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

TP 2 - *Programmation Qt*

© B. Besserer, R. Péteri

Année universitaire 2016-2017

Affichage double et conversion de température

Exercice 1

Le but de ce premier exercice est de créer une classe `AffichT` gérant un widget (la classe `AffichT` hérite de la classe `QWidget`) contenant un couple de widgets : un widget de type `QProgressBar` et un widget de type `QSpinBox`.

Le widget `QSpinBox` (placé sous le `QProgressBar`) permet de modifier une valeur numérique directement depuis le clavier ou bien en utilisant les ascenseurs verticaux et horizontaux. Le widget `QProgressBar` gère l'affichage d'une valeur numérique. Outre son constructeur et son destructeur, la classe `AffichT` comporte la méthode `Reset`, qui permet de remettre la valeur du SpinBox et de la ProgressBar à 0. Le fichier `AffichT.h` est donné ci-après.

```
#ifndef AFFICHT_H
#define AFFICHT_H

#include <QSpinBox>
#include<QProgressBar>

class AffichT : public QWidget
{
Q_OBJECT
public:
AffichT(); // constructeur
~AffichT(); // destructeur
void Reset();
private:
QProgressBar *m_TheProgressBar;
QSpinBox *m_TheSpinBox;
};
#endif
```

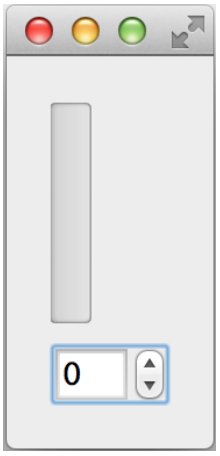


Fig.1: widget AffichT

Récupérez sur Moodle le fichier `tp2_exo1_etudiants.zip` et décompressez-le. On partira d'un projet Qt vide, et vous créerez votre fichier .pro. Complétez le fichier `AffichT.cpp` qui gère la géométrie des widgets.

- Pour compléter le constructeur :
  - Instanciez les deux widgets membres de la classe `AffichT` : `m_TheSpinBox` et `m_TheProgressBar`, respectivement de type `QSpinBox` et `QProgressBar`. `m_TheProgressBar` sera orientée verticalement (méthode `setOrientation`)
  - Ces deux widgets doivent être alignées l'une au-dessous de l'autre en utilisant un gestionnaire de géométrie de type `QVBoxLayout`. Pour gérer la géométrie, une fois créé le `QVBoxLayout` (nommé layout) et une fois les deux widgets gérées par ce gestionnaire (grâce à `addWidget`), n'oubliez pas d'utiliser `this->setLayout(layout);`
  - Ces deux contrôles doivent être connectés, à savoir qu'un changement dans la valeur de la SpinBox doit entraîner une mise à jour de la valeur de la ProgressBar. Pour cela, utilisez le signal `valueChanged(int)` et le slot `setValue(int)`, définis sur les widgets `QSpinBox` et `QProgressBar`.
- Pour compléter le destructeur :

— Supprimez les deux widgets `m_TheSpinBox` et `m_TheProgressBar`.
— Complétez la méthode Reset.


1 A l'issue de cet exercice, vous devez obtenir à l'execution une fenêtre ressemblant à celle de la figure 1, dans laquelle les deux widgets sont connectés. Par défaut, la méthode Reset n'est jamais appelée.

Exercice 2

Le but de ce deuxième exercice est de créer une classe `myMainWindow` héritant de `QMainWindow` et comprenant :

— Une barre des menus comprenant deux menus : `File` et `Display` (de type `QMenu`)
— Une zone d'affichage (`QLabel`) servant à l'affichage des messages d'erreur notamment pour débugger le programme
— Un widget de type `AffichT` comprenant deux widgets de type `QProgressBar` et `QSpinBox` connectés
— Un bouton poussoir (`QPushButton`) servant à appliquer un slot appelé `clear()`

Voici un aperçu de l'aspect possible de votre widget (ici sur Mac OS, la barre de menu est hors de la fenêtre principale).

Récupérez sur Moodle le fichier `tp2_exo2_etudiants.zip` et décompressez-le dans un nouveau dossier. Copiez-collez dans le dossier ainsi créé les fichiers `AffichT.h` et `AffichT.cpp` que vous avez créé à l'exercice 1.
Avec le code qui vous est fourni, la barre des menus ne comporte qu'un seul menu "File" ne contenant que l'article de menu `Open` (de type `QAction`).

