

Table des matières

1 - OBJECTIFS.....	2
2 - AVANT-PROPOS.....	2
3 - DÉCOUVRIR QT.....	2
4 - VÉRIFICATION DE L'INSTALLATION DE QT5.....	3
5 - PREMIÈRE APPLICATION.....	3
6 - AJOUT D' UN BOUTON À LA FENÊTRE PRINCIPALE.....	5
7 - DES APPELS PAR SIGNAUX ET SLOTS.....	5
8 - CRÉER UNE CLASSE QT.....	6
9 - GÉRER LE TEMPS : LE TIMER.....	8
10 - EXERCICE : LE CHRONOMÈTRE.....	12
11 - EXERCICE : LE CHRONOMÈTRE UTILISATION DE DESIGNER.....	13

1 - Objectifs

L'objet de ce TP est découvrir les concepts de base de la programmation des Interfaces Homme-machine en utilisant la librairie Qt. A la fin de ce TP vous serez capable de :

- Appréhender la structure générale d'une application Qt
- Compiler une application Qt (MOC)
- Utiliser quelques composants graphiques de base de la boîte à outil Qt
- Utiliser le mécanisme signal/slot de Qt
- Naviguer dans la documentation de Qt



: ce symbole indique une tâche à effectuer sur l'ordinateur. Cochez cette case quand le travail demandé est effectué.



: ce symbole indique qu'il faut répondre par écrit.

2 - Avant-propos

Qt est un framework C++ permettant de développer des applications graphiques multi-plateformes en se basant sur l'approche suivante : « Ecrire une fois, compiler n'importe où »

Avec l'environnement de développement de Borland C++ Builder, c'est la librairie VCL qui est utilisée et avec l'outil de Microsoft Visual C++ c'est la librairie MFC.

Les applications écrites en C++/Qt sont indépendantes du système d'exploitation utilisé. Une application réalisée avec Qt devrait tourner aussi bien sous X Window que sur Windows xx, ou Mac OSX ; à condition évidemment de disposer de Qt pour ces divers systèmes et de recompiler les sources.

L'un des premiers avantages de la bibliothèque Qt est donc la portabilité. Son second avantage est sa simplicité d'utilisation. Il suffit juste de savoir programmer en objet, car avec **Qt**, une application Qt, une fenêtre, un bouton ou une boîte de dialogue, tous sont des objets instanciés à partir de classe.

La bibliothèque **Qt** est composée de tous les composants (appelés widgets) nécessaires à la création d'interfaces graphiques, mais aussi des classes pour l'accès aux bases de données, à la lecture/écriture de fichier XML, la gestion des threads...

Qt propose une plate-forme de développement de haut niveau; le programmeur ne se soucie plus des détails de bas niveau de la librairie graphique du système cible.

KDE utilise **Qt** comme outil de développement pour réaliser un environnement graphique complet sous X (Window Manager et applications graphiques).

Qt existe sous deux formes : OpenSource (gratuite) pour le développement d'applications GPL et Commerciale (payante) pour le développement d'applications commerciales.

On trouvera tous les renseignements sur **Qt** à l'adresse suivante: <http://qt.nokia.com/>

3 - Découvrir Qt

Une application **Qt** est une application graphique qui va interagir avec l'utilisateur de cette application. Le concept de base d'une application graphique est le **widget**. Un **widget** est l'entité de base d'une interface graphique qui réagit aux actions d'un utilisateur: manipulation de la souris, du clavier et tout autre événement du système graphique.

Les **widgets** sont de forme rectangulaires sont organisés hiérarchiquement. Un **widget** particulier appelé *widget principale* ou *widget racine* se trouve au sommet de cette hiérarchie.

Du point de vue du programmeur, une application **Qt** est un objet de la classe **QApplication**. C'est cet objet qui se charge de la gestion des événements d'une application donnée: c'est la classe centrale de **Qt** qui reçoit de l'environnement graphique des événements et qui les transmet à un objet graphique particulier (*widget*).

Chaque programme ne comporte qu'un objet de type **QApplication**.

Les **widgets** de **Qt** sont des objets de la classe **QWidget**.

4 - Vérification de l'installation de QT5

La version actuelle de Qt est QT5.

