Fundusze Europejskie
Wiedza Edukacja Rozwój

Unia Europejska
Europejski Fundusz Społeczny

Programowanie zespołowe 2.0

Wprowadzenie do biblioteki Qt i środowiska *QtCreator*

Scenariusz zajęć

Czas realizacji: 180 minut
Cele ogólne: Omówienie podstaw tworzenia aplikacji wykorzystujących bibliotekę Qt.
Cel szczegółowy: Przygotowanie kilku prostych aplikacji z graficznym interfejsem użytkownika za pomocą narzędzia *QtCreator*
Konieczne wiadomości wstępne: Umiejętność programowania w języku C++.
Metoda prowadzenia zajęć: Dyskusja z elementami wykładu.

Wprowadzenie

Qt to zestaw bibliotek i narzędzi programistycznych umożliwiający wygodne tworzenie aplikacji z graficznym interfejsem użytkownika (GUI). Dedykowany głównie dla języka C++, ale istnieją też wersje dla innych języków (np. C# i Pythona). Jest to biblioteka wieloplatformowa – w większości przypadków ten sam kod źródłowy można skompilować pod systemami Windows, Linux i iOS. Istnieje wersja darmowa (na licencjach GPL 2.0, GPL 3.0 i LGPL 3.0) oraz komercyjna. Aktualna (czerwiec 2019 r.) wersja biblioteki to 5.13.

Instalacja

Program możemy pobrać ze strony `https://www.qt.io/download`, na której wybieramy wersję *Open Source*. Strona zwykle wykryje nasz system operacyjny i zaproponuje odpowiedni instalator, np. *Qt Online Installer for Windows*. Po kliknięciu przycisku *Download* pobrany zostanie niewielki plik instalatora. Po jego uruchomieniu rozpocznie się właściwy proces instalacji. Propozycję utworzenia konta Qt możemy zignorować. Kiedy pojawi się prośba o wybór elementów instalacyjnych najlepiej rozwinąć gałąź z najnowszą dostępną wersją Qt, a na liście która się pojawi zaznaczyć opcję *MinGw 64-bit* (lub ewentualnie wersję 32-bit). Można oczywiście wybrać inną wersję np. taką przeznaczoną do współpracy z Microsoft Visual Studio. Należy pamiętać, że tylko ta jedna zaznaczona wersja zajmie na dysku około 5–6GB. Zaznaczenie wszystkich dostępnych opcji (całej gałęzi) spowoduje zajęcie ponad 40GB. Pozostałe kroki instalacyjne są dosyć standardowe nie wymagają dodatkowego komentarza.

Pierwszy projekt

Po zakończeniu procesu instalacji, oprócz biblioteki Qt i kompilatora dostajemy zintegrowane środowisko programistyczne *QtCreator*. Skorzystamy z niego aby utworzyć naszą pierwszą aplikację. Aplikacje Qt możemy oczywiście tworzyć w dowolnym edytorze i kompilować z linii komend, ale *QtCreator* znacznie ułatwia ten proces.

Po uruchomieniu programu, mamy możliwość utworzenia nowego projektu, otwarcia jednego z ostatnio używanych, a także skorzystania z przykładowych projektów i tutoriali. Wybierzmy opcję *New Project*. W oknie, które się pojawi musimy wybrać rodzaj tworzonej aplikacji – w naszym przypadku będzie to *Qt Widgets Application* czyli typowa aplikacja „okienkowa". Możemy tez tworzyć aplikacje konsolowe, a także zwykłe aplikacji języka C++ niewykorzystujące biblioteki Qt.

Następnie określamy nazwę projektu i położenie jego plików źródłowych. W wybranym miejscu zostanie utworzony nowy katalog o nazwie takiej samej jak nazwa projektu. Dobrym zwyczajem jest unikanie spacji i polskich znaków diakrytycznych w nazwie projektu i ścieżce jego katalogu.

