

# Programowanie w środowiskach RAD

## Qt i C++

**Roman Simiński**

roman.siminski@us.edu.pl

www.siminskionline.pl

Wprowadzenie do programowania w języku C++  
z wykorzystaniem biblioteki Qt

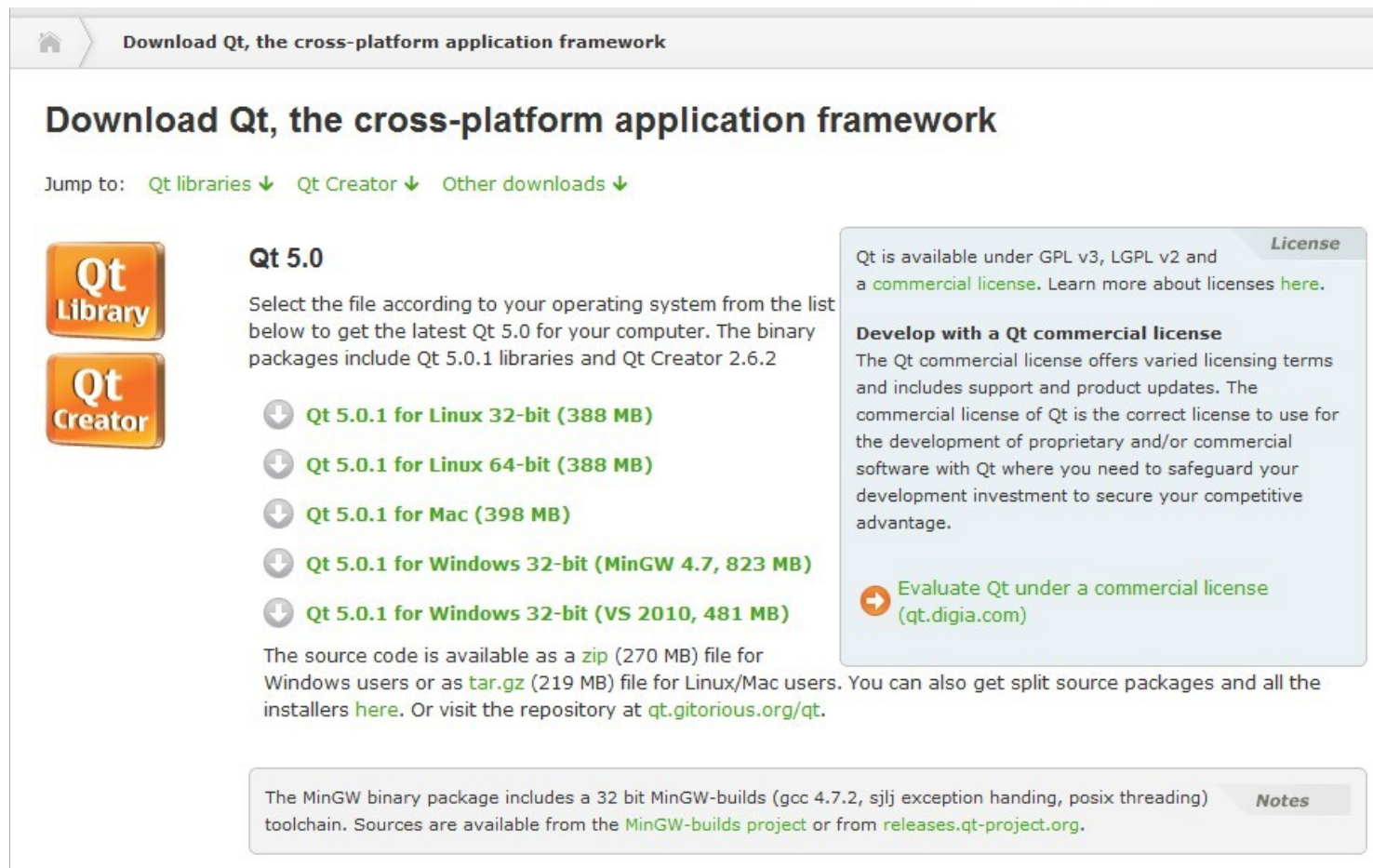
podstawy i języki  
programowania

```
for( i = 0; i != '\0'; i++)  
; class Point3D : public Point
```

*pjp* Pascal  
I := Succ(I);  
C++

Aktualnie rozwój Qt jest nadzorowany przez firmę Digia: <http://qt.digia.com/>

Podstawowe źródło informacji oraz możliwość pobrania: <http://qt-project.org/>





The screenshot shows the Qt download page. At the top, there is a navigation bar with a home icon and the text "Download Qt, the cross-platform application framework". Below this, the main heading "Download Qt, the cross-platform application framework" is displayed. A "Jump to:" section contains links for "Qt libraries", "Qt Creator", and "Other downloads". On the left, there are two orange buttons labeled "Qt Library" and "Qt Creator". The main content area is titled "Qt 5.0" and provides instructions on how to select the correct binary package based on the operating system. It lists five download links: "Qt 5.0.1 for Linux 32-bit (388 MB)", "Qt 5.0.1 for Linux 64-bit (388 MB)", "Qt 5.0.1 for Mac (398 MB)", "Qt 5.0.1 for Windows 32-bit (MinGW 4.7, 823 MB)", and "Qt 5.0.1 for Windows 32-bit (VS 2010, 481 MB)". Below these links, it mentions that source code is available as a zip file for Windows and a tar.gz file for Linux/Mac. A "License" box on the right explains the licensing options (GPL v3, LGPL v2, and commercial license) and provides a link to "Evaluate Qt under a commercial license (qt.digia.com)". A "Notes" box at the bottom explains the contents of the MinGW binary package.