Qt est-il installé sur votre machine ?

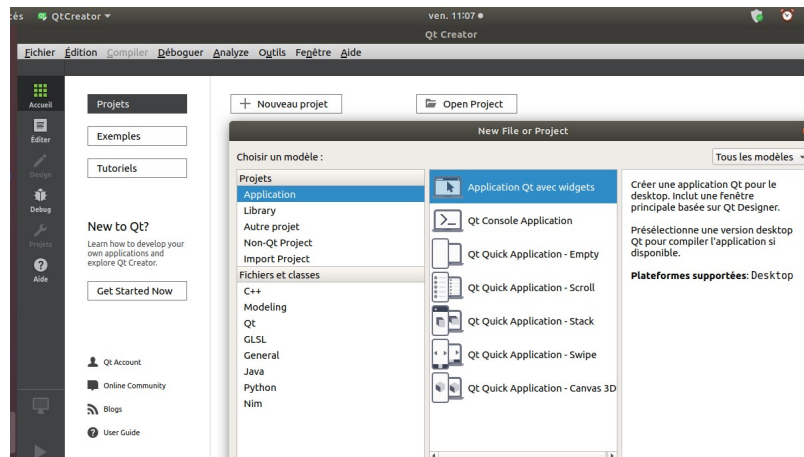
Dans `/usr/lib/x86_64-linux-gnu/`, vérifier que vous avez un fichier du type `libQt5Widgets.so.5.x.y`

les valeurs de x et de y indiquent la version installée sur votre machine.

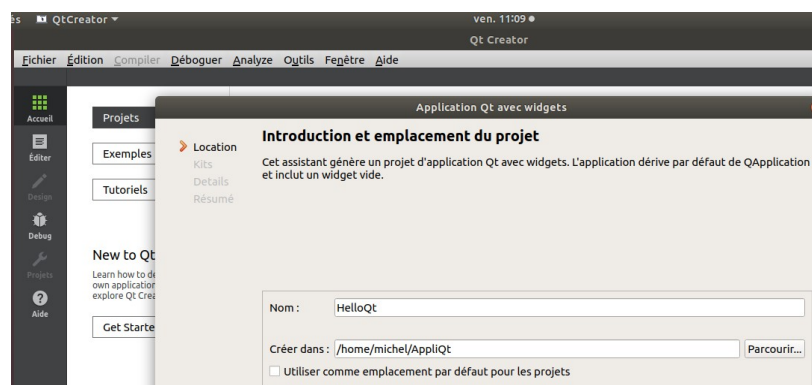
Si QT5 est installé préciser la version ?

5 - Première application

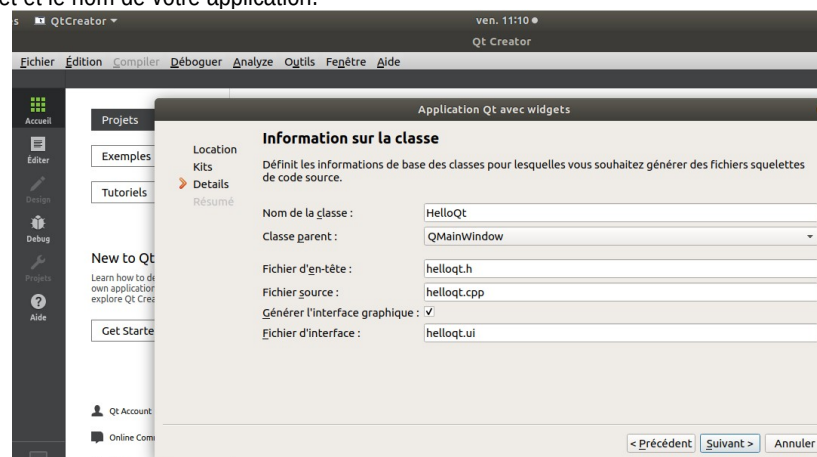
Lancer QtCréateur :



Choisir application avec Qt Widgets



Créer un répertoire vierge et le nom de votre application.




Analyse des 3 fichiers obtenus main.cpp, HelloQt.cpp et HelloQt.h.