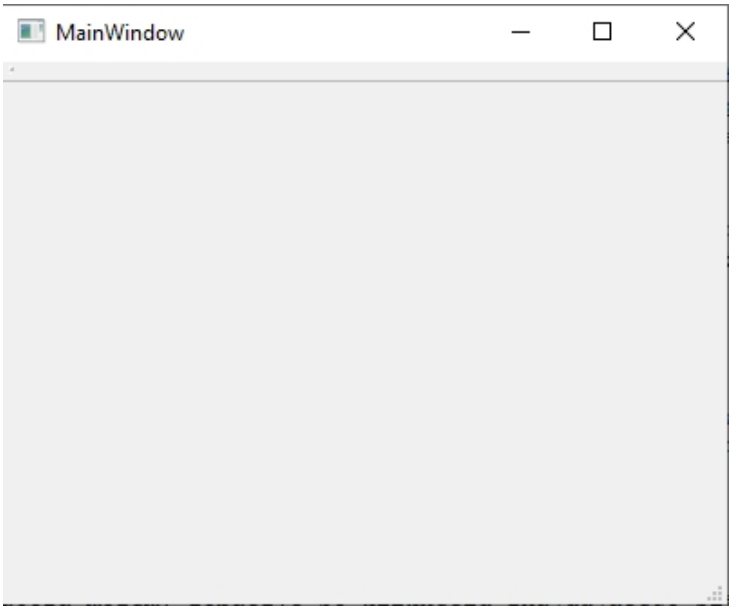
Kolejnym krokiem jest tzw. *Kit Selection* czyli wybór wersji kompilatora, której zamierzamy używać. Jeżeli przy instalacji zaznaczyliśmy tylko jedną wersję zostanie ona domyślnie zaznaczona.

Następnie wybieramy nazwę głównej klasy naszego projektu. Domyślnie proponowana jest nazwa *MainWindow*. Możemy też określić jej klasę bazową, domyślnie jest to *QMainWindow* i w większości przypadków jest to dobry wybór. W ostatnim oknie możemy połączyć nasz projekt z używanym przez nas systemem kontroli wersji.

Po naciśnięciu przycisku *Finish* zostaną wygenerowane pliki źródłowe naszej aplikacji. Zanim się im przyjrzymy spróbujemy skompilować i uruchomić naszą aplikację. W tym celu wciskamy znajdujący się w lewym dolnym narożniku programu przycisk *Run* (symbolizowany przez ikonę z zielonym trójkątem). Możemy też użyć skrótu *Ctrl+R*. Komunikaty kompilatora możemy zobaczyć po kliknięciu znajdującego się na dolnej belce przycisku *Compile Output*. Po krótkiej chwili kompilacja zakończy się, a na ekranie pojawi się okno naszej aplikacji (takie jak na rysunku 1). Okno jest puste, ale zawiera standardowe elementy okna systemowego.

Pliki źródłowe

Zanim zaczniemy rozbudowywać naszą aplikację, przyjrzyjmy się wygenerowanym plikom źródłowym. Po lewej stronie *QtCreatora* widzimy tzw. drzewo projektu. Po jego rozwinięciu na samej górze powinniśmy ujrzeć plik *nazwaprojektu.pro*. Jest to tzw. plik projektu, zawiera on pewne ogólne opcje naszego projektu. W szczególności powinien zawierać następujące wiersze



Grafika 1: Główne okno naszego pierwszego projektu

```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = HelloWorld
TEMPLATE = app
CONFIG += c++11
SOURCES += \
        main.cpp \
        mainwindow.cpp
HEADERS += \
        mainwindow.h
FORMS += \
        mainwindow.ui
```

Wiersze zaczynające się od słowa `QT` zawierają listę modułów, które dołączamy do aplikacji. W przypadku aplikacji okienkowych sa to zwykle: `core`, `gui` i `widgets`. Linia `TARGET` określa nazwę pliku wynikowego, a w linii `CONFIG` znajduje się opcja pozwalająca korzystać ze standardu C++11. W kolejnych wierszach znajduje się lista plików wchodzących w skład projektu. Zwykle nie musimy ręcznie modyfikować pliku projektu, czasem jednak jest to konieczne. Na przykład jeżeli tworzymy aplikacje bazodanową, to w linii `QT` musimy dodać moduł `sql`.