1. Complétez le fichier `myMainWindow.cpp`. Pour cela :
   — Complétez le constructeur :
     — Créez le widget `QLabel` déclaré dans myMainWindow.h. Pour gérer sa mise en forme, utilisez les méthodes `setFrameStyle` et `setAlignment`.
     — Créez les widgets `AfficheT` et `Pushbutton` déclarées dans `myMainWindow.h`.
     — Créez un objet conteneur appelé vbox et de type `QVBoxLayout`. vbox contiendra les trois widgets précédemment créés.
   — Complétez le destructeur : Supprimez les trois widgets `QLabel, AfficheT et PushButton`.
   — Complétez la définition de la méthode `open`. Celle-ci ne fera qu'afficher dans le widget de type `QLabel` le texte "Invoked File/Open". Pour cela, utilisez la méthode `setText` définie sur la classe `QLabel`.
   — De la même manière, complétez la définition de la méthode clear. Celle-ci ne fera qu'afficher dans le widget de type `QLabel` le texte "Button clear pressed".

— Complétez la définition de la méthode `createActions`. Pour cela connectez la sélection de l'article de menu Open (signal `triggered()`) au slot défini à partir de la méthode `open()`.
— Connectez le bouton `QPushButton` à la méthode clear() qui sera déclenchée lors de l'appui sur le bouton.

2. Modifiez les fichiers `myMainWindow.cpp` et `myMainWindow.h` de manière à :

— Rajouter au menu `File` un article de menu appelé `Save` qui ne fera qu'afficher dans le widget de type QLabel le texte "Invoked File/Save". Cet article de menu sera associé au raccourci clavier Ctrl+S.
— Rajouter au menu File un article de menu appelé `Exit` qui fermera l'application (méthode `quit()` de `qApp`) et sera associé au raccourci clavier Ctrl+Q. L'article de menu `Exit` sera séparé de l'élément `Save` par un séparateur (ligne horizontale grise).
— Rajouter à la barre des menus un menu appelé `Display`. Rajoutez dans ce menu l'article de menu appelé "Reset to defaults..." et qui fera s'afficher dans le widget de type `QLabel` : "Invoked Display/Reset to defaults"  2

Exercice 3

Le but de ce troisième exercice est de modifier le code existant de manière à ce que `AfficheT` contienne deux couples ProgressBar-SpinBox, alignés horizontalement. Le couple de widgets de gauche affiche la température en degrés Celsius, le second en degrés Fahrenheit. Pour l'alignement, utilisez un conteneur de type QGridLayout (au lieu du QVBoxLayout utilisé précédemment).

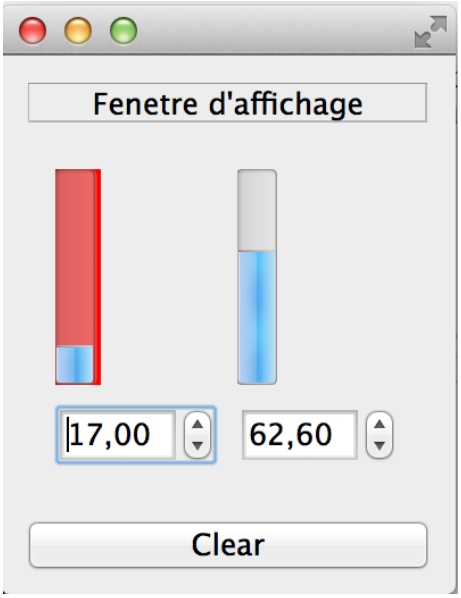
• Les widgets devront être interconnectés de manière à ce qu'un changement dans la température en degrés Celsius se répercute sur un changement en degrés Fahrenheit. Plus précisément, la SpinBox de gauche (Celsius) envoie une valeur à sa ProgressBar et à la SpinBox de droite (Fahrenheit) qui, à son tour, envoie un signal à sa ProgressBar. Pour gérer la conversion, il faudra créer votre propre ProgressBar, que vous appellerez `myProgressBar`, avec le nouveau slot `SetValueC2F`, basé sur le slot prédéfini `SetValue` et qui aura pour effet de convertir la valeur.

Pour passer des degrés Fahrenheit aux degrés Celsius, il faut soustraire 32 puis multiplier le résultat par 5/9.
Attention au arrondis implicites !

