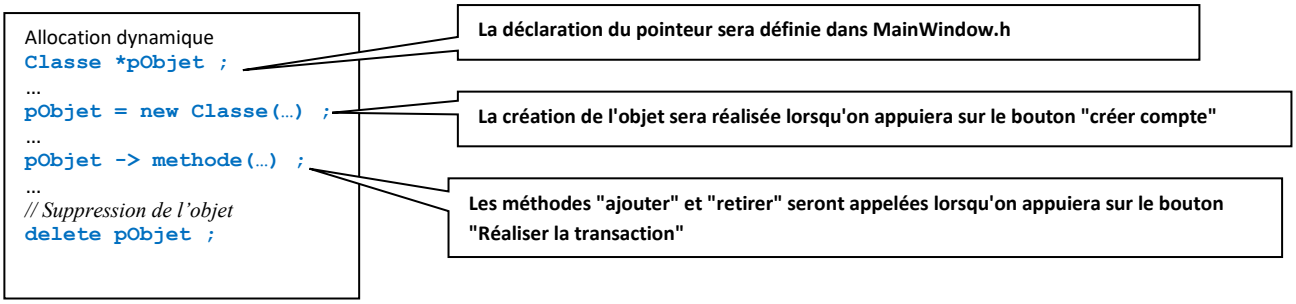
2.8 Utilisation dans cette fenêtre de la classe CCompte (testée en mode console)

La classe CCompte correspond à la partie métier de notre application. L'essentiel de l'intelligence est dans cette classe.

Notre classe MainWindow ne correspond qu'à l'IHM (Interface Homme Machine) et ne doit donc s'occuper que de la partie visuelle de l'application.

Il est très important de bien découpler la partie métier de la partie IHM

Nous allons intégrer un compte bancaire dans notre application. Pour cela, nous allons nous conformer à la colonne 2 de l'algorigramme qui nous permet de déterminer comment créer les objets.



- Copier les fichiers Compte.h et Compte.cpp dans le répertoire de votre projet Qt.
- Ajouter ces fichiers à votre projet (bouton droit sur le projet, puis Ajouter fichier existant).
- Exécuter qmake pour recréer le fichier "MakeFile"
- On peut maintenant ajouter les éléments de codage dans le fichier mainwindow.h et mainwindow.cpp.

```
// mainwindow.h

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtWidgets>

class CCompte;  // Définit l'existence de la classe CCompte

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_pbCreerUnCompte_clicked();

    void on_pbRealiserTransaction_clicked();

private:
    Ui::MainWindow *ui;

   CCompte *pCompte;
};

#endif // MAINWINDOW_H
```

```cpp
// mainwindow.cpp

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "Compte.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    ui->gbGestionCompte->setVisible(false);
}

MainWindow::~MainWindow()
{
    delete ui;
}



void MainWindow::on_pbCreerUnCompte_clicked()
{
    QString strValeur = ui->leSommeInitiale->text();
    bool ok;
    double sommeInit=strValeur.toDouble(&ok);
    if (ok)
    {
        pCompte=new CCompte(sommeInit);
        ui->gbGestionCompte->setVisible(true);
        ui->lbSoldeDuCompte->setText(strValeur);
        ui->leSommeInitiale->setEnabled(false);
        ui->pbCreerUnCompte->setEnabled(false);
    }
    else
        QMessageBox::about(this,"info","Saisir un nombre réel");
}

void MainWindow::on_pbRealiserTransaction_clicked()
{
    bool ok;
    double val;

    // Vérifier si on ajoute ou si on retire
    if (ui->rbAjout->isChecked())
    {
        val=ui->leSomme->text().toDouble(&ok);
        if (ok)
        {
            if(pCompte->ajouter(val))
            {
                QMessageBox::about(this,"Info","Transaction réalisée");
                ui->lbSoldeDuCompte->setText(QString::number(pCompte->donnerSolde() ));
            }
            else
                QMessageBox::warning(this,"Erreur","Transaction impossible");
        }
        else
            QMessageBox::warning(this,"Erreur","Saisir un nombre");
    }
    else
    {
        // A VOUS DE COMPLETER CE QUI MANQUE ICI

    }
}
```

Au début, le **GroupBox** n'est pas visible

Conversion d'une chaine de caractères Qt (**QString**) en nombre. Si ce n'est pas possible, ok vaudra "false" à la fin de l'appel de la méthode.

On rend visible le **GroupBox** et on désactive la possibilité de recréer le compte.