Kolejnym plikiem wchodzącym w skład projektu jest plik nagłówkowy – *mainwindow.h* (jeżeli tworząc projet nie zmieniliśmy domyślnej nazwy klasy). Najważniejszą częścią pliku jest deklaracja naszej głównej klasy

```
#include <QMainWindow>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
};
```

Ponieważ nasza klasa dziedziczy po klasie `QMainWindow` na początku dołączamy plik nagłówkowy tej klasy. `Q_OBJECT` to specjalne makro umożliwiające tworzenie własnych sygnałów i slotów (szczegóły później). Sekcja publiczna zawiera deklaracje konstruktora i destruktora. Parametrem konstruktora jest wskaźnik do rodzica danego obiektu. Ponieważ, będzie to główne okno aplikacji, domyślnie wskaźnik ten ustawiony jest na `null`. W sekcji prywatnej znajduje się tylko wskaźnik `ui`. Umożliwi on nam dostęp do elementów interfejsu graficznego aplikacji. W miarę rozbudowy naszej klasy do pliku tego będziemy dodawać deklaracje kolejnych pól i metod.

Definicje funkcji składowych klasy zawiera plik *mainwindow.cpp*. Plik ten będzie głównym miejscem do którego będziemy wstawiać nasz kod. Jego początkowa zawartość jest dość uboga:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

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

W konstruktorze wywołujemy konstruktor klasy bazowej oraz tworzymy wskaźnik `ui` do interfejsu graficznego. W destruktorze niszczymy ten wskaźnik.

Ostatnim plikiem źródłowym jest *main.cpp* o następującej zawartości.

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Widzimy, ze głównym elementem pliku jest funkcja `main`, czyli punkt wejścia do aplikacji. Tworzymy najpierw obiekt klasy `QApplication`, który będzie reprezentacją naszej aplikacji. Następnie tworzymy obiekt naszej głównej klasy (czyli główne okno aplikacji) i za pomocą metody `show()` pokazujemy je na ekranie. Na koniec wywołujemy metodę `exec` na obiekcie aplikacji co powoduje rozpoczęcie nieskończonej pętli, w której aplikacja będzie oczekiwać i reagować na działania użytkownika (np. naciśnięcia przycisków, wybór opcji z menu itp.). Plik ten ma taką samą zawartość dla wszystkich aplikacji okienkowych. Prawie nigdy nie zachodzi konieczność jego edycji.

