Université de la Rochelle
Licence d'Informatique - IHM (INFO132601)

TD 3 - *Introduction à la programmation Qt*

© B. Besserer, R. Péteri

Année universitaire 2015-2016

1 Programmation IHM haut niveau avec Qt

1.1 Généralités

1. Pour une compilation correcte d'un projet utilisant Qt, dans quel ordre les commandes doivent-elles être tapées ?

   ☐ qmake -project, make, qmake

   ☐ make, qmake -project, make

   ☐ qmake -project, qmake, make

   qmake -project, qmake, make

2. Quelle est le nom de la classe que toute application Qt doit instancier ?

   ☐ QClass

   ☐ QApplication

   ☐ QPushButton

   QApplication. QApplication object manage application-wide resources. The QApplication constructor requires argc and argv because Qt supports a few command-line arguments of its own.

3. Quelle méthode de la classe QPushButton permet de provoquer l'affichage du bouton ?

   ☐ open()

   ☐ show()

   ☐ run()

   show()

4. Soit une classe dérivée de la classe QWidget. Cette nouvelle classe doit réagir à un action de type "clic avec le bouton gauche de la souris". Parmi les propositions suivantes, laquelle n'est pas correcte :

   ☐ Redefinir la méthode mousePressEvent(...)

   ☐ Relier le signal mousePressEvent(...) avec un slot

   ☐ Etudier les paramètres transmis par mousePressEvent()

   Relier le signal mousePressEvent(...) avec un slot
   *virtual void mousePressEvent (QMouseEvent * event) : fonction virtuelle de la classe QWidget. The QMouseEvent class contains parameters that describe a mouse event.*

5. Quelle instruction permet la génération d'un SIGNAL (au sens Qt) :

   ☐ `QObject::connect(...)`,

   ☐ `QTrumpet::taratata(...)`

   ☐ Avec la macro Qt `emit`

   ☐ `QSocket::sendQtSignal(...)`

Avec la macro Qt `emit`. signals only emit themself and their arguments, dont need code to work. You define what they should do on connections. A Signal is a method that is emitted rather than executed when called. So we only declare prototypes of signals that might be emitted.
A Slot is a member function that can be invoked as a result of signal emission. We have to tell the compiler which method has to be treated as slots by putting them in one of the following sections : public slots, protected slots or private slots.

6. Un utilisateur peut créer des widgets personnalisés (Custom Widgets). Pour cela :
   - ☐ Il définira une classe dérivée de `QApplication`
   - ☐ Il définira une classe dérivée de `QWidget`
   - ☐ Il peut définir une classe dérivant de n'importe quelle classe Qt

Il peut définir une classe dérivant de n'importe quelle classe Qt. Ex : QHBoxLayout->QBoxLayout->QLayout->QObject

1.2 Application minimale avec Qt

Que fait le code suivant ? (ajoutez les commentaires ; précisez où se situe l'entrée dans la boucle de gestion de message).

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

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel *hello = new QLabel("<font color=blue>Hello World !</font>",0);
    app.setMainWidget(hello);

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

    return app.exec();
}
```

Lines 1 and 2 include the definitions of the QApplication and QLabel classes. For every Qt class, there is a header file with the same name (and capitalization) as the class that contains the class?s definition.