Remarque : pour les perfectionnistes (bien !), il peut être judicieux d'utiliser dans la classe `AfficheT` la classe `QDoubleSpinBox` pour les problèmes d'arrondi. Une widget de type `QDoubleSpinBox` est la version avec valeurs de type `double` de la widget `QSpinBox`.

• Faites en sorte que le choix de l'article de menu `Display -> reset to defaults` appelle la méthode `Reset()` définie sur la classe `AfficheT`.

Voici un aperçu de ce que vous devez obtenir, avec en plus un seuil d'alerte (qui sera mis en place dans l'exercice 5).



3

Exercice 4 — Mise en place d'un timer

4 Rajoutez un timer (`QTimer`) qui augmente de un, à chaque seconde, le nombre de degrés Celsius.

Exercice 5 — Seuil d'alerte

Faites en sorte qu'à partir d'un seuil fixé (déclarer une variable membre qui prendra - pour l'instant - une valeur fixe lors du constructeur), la ProgressBar des degrés celsius devienne rouge. Un beep sera émis au franchissement de ce seuil. Il faut implémenter dans la classe `myProgressBar` une modification des valeurs de la **palette** pour changer la couleur de la ProgressBar sans toucher aux routines graphiques déjà implémentés.

5 Il vous faudra créer une palette "rouge" et l'affecter à la propriété palette du widget (méthode `setPalette`), on mettra aussi `setAutoFillBackground(true)`). On utilisera la méthode `QApplication::beep()` de l'application pour générer le son.

```
   #include <QtGui>

    #include "myDial.h"

    myDial::myDial()
    {
m_Threshold=37;
 //       setMinimum (0);
// setMaximum (200);
            }

/** Destructeur **/
myDial::~myDial()
{

}

        /** accesseur pour la donnée privée threshold **/
/* void myDial::SetThreshold(int value)
{
m_Threshold = value;
}
*/

void myDial::setValueC2F(int value)
{
               setValue(int(value*(9/5)+32));
             //  temp = value*(10/5)+32;
               printf("temp=%d",temp);
       //       int temp = (value*1.8);
         //               setValue(temp);
        }
```

```
void myDial::Increment()
{
int val=value();
setValue(val+1);
}

//Gestion du seuil
void myDial::Threshold(int value)
{
this->setAutoFillBackground(true);
QPalette palette = this->palette(  );
if (value>this->m_Threshold)
{
QApplication::beep();
palette.setColor(QPalette::Active, QPalette::Background, QColor(Qt::red));
this->setPalette( palette );
}
else
{
palette.setColor(QPalette::Active, QPalette::Background, QColor(Qt::lightGray));
this->setPalette( palette );
}
}

void myDial::setValueF2C(int value)
{
                              setValue(int(value-32)*(5/10));
             // temp = (value/4);
               //          setValue(temp);
                        }

/** surcharge de la methode pour le signal sonore **/
```

Pour ceux qui ont fini - un peu de DRAG & DROP

On peut activer des fonctionnalités de Drag & Drop pour les widgets. En guise d'exemple, on va positionner le seuil d'alerte du widget myProgressBar en sélectionnant à la souris une valeur (fragment de texte dans un éditeur par exemple) et en glissant cette valeur sur la ProgressBar.

Pour cela, dans la classe dérivée `myProgressBar`, vous devez surcharger au moins les deux méthodes :
`void dropEvent(QDropEvent *event);` et `void dragEnterEvent(QDragEnterEvent *event);`
Dans le fichier d'implémentation `myProgressBar.cpp`, cela ressemblera à :

```
void myProgressBar::dragEnterEvent(QDragEnterEvent *event)
{
  if (event->mimeData()->hasFormat("text/plain"))
    event->acceptProposedAction();
}


void myProgressBar::dropEvent(QDropEvent *event)
{
  if (event->mimeData()->hasFormat("text/plain"))
  {
    QString plainText = event->mimeData()->text();
    event->acceptProposedAction();

    // convertir plainText en int (si c'est possible) et l'affecter à la variable membre
    ...
  }
  else
  {
    event->ignore();
  }
}
```

et dans le constructeur de la classe `myProgressBar`, ajouter : `setAcceptDrops(true);`

6 Implémentez le Drag & Drop et faites de même en écrivant une valeur dans un fichier texte et en glissant le fichier texte sur la ProgressBar. Dans ce cas, la chaine plaintext contient le chemin d'accès vers ce fichier. A vous de l'ouvrir et d'en exploiter le contenu (pensez à utiliser `QFile`).