TP3 – Introduction à Qt

Partie I – Programme de base, slot/signal

1 – Éditez et compilez le programme ci-dessous avec Qt Creator.

Dans cet exercice, on créera un projet de type Qt Gui.
Vous devez enlever les classes par défaut.

```
#include <QApplication>
#include <QPushButton>

int main( int argc, char ** argv) {
    QApplication mainApp(argc, argv);

    QPushButton btnHello("Bonjour!", 0);
    btnHello.resize(200, 60);
    mainApp.setActiveWindow(&btnHello);
    btnHello.show();

    return mainApp.exec();
}
```

2 – Reliez le signal **clicked()** de **btnHello** avec le slot **quit()** de **mainApp** avec la méthode **QObject::connect()**.

Rappel de *syntaxe*:

QObject::connect*(adresse de l'objet 1,* ***SIGNAL****(signal avec paramètres de l'objet 1), adresse de l'objet 2,* ***SLOT****(slot avec paramètres de l'objet 2));*

Faites attention dans cet appel à déclarer les signaux et les slots avec leur prototypes.

Partie II – QObject

Implémentez la classe **Element** (dérivée de **QObject**) en ajoutant le fichier de code source (.cpp) et les fonctions demandées, puis testez le programme suivant.

Dans cet exercice, on créera un projet de type Qt Console. Pour manipuler le nom d'un ***QObject****, vous pouvez utiliser les fonctions* ***setObjectName()*** *et* ***objectName()*** *de* ***QObject****. Pour utiliser le même objet* ***cout*** *déclaré dans* ***main.cpp****, vous pouvez le déclarer comme* ***extern*** *dans* ***element.cpp****. La méthode* ***dumpObjectTree()*** *ne peut être utilisée que dans un projet compilé en mode Debug.*

Fichier ***element.h*** *(à vous d'écrire le fichier* ***element.cpp****)*

```
#ifndef ELEMENT_H
#define ELEMENT_H
```

```
#include <QObject>
#include <QString>
#include <QTextStream>

class Element : public QObject {
    Q_OBJECT
public:
    Element(QObject * parent = 0, QString name = "");
    ~Element();
};

#endif // ELEMENT_H
```

Fichier **main.cpp**

```
#include <QTextStream>
#include <cstdio>
#include "element.h"

QTextStream cout(stdout, QIODevice::WriteOnly);
void createGuiTree() {
    cout<<"Debut de la fonction"<<endl;

    // allocation statique, la memoire sera liberee a la fin
    Element guiTree(0, "Interface");

    // allocation dynamique des sous structures
    Element * mainWidget = new Element(&guiTree, "MainWidget");
    Element * menuBar = new Element(mainWidget, "MenuBar");
    Element * fileMenu = new Element(menuBar, "File");
    new Element(fileMenu, "QuitItem");
    new Element(menuBar, "Edit");
    new Element(menuBar, "Help");
    new Element(mainWidget, "StatusBar");

    cout<<"Afficher les objets"<<endl;
    guiTree.dumpObjectTree();

    cout<<"Fin de la fonction"<<endl;
}
int main(int argc, char *argv[]) {
    createGuiTree();
    return 0;
}
```

Sortie attendue lors de l'exécution du programme

```
Debut de la fonction
-- Creer l'element Interface
-- Creer l'element MainWidget
-- Creer l'element MenuBar
-- Creer l'element File
-- Creer l'element QuitItem
```

