Podstawy Informatyki

Wstęp do programowania w Qt

13.12.2019

dr inż. Michał Rad

Plan wykładu:

1. Co to jest Qt?
2. Pierwsza aplikacja
3. Pliki projektu
4. Sygnały i sloty
5. Qt designer

6. *Wykład oparty o:*
https://wiki.qt.io/Qt_for_Beginners

Co to jest Qt?

- Qt to *framework* używany zwykle jako zestaw narzędzi służący do tworzenia aplikacji graficznych *(GUI applications)*
- *Framework – (za Wikipedią) szkielet do budowy aplikacji. Definiuje on strukturę aplikacji oraz ogólny mechanizm jej działania, a także dostarcza zestaw komponentów i bibliotek ogólnego przeznaczenia do wykonywania określonych zadań.*

Od czego zacząć?

- Instalacja – najłatwiej zainstalować cały pakiet (biblioteki oraz środowisko zintegrowane – Qt Creator)
- Proces instalacji nie jest trudny – po zainstalowaniu mamy do dyspozycji wiele gotowych przykładów które można skompilować testować i zmieniać
- Można zacząć od uruchomienia jednego z nich

Pierwsza prosta aplikacja

- My zaczniemy od „pustego” projektu:
  File > New file or project > Other Projects > Empty Qt Project
- *Plik projektu (extension.pro). Qt używa narzędzia wiersza poleceń, które analizuje te pliki projektu w celu wygenerowania "plików Makefile", plików używanych przez kompilatory do budowania aplikacji. To narzędzie nazywa się qmake. Ale nie powinniśmy martwić się zbytnio o qmake, ponieważ Qt Creator wykona dla nas to zadanie.*

- Co ma być w pliku *.pro:

```
TEMPLATE = app
TARGET = name_of_the_app
QT = core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

- Następnie dodajemy główny plik programu:
- File > New file or project > C++ > C++ Source file

- W pliku *.pro pojawi się odpowiedni zapis

- W głównym pliku powinien znajdować się co najmniej kod:

```
#include <QApplication>


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

 return app.exec();
}
```

AGH

- QApplication to bardzo ważna klasa. Dba o obsługę argumentów wejściowych, ale także o wiele innych rzeczy, a przede wszystkim o tzw. *event loop*. Pętla zdarzeń to pętla czekająca na działanie użytkownika w aplikacjach GUI.
- Po zapisaniu tego kodu program powinien się skompilować, ale po uruchomieniu nic jeszcze się nie będzie działo

AGH

- Spróbujmy dodać jakiś obiekt, np. przycisk:

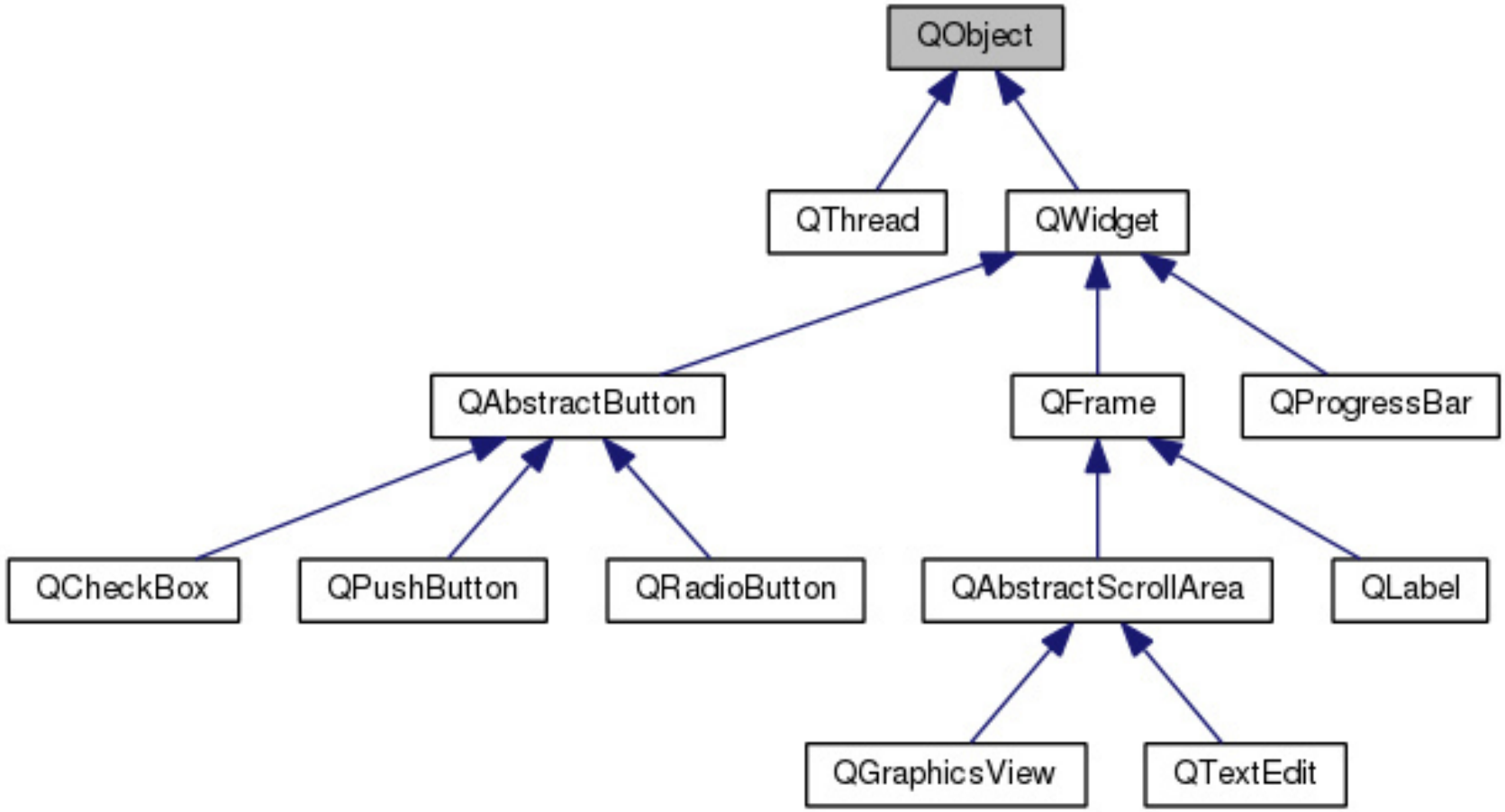
```
#include <QApplication>
#include <QpushButton>

int main(int argc, char **argv)
{
 QApplication app (argc, argv);
 QPushButton button ("Hello world !");
 button.show();
  return app.exec();
}
```

- Możemy zmienić własności przycisku:

- `button.setText("To mój guzik");`
- `button.setToolTip("Podpowiedź");`

- Oczywiście można zmieniać jeszcze inne własności

Hierarchia klas w Qt



Parenting system

- System rodzicielski jest wygodnym sposobem radzenia sobie z obiektami w Qt, zwłaszcza widżetami. Każdy obiekt dziedziczący po obiekcie QObject może mieć element nadrzędny i element potomny. To drzewo hierarchii ułatwia wiele rzeczy:
  - Kiedy obiekt zostanie zniszczony, wszystkie jego dzieci również zostaną zniszczone. Tak więc wywołanie delete staje się opcjonalne w niektórych przypadkach.
  - Wszystkie obiekty QObject mają metody findChild i findChildren, które można wykorzystać do wyszukiwania dzieci danego obiektu.
  - Widgety podrzędne w QWidget automatycznie pojawiają się w widgecie nadrzędnym.

- Nie należy tego systemu mylić z dziedziczeniem w znaczeniu języka C++

- Poniższy fragment, który tworzy QPushButton wewnątrz QpushButton:

- `QPushButton button1 ("test");`
- `QPushButton button2 ("other", &button1);`

- Nie ma żadnej korzyści z umieszczenia przycisku wewnątrz przycisku, ale w oparciu o ten pomysł możemy chcieć umieścić przyciski wewnątrz kontenera, który na razie niczego innego nie wyświetla. Ten kontener to po prostu QWidget.

AGH

