Laboratorium 8 – wprowadzenie do Qt

Wprowadzenie

Qt jest zestawem bibliotek oraz narzędzi dla języka C++ służącym do tworzenia graficznych interfejsów użytkownika. Razem z biblioteką dystrybuowane jest zintegrowane środowisko programistyczne Qt Creator, udostępniające graficzny edytor interfejsów, działający na zasadzie „przeciągnij i upuść” oraz szereg funkcji wspomagających szybsze zarządzanie oraz pisanie kodów źródłowych aplikacji. Dla ułatwienia identyfikacji klas udostępnionych przez Qt, nazwa każdej klasy biblioteki rozpoczyna się od wielkiej litery „Q”.

Struktura projektu oraz proces kompilacji

Po utworzeniu nowego projektu widzimy następujące plik:

<NAZWA_PROJEKTU>.pro - plik projektu zawiera informacje o użytych modułach (na początku tylko core i gui oraz widgets zależnie od wersji biblioteki) oraz listę plików źródłowych.

mainwindow.h - Pliki nagłówkowy zawierający definicję klasy okna głównego. Zwykle okno główne dziedziczy po klasie `QMainWindow`. `Q_OBJECT` jest makro instrukcją która musi znaleźć się w części prywatnej każdej klasy zarządzanej przez Qt (dziedziczącej pośrednio lub bezpośrednio po `QObject`).

Warto zwrócić uwagę na klasę `MainWindow` w przestrzeni nazw `Ui`

```
namespace Ui {
class MainWindow;
}
```

Taki zapis (słowo kluczowe class oraz nazwa bez definicji) jest informacją dla kompilatora że w klasa `Ui::MainWindow` istnieje ale nie chcemy zamieszczać w tym miejscu (lub dołączać pliku nagłówkowego) jej definicji. Jest to za mało by na tej podstawie utworzyć instancję obiektu ale wystarczy żeby zadeklarować wskaźnik.

mainwindow.cpp - Implementacja metod klasy `MainWindow`. W pliku dołączony jest plik nagłówkowy `ui_mainwindow.h` zawierający definicję klasy `Ui::MainWindow`. Klasy z przestrzeni nazw `Ui` generowane są automatycznie na podstawie plików *.ui. Zawierają wskaźniki do wszystkich elementów interfejsu oraz metodę `setupUi` która odpowiada za ich tworzenie.

mainwindow.ui - Plik w formacie xml z opisem interfejsu. Na jego podstawie generowane są klasy z przestrzeni nazw `Ui`. Plik ten nie powinien być modyfikowany ręcznie. Qt Creator posiada deigner który uruchamia się automatycznie po kliknięciu na plik.

main.cpp – Zawiera funkcję main w której tworzone jest główne okno programu oraz uruchamiana jest pętla główna (`a.exec()`) w której Qt przetwarza zdarzenia.

Designer

Po kliknięciu na plik `mainwindow.ui` otwiera się narzędzie projektowanie interfejsu. Po lewej znajduje się lista dostępnych komponentów. W centrum projekt okna. Po prawej na górze hierarchia dodanych do okna komponentów (każdy komponent „należy” do jakiegoś komponentu nadrzędnego). Poniżej znajduje się lista właściwości zaznaczonego obecnie komponentu. Właściwości pogrupowane są klasami z hierarchii dziedziczenia, przez które zostały udostępnione.

Dla naszego okna głównego są to:

- `QObject`
- `QWidget`
- `QMainWindow`

Aby dodać do okna komponent wystarczy przeciągnąć go z listy w odpowiednie miejsce projektu. Warto też od razu zmienić właściwość `objectName` na coś co będzie jednoznacznie świadczyć o przeznaczeniu obiektu – jest to nazwa pod jaką komponent będzie dostępy w kodzie.

Ćwiczenie 1

Odnajdź właściwość okna odpowiedzialną za tekst na belce tytułowej i zmień go na „Kalkulator”. Dodaj do okna przycisk (Push Button) zmień jego nazwę na `calculateButton`. Odnajdź właściwość przycisku pozwalającą na ustawienie tekstu na przycisku oraz zmień ją na „Oblicz”. Uruchom program.

Sygnały i sloty

Do łączenia ze sobą zdarzeń występujących na interfejsie wykorzystywany jest mechanizm signals & slots. Wszystkie klasy, które dziedziczą po klasie `QObject` lub po jednej z jej podklas (np. `QWidget`) mogą zawierać sygnały i gniazda. Sygnały są wywoływane przez obiekty, podczas zmiany ich stanu w sposób, który może być interesujący dla innych obiektów. Wywołania następują niezależnie od tego czy jakieś obiekty je nasłuchują.

Aby zdefiniować własne sygnały/sloty należy posłużyć się następującymi słowami kluczowymi:

- `public slots:`
- `signals:`

Używa się ich wewnątrz klas taka jak modyfikatorów dostępu.