 : Donner les relations de classes entre les classes HelloQt et QMainWindow?

.....

Compiler puis exécuter votre programme Ctrl-R ou flèche verte en bas à gauche.

 : Que se passe-t'il lors de l'exécution?


.....

 : Que se passe-t'il si vous supprimez uniquement la ligne "w.show();" ?

.....

 : Que se passe-t'il si vous supprimez uniquement la ligne "return a.exec();" ?

.....

 : Quelles sont les bibliothèques Qt incluses au linkage d'une application Qt (voir le Makefile) ?

.....

Utiliser la documentation de Référence de Qt

Qt propose une documentation pour l'aide à la programmation sous Qt : <http://doc.qt.io/>

Modifier le fichier HelloQt.cpp de la façon suivante :

```
#include "helloqt.h"
#include "ui_helloqt.h"
#include <QPushButton>
#include <QApplication>

HelloQt::HelloQt(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::HelloQt)
{
    ui->setupUi(this);
    this->setFixedSize(300,200);
    this->setWindowTitle("Hello Qt !");
    // déclaration du nouveau bouton
    QPushButton *pbQuitter = new QPushButton("Quitter",this);
    // Quand on clique sur le nouveau bouton , on change le texte de celui-ci
}

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

Parcourez la documentation afin de trouver la documentation de la classe QPushButton.

 : Que permet de faire cette classe et de quelle classe hérite t'elle?


.....

6 - Ajout d' un bouton à la fenêtre principale


Nous souhaitons ajouter sur la fenêtre de l'exemple précédent un bouton. Rien n'est plus simple que d'ajouter un bouton sur le widget principal : il suffit de savoir que la classe Qt permettant d'instancier un bouton poussoir est `QPushButton` et d'étudier sa documentation .

 Quel est sont les constructeurs de la classe `QPushButton` ?

.....

 Quel est le rôle du paramètre "`QWidget *parent`" dans un constructeur de la classe `QPushButton` ?

.....

 Quel est le rôle du paramètre "`const QString & text`" dans le constructeur de la classe `QPushButton` ?

.....


☐ Déclarez dans `HelloQt.cpp` une instance de la classe `QPushButton` appelée « `pbQuitter` » ayant « `Quitter` » pour label et appartenant au widget « `HelloQt` » . N'oubliez pas de d'inclure le fichier d'en-tête correspondant au bouton poussoir : `<QPushButton>` . Compilez et testez le programme.

La classe `QPushButton` dispose d'un certain nombre de méthodes dont, par exemple, celle qui définit la taille d'un widget et le place à une coordonnée dans sa fenêtre parente.

 Donnez la méthode permettant de créer un bouton de taille 60*30 placé à l'emplacement (125,170) de la fenêtre principale

.....

☐ Modifiez le programme et testez le résultat.

 Que se passe-t'il si vous cliquez sur le bouton "`Quitter`" ?

.....

7 - Des appels par signaux et slots


Si on désire provoquer la fin de l'application lorsque l'utilisateur clique sur le bouton « `Quitter` », rien n'est plus simple avec un mécanisme de signal et de slot. En effet, Qt propose un système d'appels entre objets original. Chaque objet Qt peut émettre des signaux, et peut être doté de slots. Le lien entre un signal et un slot se fait à l'aide de la simple instruction **connect**.

Par exemple, l'objet standard `QPushButton` est doté d'un signal nommé **clicked()** , et qui est déclenché lorsque l'utilisateur clique sur le bouton (soit à l'aide de la souris, soit en le validant au clavier, peu importe). De même, la fonction **quit()** de l'objet standard `QApplication` est en fait un slot. Il nous suffit donc d'ajouter à *fin du constructeur* la ligne de code suivante:

```
// on connecte le signal du bouton quit à l'alias de l'application qApp
connect( pbQuitter, SIGNAL(clicked()), qApp, SLOT(quit()) );
```

Ainsi, lorsque l'utilisateur cliquera sur le bouton, l'application prendra fin.

☐ Modifiez le programme et testez le résultat.

 Que se passe-t'il maintenant si vous cliquez sur le bouton "`Quitter`" ?


.....

Cette méthode établit une communication unidirectionnelle entre deux objets **Qt** . A chaque objet **Qt** (objet de la classe `QObject` ou d'une classe dérivée), on peut associer des *signaux* qui permettent d'envoyer des messages et des *slots* pour recevoir des messages. Notons que la classe `QWidget` est une classe dérivée de la classe `QObject` (comme d'ailleurs toutes les classes **Qt**).


La sémantique de cette instruction est : le signal « cliquer » effectué sur le widget « `pbQuitter` » est relié au slot « `quit()` » de l'application « `qApp` » de telle sorte que l'application s'arrête lorsque on clique sur le bouton.

Le système de communication établi en **Qt** est basée sur cette notion d'émission et réception de signaux. L'objet émetteur du signal n'a pas à se soucier de l'existence d'un objet susceptible de recevoir le signal émis. Ce qui permet une très bonne encapsulation et le développement totalement modulaire.

Un signal peut être connecté à plusieurs slots et plusieurs signaux à un même slot.

 Trouvez dans la documentation les deux autres signaux que peut émettre un QPushButton, testez le programme avec ces signaux et décrivez le résultat (nom du signal : résultat observé) :

.....

 Que se passe-t'il si vous écrivez « SIGNAL(click()) » à la place de SIGNAL(clicked()) dans l'instruction connect ?

.....

 Modifier votre programme afin que l'objet pbQuitter devienne un attribut privé de la classe HelloQt?

.....

8 - Créer une classe Qt

Il est bien aussi possible de définir ses propres signaux, et ses propres slots. Ainsi, si nous désirons ajouter un second bouton, et faire en sorte que lorsque l'on clique sur le premier bouton, le texte sur le second bouton soit modifié; nous allons pour cela créer un nouvel objet, dérivé du QPushButton. Voici à quoi ressemblera cette nouvelle classe:

```
//*****
// Fichier : MyButton.h
// Classe : MyButton
//*****

#ifndef MYBUTTON_H
#define MYBUTTON_H

#include <QPushButton>

class MyButton : public QPushButton
{
    Q_OBJECT
public:
    MyButton(const QString & text, QWidget *parent);
    virtual ~MyButton();
public slots:
    void slotChangeText();
};

#endif // MYBUTTON_H
```

Ainsi que vous le remarquez, les clauses **Q_OBJECT** et **slots:** ne sont pas habituelles. Ce sont en fait d'une part des macros qui seront donc insérées dans le .h par le préprocesseur, mais également des clauses qui seront interprétées par **le compilateur de méta-objets**.

Regardons de plus près, nous déclarons donc une nouvelle classe héritée de QPushButton, rien de nouveau de ce côté là. La clause Q_OBJECT indique que je suis en train de définir un méta-objet Qt, ensuite, je déclare un slot d'accès public, nommé "slotChangeText()" et qui n'admet pas de paramètre.

L'implémentation de la classe va, très classiquement, ressembler à la chose suivante :

```

//*****
// Fichier : MyButton.cpp
// Classe : MyButton
//*****

#include "MyButton.h"

MyButton::MyButton(const QString & texte, QWidget *parent) :
QPushButton(texte, parent)
{
}

MyButton::~MyButton()
{
}

void MyButton::slotChangeText()
{
    if (text() == QString("Docteur Jeckyl"))
        setText(QString("Mister Hide"));
    else
        setText(QString("Docteur Jeckyl"));
}

```

On rajoute notre nouveau bouton à l'application principale dans HelloQt.cpp (n'oubliez pas l'inclusion du fichier d'entête!). :