```
// fonction principale
int main( int argc, char **argv)
{
  // creation d'une instance de la classe QApplication permettant de
  // prendre en compte les différents arguments -geometry -display via argc, argv ...
  // c'est aussi une variable globale accessible sous le nom qApp
  QApplication app( argc, argv );
  // creation d'un QLabel. Le texte affiché est formaté à l'aide d'un
  // sous-ensemble d'HTML (voir $QTDIR/doc/html/qstylesheet.html)
  QLabel *hello = new QLabel("<font color=blue>Hello World !</font>",0);
  // définition du Widget principal qui va contenir les autres widgets
  app.setMainWidget(hello);
  // modification des propriétés du QLabel pour qu'il puisse s'afficher. Widgets are always created hidden, so
can customize them before showing them, thereby avoiding flicker.
  hello->resize(100, 30);
  hello->show();
  // lancement de la boucle principale (gestion et traitement des
  // interruptions, ...)
  // cette boucle ne se termine que sur l'appel de app.exit()
  // ou la destruction du widget
  return app.exec();
}
```

1.3 Première utilisation de Signal et Slot

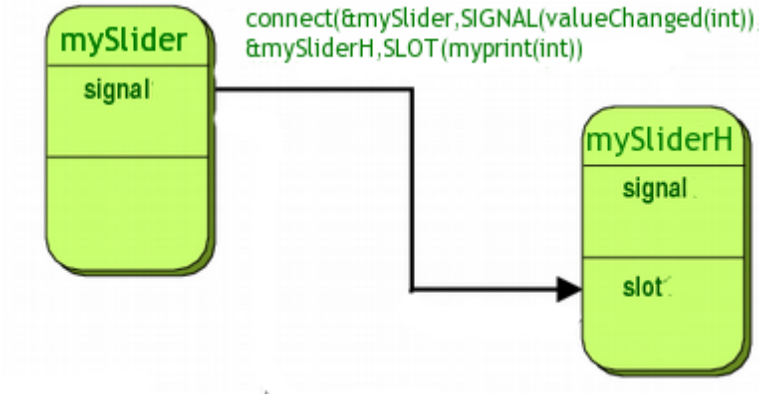
Vous devez écrire une application Qt affichant un contrôle de type Slider (réglage linéaire, potentiomètre, ...), de valeur minimale 0 et maximale 100. Dès que la valeur est modifiée, cette valeur doit être affichée dans la console (avec `printf` ou `cout`).

Pour réaliser cela, nous allons définir un objet Qt qui recevra l'information lors d'un changement d'état du curseur du Slider et qui en assurera l'affichage.

```
// Fichier mySliderHandler.h
#include <iostream>
#include <qobject.h>
#include <qapplication.h>
#include <qslider.h>
using namespace std;
class mySliderHandler : public QObject{
Q_OBJECT
public
slots:
...
};

// Fichier main.cpp
#include "mySliderHandler.h"
int main(int argc, char **argv){
QApplication a(argc, argv);
QSlider mySlider ...
...
a.setMainWidget(...);
mySlider.show();
return a.exec();
}
```

1. Qu'est ce que l'instruction `Q_OBJECT` ?
   Q_OBJECT est une macro au début de la définition de classe et est nécessaire pour toutes les classes qui définissent des signaux ou des slots. La macro Q_OBJECT doit apparaître dans la section privée de la définition d'une classe qui déclare ses propres signaux et slots et/ou qui utilise d'autres services fournis par le système de méta-objets de Qt. moc doit être exécuté sur les classes qui se servent de la macro Q_OBJECT
2. Décrivez à l'aide d'un schéma le mécanisme que vous allez implémenter (2 objets : une instance de l'objet Slider et une instance de l'objet mySliderHandler, ainsi que les mécanismes Qt d'échange d'information (signal, slot) qui seront mis en oeuvre.



3. Complétez le programme donné ci-dessus.

```
// Fichier mySliderHandler.h
#include <iostream>
#include <qobject.h>
#include <qapplication.h>
#include <qslider.h>
using namespace std;

class mySliderHandler : public QObject{
Q_OBJECT

public
slots:
      void myprint(int);
};


// Fichier mySliderHandler.cpp
#include "mySliderHandler.h"

void mySliderHandler::myprint(int i)
{
 printf("Valeur du Slider: %d \n", i);
```

```
}


// Fichier main.cpp
#include "mySliderHandler.h"

int main(int argc, char **argv){
QApplication a(argc, argv);
QSlider mySlider;
mySliderHandler mySliderH;

mySlider.setRange(0,100);
mySlider.setValue(20);
mySlider.show();