```
#include <QApplication>
#include "window.h"

int main(int argc, char **argv)
{
 QApplication app (argc, argv);
 QWidget window;
  window.setFixedSize(120, 50);
  QPushButton *button = new QPushButton("Hello World", &window);
  button->setGeometry(10, 10, 80, 30);
 window.show();
 return app.exec();
}
```

Subclassing QWidget

- Do tej pory umieściliśmy cały nasz kod w głównej funkcji. Nie stanowiło to problemu dla naszych prostych przykładów, ale w przypadku coraz bardziej złożonych aplikacji możemy chcieć podzielić nasz kod na różne klasy. Często się zdarza, że tworzy się klasę, która służy do wyświetlania okna i implementuje wszystkie widżety zawarte w tym oknie jako atrybuty tej klasy.
- Wewnątrz Qt Creator możesz automatycznie utworzyć nową klasę za pomocą polecenia Plik> Nowy plik lub projekt> C + +> C ++ Class

- Można zobaczyć, że Qt Creator automatycznie generuje szablon klasy. W nagłówku pojawiają się pewne nowe elementy:
- Makro Q_OBJECT.
- Nowa kategoria metod: signals
- Nowa kategoria metod: public slots
- Elementy te zostaną wyjaśnione nieco później. Implementacja okna odbywa się w konstruktorze. Możemy zadeklarować rozmiar okna, a także widżety, które to okno zawiera i ich pozycje. Na przykład implementację poprzedniego okna zawierającego przycisk można wykonać w następujący sposób:

AGH

- main.cpp:

```
#include <QApplication>
#include "window.h"

int main(int argc, char **argv)
{
 QApplication app (argc, argv);
 Window window;
 window.show();
 return app.exec();
}
```

- window.h

```
#ifndef WINDOW_H
#define WINDOW_H

#include <QWidget>

class QPushButton;
class Window : public QWidget
{
 public:
  explicit Window(QWidget *parent = 0);
 private:
 QPushButton *m_button;
};
#endif // WINDOW_H
```

- window.cpp

```
#include "window.h"
#include <QPushButton>

Window::Window(QWidget *parent) :
 QWidget(parent)
 {
 // Set size of the window
 setFixedSize(100, 50);

 // Create and position the button
 m_button = new QPushButton("Hello World", this);
 m_button->setGeometry(10, 10, 80, 30);
}
```

Sygnały i sloty

- Aby nasze obiekty reagowały na działanie uzytkownika, Qt zapewnia dwie koncepcje wysokiego poziomu: sygnały i sloty (gniazda).
- **Sygnał** jest komunikatem, który obiekt może wysłać, w większości przypadków informując o zmianie statusu.
- **Slot** to funkcja używana do akceptowania i reagowania na sygnał.
- Oto kilka przykładów sygnałów i slotów z naszej dobrze znanej klasy QPushButton.
- clicked
- pressed
- released
- Jak widać, ich nazwy są dość oczywiste. Sygnały te są wysyłane po kliknięciu (naciśnięciu i zwolnieniu), naciśnięciu lub zwolnieniu przycisku.

Sygnały i sloty

- Oto kilka przykładów slotów z różnych klas
- QApplication :: quit()
- QWidget :: setEnabled()
- QPushButton :: setText()
- Aby zareagować na sygnał, slot musi być podłączony do sygnału. Qt zapewnia metodę *QObject :: connect(...).*
- Jest ona używana w następujący sposób, z dwoma makrami SIGNAL i SLOT:

- Przykład połączenia sygnału ze slotem:

```
 connect(m_button, SIGNAL (clicked()), this, SLOT
(wypisz()));
```

- Wykorzystując dość dużą kolekcję różnych obiektów oraz posługując się systemem łączenia sygnałów ze slotami można tworzyć dość zaawansowane, a w każdym razie użyteczne aplikacje