UWAGA: Słowa te nie są elementem standardu języka c++, są rozszerzeniem jakie zostało wprowadzone w bibliotece Qt. Przed kompilacją są przetwarzane przez odpowiedni preprocesor dostarczony razem z biblioteką.

Sloty są zwykłymi funkcjami i poza tym że mogą być wywoływane w odpowiedzi na sygnał mogą być wywołane też w kodzie. Możliwe jest użycie także innych modyfikatorów dostępu zamiast public. Trzeba jednak pamiętać że będą one miały zastosowanie tylko w przypadku ręcznego wywoływanie tych metod w kodzie.

Sygnały nie posiadają implementacji, są tylko deklaracją funkcji. Aby spowodować „nadanie” zadeklarowanego wcześniej sygnału należy posłużyć się słowem kluczowym `emit`.

Jeśli sygnał jest połączony do kilku gniazd, gniazda zostaną wywołane w tej samej kolejności, w jakiej zostały zdefiniowane połączenia.

Do łączenia ze sobą sygnałów i gniazd służy funkcja:

`QObject::connect(const QObject * sender, const char * signal, const QObject * receiver, const char * method, Qt::ConnectionType type = Qt::AutoConnection)`

Posiada ona następujące argumenty:

1. *sender* – obiekt źródłowych
2. *signal* – sygnał
3. *receiver* – obiekt odbierający sygnał
4. *method* – gniazdo, które zostanie wywoływane po odebraniu sygnału
5. *type* – *opcjonalny* parametr, określający sposób połączenia

Przykład użycia:

```
QLabel *label = new QLabel;
QScrollBar *scrollBar = new QScrollBar;
QObject::connect(scrollBar,SIGNAL(valueChanged(int)),label,SLOT(setNum(int)));
```

Nie zawsze trzeba ręcznie tworzyć odpowiednie sloty. Można to zrobić z poziomu designera, klikając prawym przyciskiem myszy na komponent emitujący interesujący nas sygnał następnie „Przejdź do slotu”. Zobaczymy wtedy listę dostępnych sygnałów, po wybraniu odpowiedniego ide stworzy odpowiedni slot za nas.

Konsola i qDebug()

Qt posiada kilka funkcji pozwalających na wypisanie tekstu na konsolę jedną z nich jest `qDebug()` zdefiniowana w pliku nagłówkowym `QDebug`. Zwraca ona strumień.

Przykład użycia:

```
qDebug() << "test";
```

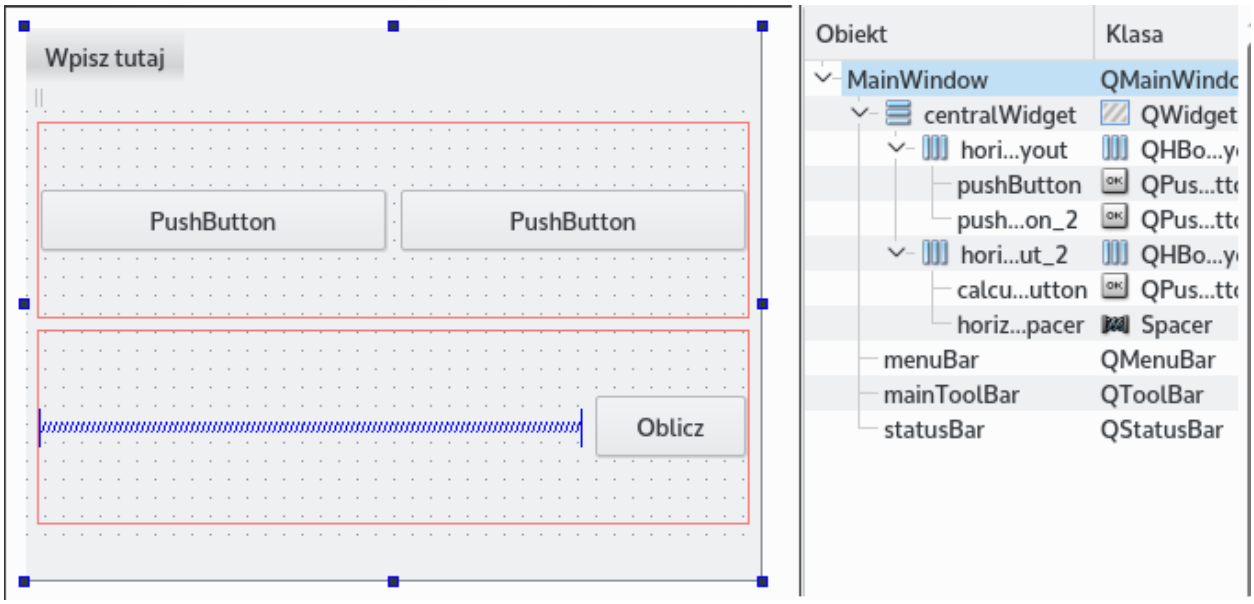
Ćwiczenie 2

Utwórz slot pozwalający na obsługę zdarzenia kliknięcia przycisku z ćwiczenia 1. Korzystając z funkcji `qDebug()` wyświetl w konsoli „Kliknięto calculateButton”. Program skompiluj i przetestuj.