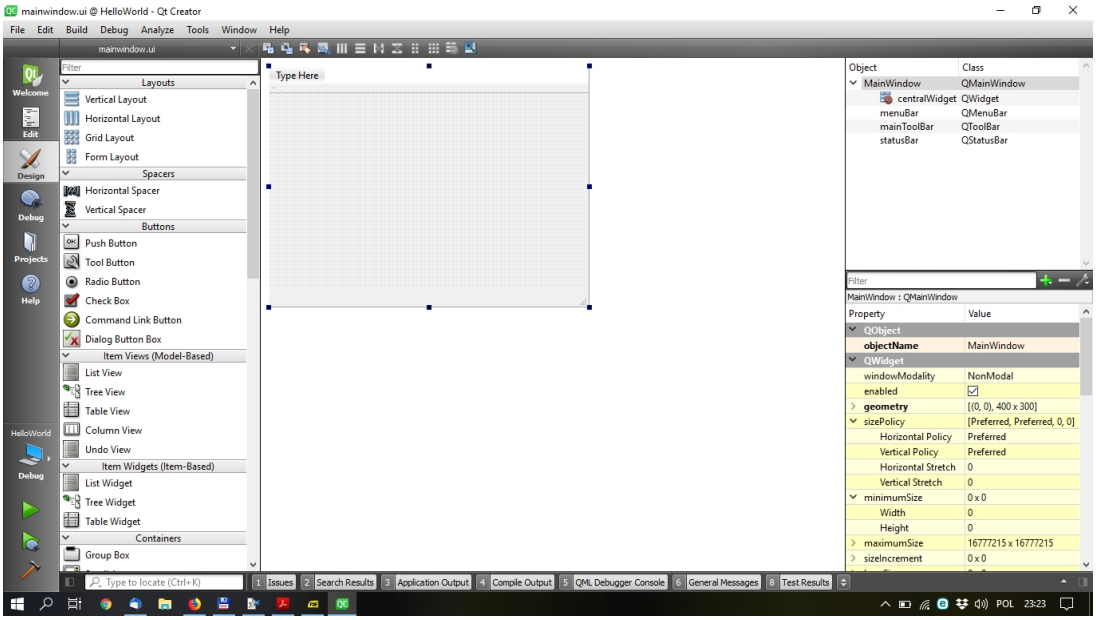
Na samym dole drzewa projektu znajduje się jeszcze jeden plik *mainwindow.ui*. Jest to plik XML zawierający opis interfejsu graficznego naszej aplikacji. Po dwukrotnym kliknięciu na ten plik nie zobaczymy jego zawartości. Zamiast tego otworzy się tzw. *QtDesigner* czyli wizualny edytor GUI.

Edytor *QtDesigner*

Centralną część okna edytora zajmuje widok okna naszej aplikacji. Po lewej stronie widzimy *przybornik* zawierający elementy GUI, które możemy dodać do projektu. Po prawej stronie znajduje się *drzewo interfejsu graficznego*, które pokazuje wzajemne zależności pomiędzy elementami GUI oraz *edytor właściwości* pozwalający zmieniać właściwości aktualnie zaznaczonego elementu GUI.

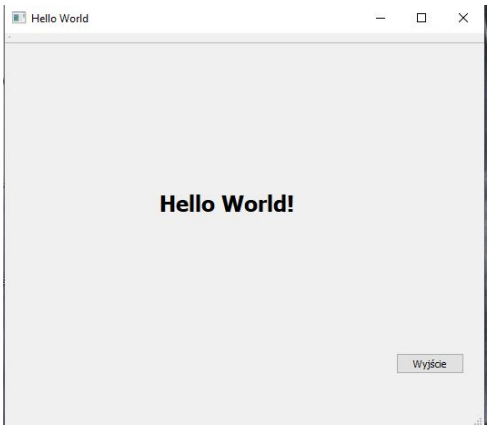
Korzystanie z *QtDesignera* jest bardzo intuicyjne. Na początku spróbujmy zmienić rozmiar okna, w tym celu wystarczy przeciągnąć myszką narożnik okna. Jeżeli zależy nam na ustawieniu dokładnych wymiarów korzystamy z edytora właściwości. Znajdujemy w nim właściwość `geometry` rozwijamy ją i wpisujemy odpowiednie wartości w pola `Width` i `Height`. Możemy też przy okazji zmienić nazwę znajdującą się belce tytułowej okna aplikacji (właściwość `windowTitle`. Dokonane zmiany będą widoczne po ponownym skompilowaniu projektu. Zauważmy, że w plikach źródłowych nie zaszły żadne zmiany. Większość operacji w *QtDesignerze* zmienia tylko plik XML *mainwindow.ui*, który dopiero podczas kompilacji przetwarzany jest na ciąg odpowiednich poleceń języka C++.