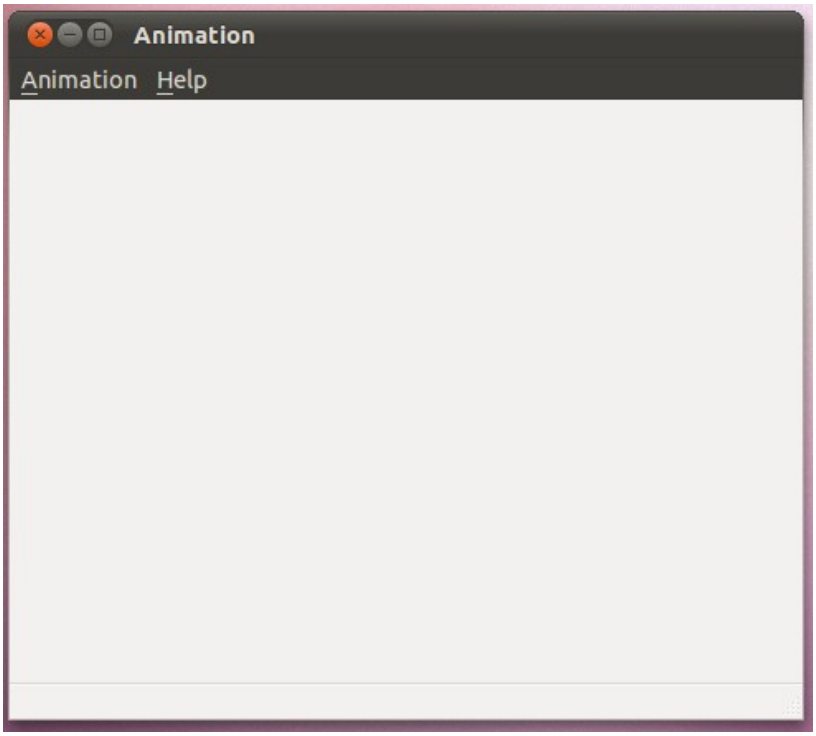
```
-- Creer l'element Edit
-- Creer l'element Help
-- Creer l'element StatusBar
Afficher les objets
Element::Interface
   Element::MainWidget
      Element::MenuBar
         Element::File
            Element::QuitItem
         Element::Edit
         Element::Help
      Element::StatusBar
Fin de la fonction
-- Liberer l'element Interface
-- Liberer l'element MainWidge
-- Liberer l'element MenuBar
-- Liberer l'element File
-- Liberer l'element QuitItem
-- Liberer l'element Edit
-- Liberer l'element Help
-- Liberer l'element StatusBar
```

*La plupart des classes (mais pas toutes) de Qt sont dérivées de **QObject**, dont le constructeur par défaut prend un pointeur en argument pour transmettre l'adresse de l'objet parent (s'il existe) de l'objet construit. Avec cette construction, les objets dérivés de **QObject** peuvent être reliés sous forme d'un arbre. Un des avantage de cette relation entre objets parents et enfants est une gestion facilitée de la mémoire allouée dynamiquement : lors de la destruction d'un objet, la mémoire allouée dynamiquement pour ses enfants est aussi libérée.*

Partie III – Fenêtre principale

Vous pouvez consulter l'exemple de la documentation de Qt pour cet exercice.
Site : http://doc.qt.nokia.com/latest/*, Examples > Main Windows Examples > Menus*

1 – Déclarez et implémentez la classe **MyMainWindow** dérivée de QMainWindow avec les attributs et les méthodes nécessaires pour pouvoir afficher la fenêtre suivante.
Le menu Animation contient les items : **Play**, **Pause**, **Reset** et **Quit**. Le menu **Help** contient l'item : **About**. Le widget central est un QWidget. Quand les élements de **MyMainWindow** sont affichés, il faut mettre un message « Done » dans la barre de statut.
2 – Reliez l'action associée à l'item **Quit** avec la fermeture de la fenêtre principale.
3 – Reliez l'action associée à l'item **About** au slot **about()** de MyMainWindow(). Le slot **about()** affichera un **QMessageBox** expliquant l'utilité de cet exercice. Il s'agit à cette étape de mettre un message « **About** » dans la barre de statut.



4 – (Si vous avez le temps) Ajoutez un toolbar pour les actions **Play**/**Pause**/**Reset**.