QObject::connect(&mySlider, SIGNAL(valueChanged(int)),&mySliderH,SLOT(myprint(int)));
return a.exec();
}
```

4. Donnez la séquence de commandes à entrer dans la console Linux pour générer le programme exécutable.
   qmake -project, qmake, make

2 Comptage de clics avec Qt

On souhaite créer un bouton de type PushButton disposant d'une fonctionnalité particulière : celle de compter les clics fait sur ce bouton depuis le lancement de l'application.

1. Commencez par écrire une fichier ButtonCount.h et un fichier ButtonCount.cpp qui implemente une classe ButtonCount, dérivée de QPushButton, et disposant d'une variable membre m_Count. On redefiniera le constructeur `QPushButton ( const QString & text, QWidget * parent = 0 )`, en y ajoutant la mise à zero de la variable.

   ButtonCount.h

```
#ifndef BUTTONCOUNT_H
#define BUTTONCOUNT_H

#include <QObject>
#include <QPushButton>

class ButtonCount : public QPushButton
{
Q_OBJECT
public:
ButtonCount( const QString & text, QWidget *parent = 0 );
protected:
int m_Count;
};
#endif
```

   ButtonCount.cpp

```
#include <QtGui>
#include "ButtonCount.h"

ButtonCount::ButtonCount(const QString & text, QWidget * parent) : QPushButton(text,parent)
{
m_Count = 0;
}
```

2. Maintenant, definissons un SLOT nommée `Increment()` (c'est à dire une méthode) permettant d'incrémenter la variable de comptage à chaque clic. Le classe QPushButton possède déjà un signal `void clicked()`, hérité de QAbstractButton. Afin d'exploiter ce signal, l'objet ButtonCount va connecter ce signal à son propre `slot Increment()`. La connection sera faite dans le constructeur. Ainsi, à chaque clic, la variable `m_Count` doit s'incrementer.

   Completer votre fichier ButtonCount.h et le fichier ButtonCount.cpp pour déclarer le slot, ecrire la methode correspondante et modifier le constructeur pour assurer la connection signal → slot.

   ButtonCount.h

```
   #ifndef BUTTONCOUNT_H
    #define BUTTONCOUNT_H

#include <QObject>
#include <QPushButton>

class ButtonCount : public QPushButton
{
Q_OBJECT
public:
ButtonCount( const QString & text, QWidget *parent = 0 );
protected:
int m_Count;
public slots:
void Increment(void);
};
#endif
```

ButtonCount.cpp

```
#include <QtGui>
#include "ButtonCount.h"

ButtonCount::ButtonCount(const QString & text, QWidget * parent) : QPushButton(text,parent)
{
m_Count = 0;
// connection du signal (existant pour la classe QPushButton) a notre slot
connect(this, SIGNAL(clicked()), this, SLOT(Increment()));
}

// un slot s'écrit comme une simple methode
void ButtonCount::Increment()
{
m_Count++;
}
```

3. Incrementer une variable sans pouvoir la consulter, ce n'est pas très interessant. Qt dispose de nombreux widgets permettant d'afficher des informations, comme par exemple le widget QLCDNumber.

The QLCDNumber widget displays a number with LCD-like digits. It can display a number in just about any size. It can display decimal, hexadecimal, octal or binary numbers. It is easy to connect to data sources using the display() slot, which is overloaded to take any of five argument types.
Parmis les types acceptés par le slot display(), nous avons :

```
void display ( const QString & s )
void display ( double num )
void display ( int num )
```

On peut donc envoyer vers cet objet une chaine de caractères, un nombre en virgule flottante ainsi qu'un nombre entier.
Revenons à notre objet ButtonCount. Nous allons ajouter à cet objet un signal. Par souci de cohérence avec d'autres contrôles de Qt, nous allons nommer notre signal `valueChanged(int)`. A chaque increment de la variable `m_Count`, un signal `valueChanged(int)` sera émis, avec la valeur courante de la variable `m_Count`. Modifez votre code pour ajouter cette fonctionnalité.

ButtonCount.h