Layouts & spacers

Przy skalowaniu okna elementy umieszczone bezpośrednio w oknie nie zmieniają swojego podrzenia. Aby to zmienić stosuje się Layouty. Najczęściej stosowane to Vertical Layout oraz Horizontal Layout. Layouty można w sobie zagnieżdżać. Elementy typu container posiadają „wbudowany” layout, aby z niego skorzystać wystarczy kliknąć na nim ppm i wybrać opcję Rozmieść.

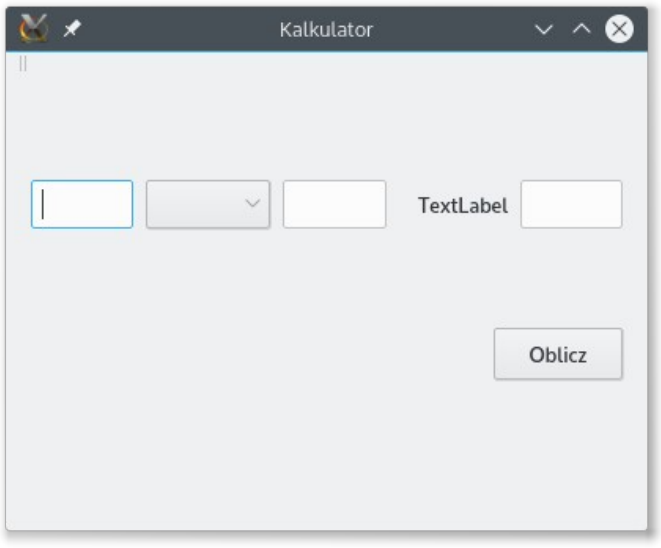
Przykład:



Na powyższym rysunku widzimy 2 przycisku wewnątrz Horizontal Layout. W Kolejnym Horizontal Layout widzimy przycisk i Horizontal Spacer. Oba layouty zostały ustawione wertykalnie poprzez „wbudowany” layout okna.

Ćwiczenie 3

Dołóż do programu z ćwiczenia 2 odpowiednie elementy aby osiągnąć poniższy efekt:



Użyj następujących kontrolek:

- Line Edit
- Combo Box
- Label

Zmień tekst „TaxtLabel” na „=”. Zmień nazwę Combo Boxa na `operation`. Zmień nazwy pól Line Edit odpowiednio na `a`, `b` i `result`. Zablokuj możliwość edycji ostatniego pola.

Combo Box

Aby dostać się z poziomu kodu do kontrolek stworzonych w designerze należy użyć obiektu klasy generowanej na podstawie pliku *.ui.

Do ustawiania wartości jakie będą dostępne w polu typu Combo Box można użyć metody:

```
void QComboBox::addItem(const QString & text, const QVariant & userData = Qvariant())
```

Pierwszy parametr jest ciągiem znaków jaki będzie widoczny w polu, a drugi jest dodatkową wartością jak może być skojarzona z dodawaną opcją (domyślnie puste);

Przykład użycia:

```
ui->operation->addItem("opcja1");
```

Aby sprawdzić która opcja jest w danej chwili zaznaczona można użyć metod `currentText()` lub `currentIndex()`.

Możliwe jest również dodanie odpowiednich opcji z poziomu designera, klikając ppm na Combo Box a następnie „Modyfikuj elementy”

Ćwiczenie 4

Dodaj do kontrolki `operation` 4 podstawowe operacje matematyczne.

Line Edit

Aby pobrać wartość z kontrolki typu line edit należy posłużyć się metodą `text()`. Która zwraca obiekt typu `QString`. Jego dużą przewagą nad standardowym typem string jest to że posada metody typu `toInt()`, `toFloat()` itd. Analogicznie do ustawienia wartości pola tekstowego służy metoda `setText()`. Przydatna będzie także metoda statyczna `Qstring::number(float)` konwertująca liczbę na ciąg znaków.

Ćwiczenie 5

Zaimplementuj wszystkie 4 podstawowe operacje matematyczne. Uruchom i przetestuj program.

Ćwiczenie 6

Zmodyfikuj program w taki sposób aby wartość była wyliczana automatycznie po zmianie w polach `a` i `b` oraz w przypadku zmiany operacji. W tym celu posłuż się sygnałem `currentIndexChanged(int index)` lub `currentIndexChanged(const QString & text)` w przypadku Combo Box oraz `textChanged(const QString & text)` w przypadku Line Edit. Można wykorzystać jeden slot przypinając go ręcznie do wielu sygnałów.