Ficher mymainwindow.h

```
#ifndef MYMAINWINDOW_H
#define MYMAINWINDOW_H

#include <QtGui/QMainWindow>

class MyMainWindow : public QMainWindow {
    Q_OBJECT

private:
    QMenu * animationMenu;
    QAction * playAct;
    QAction * pauseAct;
    QAction * resetAct;
    QAction * quitAct;

    QMenu * helpMenu;
    QAction * aboutAct;

    QWidget * centralWidget;

    void createActions();
    void createMenus();

private slots:
```

```
    void about();

public:
    MyMainWindow(QWidget * parent = 0);
};

#endif // MYMAINWINDOW_H
```

Partie IV – Graphique et animation

Vous pouvez consulter l'exemple de la documentation de Qt pour cet exercice.
Site : http://doc.qt.nokia.com/latest/, *Examples > Widget Examples > Analog Clock*

Reprenez le programme de la Partie III et faites les modifications suivantes :

1 – Créez la classe **MyWidget** dérivée de **QWidget** comme suit :

```
#ifndef MYWIDGET_H
#define MYWIDGET_H

#include <QWidget>
#include <QTimer>

class MyWidget : public QWidget {
    Q_OBJECT

private:
    int r; // taille du disque
    int currX; // position du disque
    int currY;
    int speedX; // vitesse du disque
    int speedY;
    bool playing; // état de l'animation (jouer ou pause)

    QTimer * timer;

private slots:
    void newposition();

public slots:
    void play();
    void pause();
    void reset();

protected:
    void paintEvent(QPaintEvent * event);

public:
    MyWidget(int size = 5,
             int limX = 460, int limY = 380,
             int sX = 5, int sY = 5, QWidget * parent = 0);
```

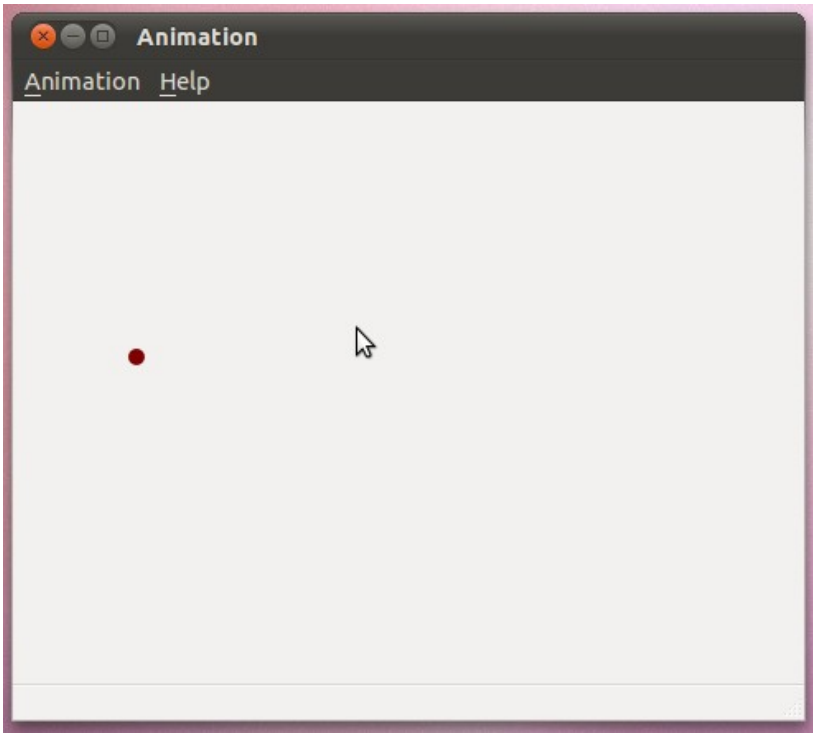
```
};

#endif // MYWIDGET_H
```

La classe est utilisée pour dessiner un disque rouge de taille **r** à la position indiqué par **currX** et **currY**. Cette position est régulièrement mise à jour pour animer le mouvement du disque :