Spróbujmy teraz dodać do okna aplikacji nowy element GUI. Rozpocznijmy od prostej etykiety (napisu). Formalnie będzie to obiekt klasy `QLabel`. Znajdujemy odpowiedni element w przyborniku (sekcja *Display Widgets* i przeciągamy go do okna aplikacji. Po dwukrotnym kliknięciu na nowy element możemy zmienić jego zawartość tekstową. Inne opcje np. wielkość i krój fontu możemy zmienić w edytorze właściwości.



Grafika 2: QtDesigner

Teraz dodamy przycisk (obiekt klasy `QPushButton`). Podobnie jak w przypadku etykiety wystarczy przeciągnąć go do okna. Następnie zmieniamy tekst na przycisku. Możemy to uczynić po dwukrotnym kliknięciu bądź ustawiając odpowiednią wartość właściwości `text`. Skompilujmy aplikację, aby sprawdzić czy nowe elementy są widoczne.



Grafika 3: Aplikacja z nowymi elementami

Sygnały i sloty

Jeżeli wszystko przebiegło pomyślnie w oknie aplikacji ujrzymy napis i przycisk. Przycisk można nacisnąć, ale nie spowoduje to żadnego efektu. Musimy utworzyć odpowiednią funkcję i „połączyć" ja z przyciskiem. W bibliotece Qt służą do tego tzw. *sygnały* i *sloty*.

Załóżmy, że użytkownik nacisnął przycisk znajdujący się w jednym z okien programu.

Wówczas przycisk ten (czyli obiekt klasy `QPushButton`) emituje sygnał o nazwie `clicked`. Sygnał ten może zostać odebrany przez inny obiekt, który następnie zareaguje wywołując odpowiednią funkcją, którą nazywać będziemy slotem. Każdy obiekt w bibliotece Qt posiada domyślną liczbę sygnałów, które może wyemitować oraz slotów, które może wywołać. Na przykład przyciski mogą emitować sygnały: `clicked`, `pressed`, `released`. Programiści maja możliwość tworzenia własnych sygnałów i slotów.

Dodamy teraz do naszej aplikacji slot, który zakończy jej działanie. Najprościej zrobić to za pomocą *QtDesignera*. Klikamy prawym przyciskiem myszy na przycisku i wybieramy z menu kontekstowego polecenie *Go to slot...*. Pojawi się lista dostępnych sygnałów, z której wybieramy `clicked()`. Po zatwierdzeniu wyboru zostaniemy przeniesieni do edycji pliku *mainwindow.cpp*, w którym pojawi się szkielet funkcji o nazwie `on_pushButton_clicked()`. Ciało tej funkcji jest puste, wypełnienie jej treścią należy do nas. Możemy również sprawdzić, że w pliku nagłówkowym pojawiła się nowa sekcja `private slots`, a w niej nagłówek naszej nowej funkcji. Funkcja ta będzie wywoływana automatycznie, po naciśnięciu przycisku.

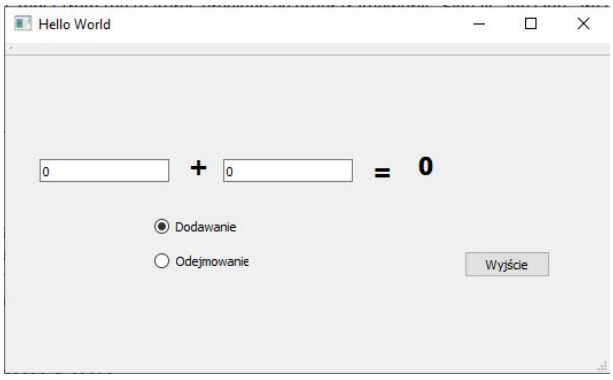
Ponieważ przycisk ma zakończyć działanie aplikacji, skorzystamy z odpowiedniej systemowej funkcji Qt:

```
void MainWindow::on_pushButton_clicked()
{
    qApp->quit();
}
```

`qApp` to wskaźnik do obiektu reprezentującego naszą aplikację. Po kompilacji i uruchomieniu programu możemy sprawdzić działanie przycisku.