```

// déclaration du nouveau bouton
MyButton *myb = new MyButton("Docteur Jeckyl",this);
myb->setGeometry(125, 70, 120, 30);
// Quand on clique sur le nouveau bouton , on change le texte de celui-ci
connect(myb,SIGNAL(clicked()),myb,SLOT(slotChangeText()));

```

❑ Créez les fichiers MyButton.h et MyButton.cpp. Modifiez les programmes HelloQt.cpp HelloQt.h. Compilez et testez l'exécution.


Dans le fichier HelloQt.h faire la déclaration de l'objet mybutton de la classe MyButton.

Instancier de façon dynamique mybutton dans le fichier HelloQt.cpp.

Placer en position 200,170 le bouton.

Faire en sorte que le slot slotChangeText() de MyButton soit appelé lorsque l'on clique sur le bouton.

 Où est généré le fichier moc_MyButton.cpp ?

 Quelles modifications sont à apporter pour que l'appui sur "Quitter" provoque le changement de nom du bouton "Docteur Jeckyl" et que l'appui sur le bouton "Docteur Jeckyl" stoppe l'application ?

.....

Lorsqu'un objet émet un signal, il ne se soucie pas de savoir si un (ou plusieurs) objets sont connectés à ce signal. S'il y en a plusieurs, chaque objet effectuera le traitement qui lui est affecté (à moins qu'un mécanisme de blocage de signal ne soit volontairement implémenté), s'il y en a aucun, le signal ne sera pas traité, ce n'est pas un problème. Ainsi, **l'objet qui émet le signal n'a pas besoin de connaître quoique ce soit sur les objets qui traiteront éventuellement ces signaux**. Et il suffit d'une seule ligne de code pour résoudre l'appel: connect (), et une autre pour lancer un signal: emit()!

9 - Gérer le temps : le timer

La classe `QTimer` propose des signaux permettant de déclencher à intervalles réguliers des signaux `timeout`. Cet intervalle est précisé lors du démarrage d'un objet `QTimer` à l'aide la méthode `start(int t)` où `t` est l'intervalle de temps en millisecondes.

Cette émission de signaux peut être arrêtée à tout moment par l'invocation de la méthode `stop()`.

La méthode `changeInterval(int t)` permet de modifier l'intervalle du temps de déclenchement des signaux `timeout` et la méthode `isActive()` retourne la valeur vraie si l'objet `QTimer` est actif et faux sinon.

Pour illustrer cette classe, nous allons créer une application qui affiche, sur le terminal, la chaîne de caractère `...top` toutes les secondes dès que l'on clique sur le bouton `start`. Cet affichage s'arrête en cliquant sur le bouton `stop`.

Créez l'arborescence suivante dans le dossier de votre « home » un nouveau projet qui utilise un `QMainWindow` `MyTimer` :

☐ Générez la classe `MyTimer` dans le dossier `Classes` et complétez le fichier de déclaration `MyTimer.h` conformément au listing ci-dessous

En définissant les slot:

```
void slotStart();
void slotStop();
void slotTimeOut();
```

Les attributs privées associées aux différent boutons poussoir et à la classe `Qtimer` :

```
QTimer *mon_Timer
```

La classe `MyTimer` doit contenir les slots `slotStart()` et `slotStop()` permettant de déclencher et interrompre l'affichage de la chaîne `...top`.

La méthode `void slotStart()` va tout simplement démarrer le `timer` et fixer la fréquence d'émission du signal `timeout` à 1000 ms.

```
void MyTimer::slotStart()
{
    mon_timer->start(.....);
}
```

Quant à la méthode `void slotStop()` va stopper le `timer_` et donc donc stopper les émission des signaux `timeout`.

```
void MyTimer::slotStop()
{
    mon_timer->stop();
}
```

Enfin, la méthode `void slotTimeOut()`, qui est appelée chaque fois que le `timer` envoie un signal `timeout`, afficher sur le terminal la chaîne de caractère `"...top"`.

```
void MyTimer::slotTimeOut()
{
    qDebug() << "...top" ;
}
```

D'où le programme suivant:

```
/**
 * Copyright (C) 2011 (@appert44.org)
 * @file MyTimer.cpp
 * @brief Classe MyTimer exemple Qt
 */

// Includes qt
#include <ui_mytimer.h >
#include <QPushButton>
```



```

#include <QFont>

// Includes application
#include "MyTimer.h"

/**
 * Constructeur
 */

MyTimer::MyTimer(QWidget *parent) : QMainWindow(parent), ui(new Ui::MyTimer)
{
    // on fixe les dimension du widget
    setMinimumSize(190,100);
    setMaximumSize(190,100);

    // création du timer
    mon_timer = new QTimer(this);

    // connexion du signal timeout du timer au slot slotTimeOut
    connect(mon_timer, SIGNAL(timeout()), this, SLOT(slotTimeOut()));

    // création et connexion du bouton start
    pbStart = new QPushButton("Start", this);
    pbStart->setGeometry(20,10,70,30);
    pbStart->setFont( QFont("Times",18,QFont::Bold));
    .....