- Le constructeur initialise les attributs du widget et en particulier crée l'objet **timer**. On associe le signal **timeout()** du **timer** au slot **newposition()** de **MyWidget**. Par défaut, le disque est mis à une position aléatoire et visible du widget.
- La fonction **newposition()** calcule la prochaine position du disque et lance la méthode **update()** (de **QWidget**). Le disque rebondit à chaque fois qu'il touche un bord de la fenêtre.
- La fonction **paintEvent()** dessine un disque à la position **currX** et **currY** du widget.
- Le slot **play()** démarre le **timer** avec la méthode **start()**.
- Le slot **pause()** arrête le **timer** avec la méthode **stop()**.
- Le slot **reset()** remet le widget à l'état initial (il régénère une nouvelle position de début pour le disque).

2 – Modifier le type de l'attribut **centralWidget** de **MyMainWindow** avec le type **MyWidget** et faites les modifications nécessaires dans le constructeur de **MyMainWindow**.
3 – Reliez les signaux et les slots entre **MyMainWindow** et **MyWidget** pour pouvoir lancer le programme.



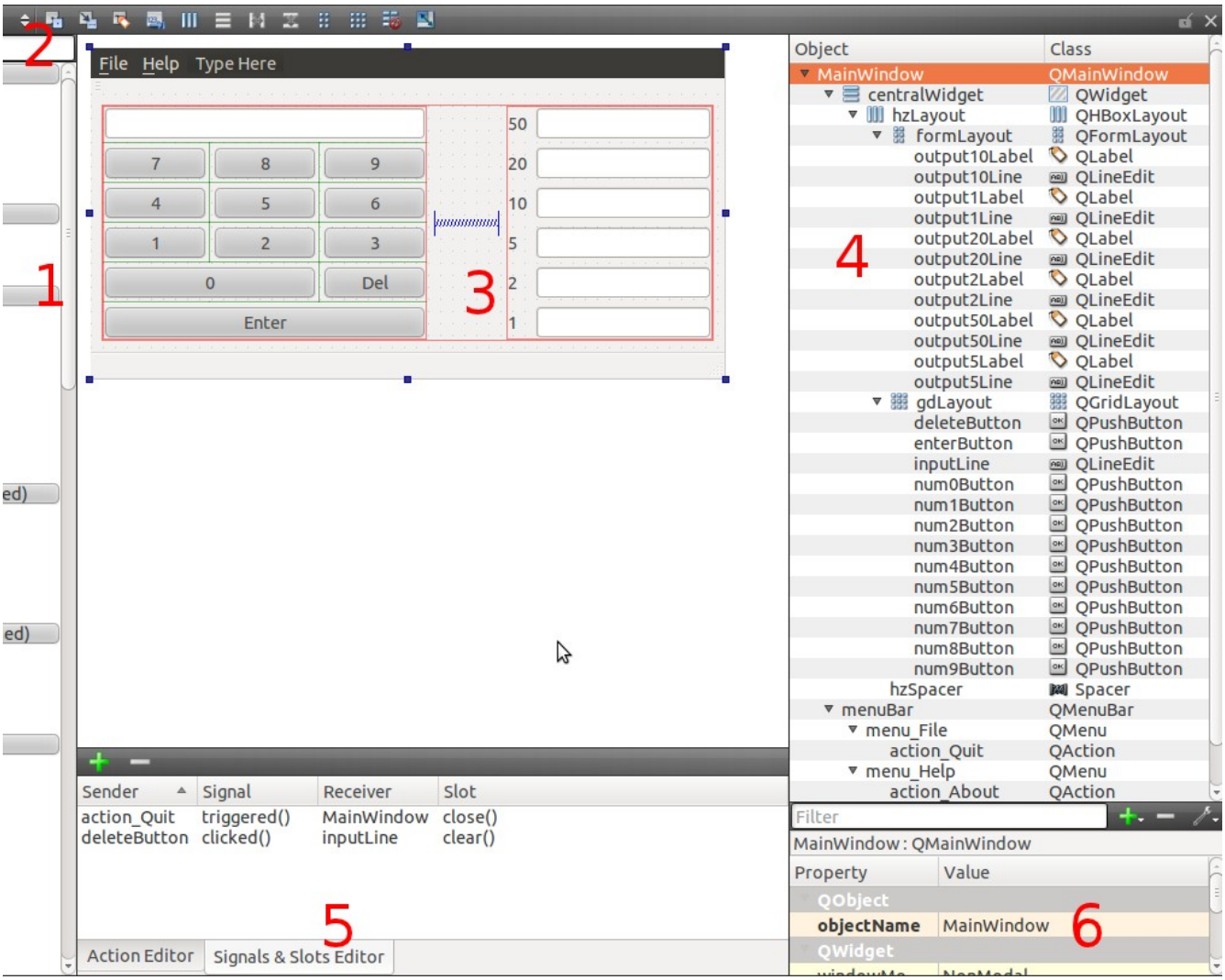
4 – (Si vous avez le temps) Modifiez le code pour jouer un son à chaque fois que le disque touche une borne de la fenêtre.
5 – (Si vous avez le temps) Modifiez le code pour animer plusieurs disques en même temps. Les disques peuvent avoir de différents caractéristiques (couleur, taille, vitesse, etc).