Prosty kalkulator

Aby lepiej zrozumieć działanie opisanego mechanizmu, spróbujmy zamienić naszą aplikację w prosty kalkulator. Dodajmy do głównego okna aplikacji dwa pola edycyjne (*LineEdit*, dwa przyciski radiowe i dwie etykiety. Ułóżmy je tak jak na rysunku 4.



Grafika 4: Kalkulator

Musimy „oprogramować" następujące zdarzenia: zmianę wartości w obydwu polach tekstowych i zmianę działania. Niezależnie od tego, które z tych zdarzeń wystąpi rezultatem powinno być ponowne wykonanie obliczeń i wyświetlenie ich wyniku. Napiszemy

więc jedną funkcje `oblicz()` i będziemy ją wywoływać po wszystkich zdarzeniach tego typu.

```
void MainWindow::oblicz()
{
    double x1 = (ui->lineEdit->text()).toDouble();
    double x2 = (ui->lineEdit_2->text()).toDouble();
    double wynik;
    if(ui->radioButton->isChecked())
        wynik = x1 + x2;
    else
        wynik = x1 - x2;
    ui->label_4->setText(QString::number(wynik));
}
```

Na początku pobieramy zawartość obydwu tekstowych pól edycyjnych. Zwróćmy uwagę, że odwołujemy się do nich w formie `ui->nazwa_elementu`. Zawartość pobieramy za pomocą metody `text()` i zamieniamy na liczbę typu `double`. Następnie sprawdzamy czy zaznaczony jest pierwszy przycisk radiowy oznaczający dodawanie. Jeżeli tak, to dodajemy obie liczby, a w przeciwnym wypadku odejmujemy. Na koniec za pomocą metody `setText` zmieniamy zawartość etykiety wyświetlającej wynik. Zwróćmy uwagę na sposób konwersji z formatu liczbowego do typu `QString`.

Na koniec musimy dodać deklarację funkcji do pliku nagłówkowego. W tym celu w pliku *mainwinddow.h* w sekcji `private` dodajemy następujący wiersz:

```
void oblicz();
```

Kolejnym krokiem będzie połączenie funkcji oblicz z odpowiednimi zdarzeniami. Przechodzimy do *QtDesignera* i klikamy prawym przyciskiem myszy kolejno na obydwa pola tekstowe i obydwa przyciski radiowe. Za każdym razem z menu kontekstowego wybieramy polecenie *Go to Slot...*. Dla pól edycyjnych wybieramy sygnał `textChanged`, a dla przycisków radiowych `clicked`. W ciele każdego z utworzonych slotów wywołujemy funkcję `oblicz`.

```
void MainWindow::on_lineEdit_2_textChanged(const QString &arg1)
{
    oblicz();
}
```

Klasa `QDebug`

Podczas tworzenia aplikacji na pewno popełnimy błędy. Proces ich znajdowania i poprawiania nazywamy debuggowaniem. Opis debuggera wbudowanego w *QtCreatora* wykracza poza zakres tego scenariusza. W większości przypadków wystarczy jednak skorzystać z prostej konsoli tekstowej wyświetlającej komunikaty. Wystarczy dołączyć do naszego programu plik nagłówkowy klasy `QDebug`

```
#include<QDebug>
```

Wtedy w dowolnym miejscu programu możemy wyświetlać komunikaty za pomocą polecenia `qDebug()`. Posługujemy się nim tak samo jak poleceniem `cout` w standardowym C++.

```
qDebug()<<"Wartości zmiennych: "< <x1 << " " << x2;
```

Odnośniki i załączniki

[1] Dokumentacja biblioteki Qt: https://doc.qt.io/qt-5/classes.html

[2] Sygnały i sloty: https://doc.qt.io/qt-5/signalsandslots.html