    // création et connexion du bouton stop
    pbStop = new QPushButton("Stop", this);
    pbStop->setGeometry(90,10,70,30);
    pbStop->setFont( QFont("Times",18,QFont::Bold));
    .....

    // création et connexion du bouton start
    pbQuit = new QPushButton("Quit", this);
    pbQuit->setGeometry(50,50,70,30);
    pbQuit->setFont( QFont("Times",18,QFont::Bold));

    // on connecte le signal du bouton quit à l'alias de l'application qApp
    .....
}

/**
 * Destructeur
 */
MyTimer::~MyTimer()
{
    delete mon_timer;
}

```

☐ Complétez le fichier de définition MyTimer.cpp conformément au listing ci-dessus. Compilez et tester le programme ci-dessus. Testez quelques modifications simples.

10 - Exercice : Le chronomètre

Nous allons à présent utiliser les concepts que l'on vient de voir pour programmer un chronomètre. Pour réaliser cette application, nous utiliserons un nouveau widget appelé `QLCDNumber` capable d'afficher un nombre de manière esthétique: c'est encore un nouveau widget



☐ En vous inspirant de l'exemple précédent, proposez les fichiers `Chronometre.h`, `Chronometre.cpp` réalisant le chronomètre présenté ci-dessus.

A chaque signal `timeout` émis, nous allons incrémenter un compteur `interrupt` ; il contiendra donc le nombre de fois que le signal `timeout` a été émis. Il suffit donc d'afficher ce nombre régulièrement pour voir défiler les unités de temps sur notre chronomètre. En disposant d'un timer qui émet un signal toutes les 10 ms, l'attribut `interrupt` contiendra le temps écoulé, en 1/100 de secondes) depuis le *click* sur le bouton *Start* .

Remarques :

- Il est nécessaire de doter notre classe `Chronometre` d'un signal permettant de se « brancher » sur le slot permettant d'afficher un nombre sur le widget `QLCDNumber`.
- la déclaration d'un nouveau signal se fait avec la déclaration suivante dans le fichier d'entête :

```
signals :
    void nomSignal(type du paramètre éventuel);
```

- l'utilisation du signal déclaré ci-dessus se fait dans le fichier d'implémentation (.cpp):