```
#ifndef BUTTONCOUNT_H
#define BUTTONCOUNT_H

#include <QObject>
#include <QPushButton>

class ButtonCount : public QPushButton
{
Q_OBJECT
public:
ButtonCount( const QString & text, QWidget *parent = 0 );
protected:
int m_Count;
signals:
  void valueChanged(int);
public slots:
void Increment(void);

};
#endif
```

ButtonCount.cpp

```
#include <QtGui>
#include "ButtonCount.h"

ButtonCount::ButtonCount(const QString & text, QWidget * parent) : QPushButton(text,parent)
{
m_Count = 0;
```

```
connect(this, SIGNAL(clicked()), this, SLOT(Increment()));
}

void ButtonCount::Increment()
{
m_Count++;
// la ligne suivante assure un feedback sonore
QApplication::beep();
// emission du signal
emit(valueChanged(m_Count));
}
```

4. Soit les lignes suivantes dans un programme principal :

```
ButtonCount *button_c = new ButtonCount("Je compte mes clics");
  QLCDNumber *lcd = new QLCDNumber(2);  // ici on indique le nombre de digits
  lcd->setSegmentStyle(QLCDNumber::Filled);  // options d'affichage
```

   Ajoutez la ligne permettant à l'afficheur LCD d'afficher le nombre de clics effectués sur le bouton.

```
connect(button_c, SIGNAL(valueChanged(int)),lcd, SLOT(display(int)));
```

A Annexes

Methodes de la classe QtSlider (QtSlider dérive de QtAbstractSlider)

`QSlider::QSlider ( Qt::Orientation orientation, QWidget * parent = 0 )`

Constructs a slider with the given parent. The orientation parameter determines whether the slider is horizontal or vertical ; the valid values are Qt : :Vertical and Qt : :Horizontal.

`void QAbstractSlider::setRange ( int min, int max )`

Sets the slider's minimum to min and its maximum to max. If max is smaller than min, min becomes the only legal value.

`void QAbstractSlider::valueChanged ( int value )  [signal]`

This signal is emitted when the slider value has changed, with the new slider value as argument.

Rappel :

`QObject::connect(obj1, SIGNAL(f1(parList)), obj2, SLOT(f2(parList)));`

obj1 et obj2 sont des pointeurs.

Class QLCDNumber

The QLCDNumber widget displays a number with LCD-like digits.
Public Slots :

```
void display ( const QString & s )
void display ( int num )
void display ( double num )
virtual void setHexMode ()
virtual void setDecMode ()
virtual void setOctMode ()
virtual void setBinMode ()
virtual void setSmallDecimalPoint ( bool )
```

QWidget Public Slots :

```
bool close ()
void hide ()
void lower ()
void raise ()
void repaint ()
void setDisabled ( bool disable )
void setEnabled ( bool )  // An enabled widget handles keyboard and mouse events; a disabled widget does not
void setFocus ()
void setHidden ( bool hidden )
void setStyleSheet ( const QString & styleSheet )
virtual void setVisible ( bool visible )
void setWindowModified ( bool )
void setWindowTitle ( const QString & )
void show ()
void showFullScreen ()
void showMaximized ()
void showMinimized ()
void showNormal ()
void update ()
```

QAbstractButton

Public Slots

```
void animateClick ( int msec = 100 )
void click ()
void setChecked ( bool )
void setIconSize ( const QSize & size )
void toggle ()
19 public slots inherited from QWidget
1 public slot inherited from QObject
```

Signals

```
void clicked ( bool checked = false )
void pressed ()
void released ()
void toggled ( bool checked )
1 signal inherited from QWidget
1 signal inherited from QObject
```

QApplication

QApplication -> QCoreApplication.
The QCoreApplication class provides an event loop for console Qt application

```
Public Slots

void quit ()
```