Partie V – Interface avec Qt Designer

Dans cette exercice, on créera un projet de type Qt Gui et utilisera les classes par défaut.

1 – Créez un projet de type Qt Gui et double-cliquez sur le fichier **mainwindow.ui** pour entrer dans le mode **Design** (**Qt Designer** plugin).

2 – Construisez l'interface graphique suivante :



Les zones numérotées sont pour :
[1] Les éléments que l'on peut ajouter dans l'interface (avec *Drag & Drop*).
[2] La sélection du mode d'édition.
[3] L'interface à l'état actuel.
[4] La structure arborescente de l'interface à l'état actuel.
[5] L'édition des signals et slots.
[6] La liste des propriétés de l'élément sélectionné dans [4].

Quelques propriétés demandées :

- Les **QLineEdit** doivent être non-modifiable (cochez la case « Read Only »).
- Les **QPushBouton** sont associés aux boutons de clavier (utilisez la propriété « Shortcut »).
- Le menu **File** contient un item **Quit** et le menu **Help** contient un item **About**.
- Le signal **triggered()** de **action_Quit** doit être relié au slot **close()** de **MainWindow**.
- Le signal **clicked()** de **deleteButton** doit être relié au slot **clear()** de **inputLine**.

3 – Repassez dans le mode **Edit** de code source et modifiez la classe **MainWindow** :

*On observe que la classe possède un pointeur privée **ui**. Ce pointeur contient l'adresse de l'objet **QMainWindow** que vous avez créé avec le Qt Designer. Vous pouvez accéder aux éléments créés par Qt Designer via ce pointeur. Les connexions de signals/slots sont maintenant à ajouter dans le constructeur de la classe **MainWindow**.*

- Implémentez les slots **numxPressed()** (x=0..9) et les reliez aux signaux **click()** des boutons de **ui**. Ces slots ajoutent le caractère correspondant à la fin de **ui->inputLine**. On utilisera les fonctions **text()** et **setText()** de ce **QLineEdit** objet.
- Implémentez le slot **enterPressed()** qui reprend le nombre actuellement présenté dans **inputLine** et essaie de le décompose en plus petits nombres 50, 20, 10, 5, 2, 1. Ces petits nombres seront affichés dans les **outputyLine (**y=50..1) associés de **ui**. Enfait, l'application permet de calculer le rendu de monnaie pour une somme d'argent indiqué par **inputLine**. Reliez le slot avec le signal **click()** du button **enterButton**.
- Implémentez le slot **about()** et le reliez à l'action **action_About** de **ui**.

4 – Exécutez et vérifiez les calculs.