```
...
// on émet le signal avec son paramètre éventuel
emit nomSignal(paramètre éventuel);
...
```

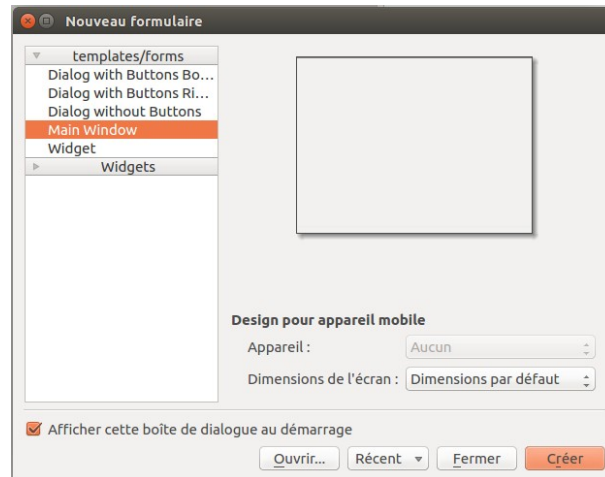
☐ Exercice complémentaire : rajoutez un bouton *Pause* au chronomètre.

11 - Exercice : Le chronomètre Utilisation de Designer

Nous allons refaire le chronomètre en utilisant le designer de Qt.

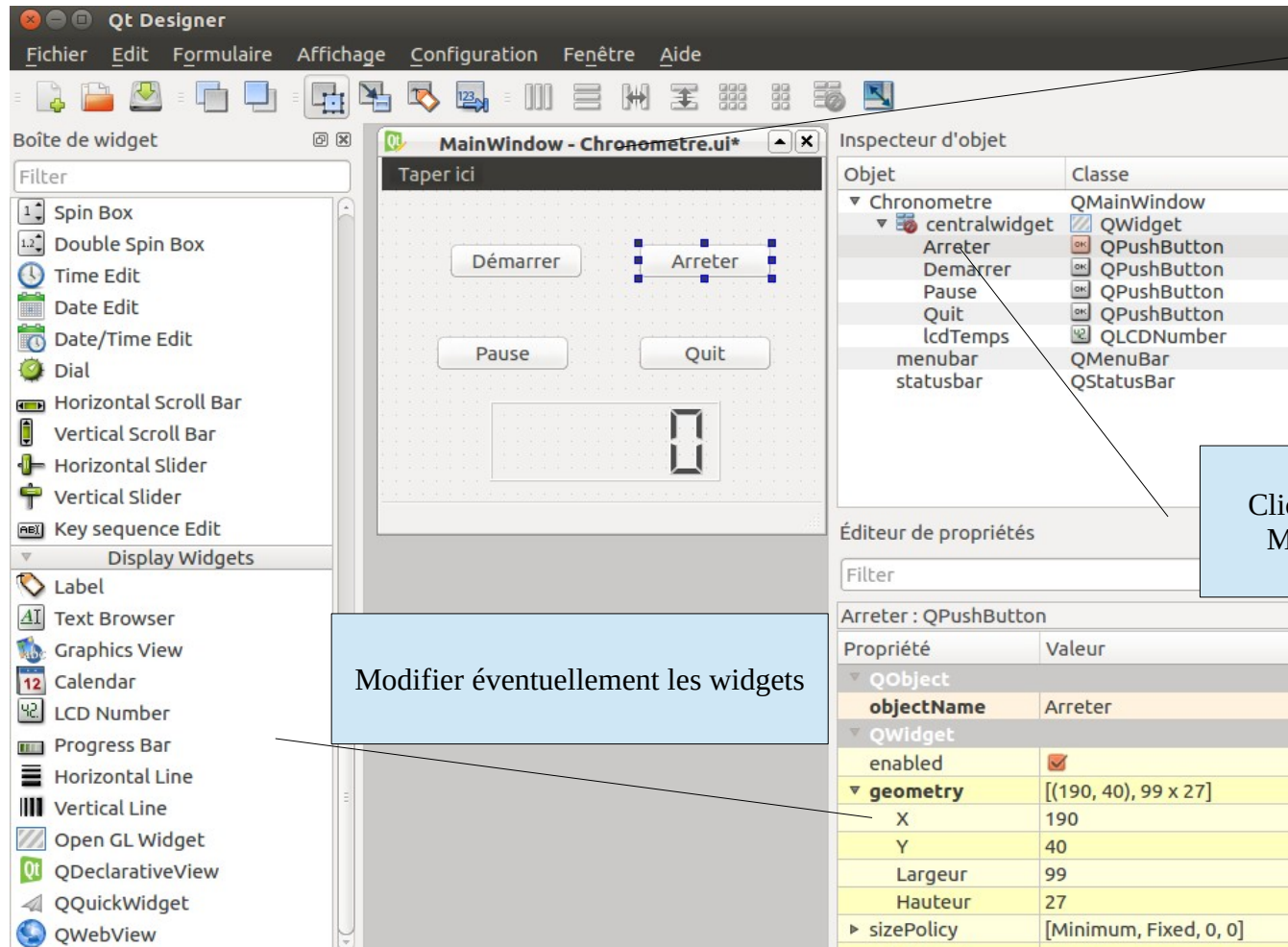
Réalisation du fichier Chronomètre.ui

Dans le Designer choisir un projet MainWindow.



Réaliser votre boîte de dialogue « MainWindows »

Il est intéressant de faire Enregistrer tout avant de sortir.



Enregistrer dans
le répertoire
Classes
Chronometre.ui

Click droit sur le bouton
Modifier ObjectName

Modifier éventuellement les widgets

Lancer la compilation de votre projet

Vérifier qu'à l'exécution la fenêtre apparaît.

Ouvrir le fichier ui_Chronometre.h et vérifier l'instanciation des différentes classes associées aux différents boutons et au LCD.