Download Qt, the cross-platform application framework

## Download Qt, the cross-platform application framework

Jump to: [Qt libraries](#) ↓ [Qt Creator](#) ↓ [Other downloads](#) ↓



### Qt 5.0

Select the file according to your operating system from the list below to get the latest Qt 5.0 for your computer. The binary packages include Qt 5.0.1 libraries and Qt Creator 2.6.2

- Qt 5.0.1 for Linux 32-bit (388 MB)
- Qt 5.0.1 for Linux 64-bit (388 MB)
- Qt 5.0.1 for Mac (398 MB)
- Qt 5.0.1 for Windows 32-bit (MinGW 4.7, 823 MB)
- Qt 5.0.1 for Windows 32-bit (VS 2010, 481 MB)

The source code is available as a [zip](#) (270 MB) file for Windows users or as [tar.gz](#) (219 MB) file for Linux/Mac users. You can also get split source packages and all the installers [here](#). Or visit the repository at [qt.gitorious.org/qt](http://qt.gitorious.org/qt).

Qt is available under GPL v3, LGPL v2 and a [commercial license](#). Learn more about licenses [here](#).

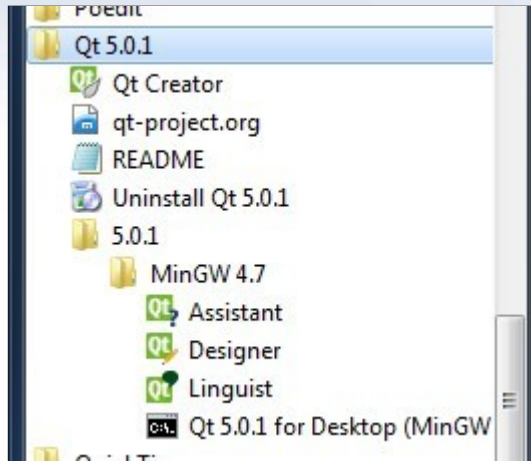
#### Develop with a Qt commercial license

The Qt commercial license offers varied licensing terms and includes support and product updates. The commercial license of Qt is the correct license to use for the development of proprietary and/or commercial software with Qt where you need to safeguard your development investment to secure your competitive advantage.

[Evaluate Qt under a commercial license \(qt.digia.com\)](http://qt.digia.com)

The MinGW binary package includes a 32 bit MinGW-builds (gcc 4.7.2, sjlj exception handling, posix threading) toolchain. Sources are available from the [MinGW-builds project](#) or from [releases.qt-project.org](http://releases.qt-project.org).

# Zawartość pakietu



- **QtCreator** — główne środowisko IDE.
- **QtAssistant** — narzędzie wspomagające pracę z podpowiedziami i dokumentacją, zintegrowane z QtCreator'em.
- **QtDesigner** — narzędzie typu RAD, pozwalające na tworzenie warstwy wizualnej aplikacji, zintegrowane z QtCreator'em.
- **QtLinguist** — narzędzie do przygotowywania różnych wersji językowych dla tworzonych programów.
- W środowisku Windows — narzędzie typu *command-line* z odpowiednio ustalonymi zmiennymi środowiskowymi.

# Programowanie sterowane zdarzeniami w Qt

- Qt realizuje funkcję głównej pętli zdarzeń, odpowiedzialnej za pobieranie informacji o zdarzeniach, ustalanie ich priorytetów, kolejkovanie, oraz ustalanie procedur obsługi poszczególnych zdarzeń.
- Programowanie GUI w Qt jest typowe dla EDP (ang. event driven programming) — interfejs jest pasywny i reaguje na konkretne zdarzenia generowane przez mysz, klawiaturę, gesty, ekran dotykowy, zdarzenia programowe.
- Typowy program Qt tworze odpowiednie obiekty, łączy je oraz uaktywnia główną pętlę komunikatów obiektu klasy QApplication, poprzez wywołanie funkcji `exec()`:

```
int main(int argc, char *argv[])
{
    QApplication application(argc, argv);

    WindowDefinedByProgrammer window;
    window.show();

    return application.exec();
}
```

# Qt Meta-Object System

Rozszerzenia Qt oferowane przez Meta-Object System bazują na:

- Klasie *QObject*, stanowiącej bazą klasę dla wszystkich klas obiektów wykorzystujących rozszerzenia Qt.
- Makrodefinicji `Q_OBJECT` osadzonej w prywatnej sekcji klasy, makro pozwala na stosowanie mechanizmów typowych dla Qt.
- Kompilatorowi MOC (Meta-Object Compiler) realizującemu wstępne przetwarzanie kodu wykorzystującego rozszerzenia oferowane przez bibliotekę Qt.
- MOC przetwarza pliki źródłowe zawierające klasy wykorzystujące makro `Q_OBJECT`.
- MOC odpowiedzialny jest za działanie mechanizmu sygnałów i slotów, systemu dynamicznych właściwości, informacji RTTI.
- Aby wszystko poprawnie działało, Qt oferuje własną wersję programu `make` – *qmake*.

# QObject

- QObject to bazowa klasa dla wielu istotnych w Qt klas, np.: QEvent , QApplication, QLayout, QWidget.
- QObject pozwala na programowanie sterowane zdarzeniami, działanie klasy QApplication oraz realizację pętli komunikatów.
- QObject nie posiada publicznego konstruktora kopiującego oraz operatora przypisania — obiekty tej klasy nie mogą być kopiowane, przekazywane przez wartość jako parametr oraz rezultat funkcji.
- Każde wystąpienie klasy QObject to obiekt o unikatowej tożsamości.

```
class QObject
{
public:
    explicit QObject(QObject* parent=0);
    QObject * parent () const;
    QString objectName() const;
    . . .
};
```

QObject posiada jednoargumentowy konstruktor zdefiniowany jako **explicit**, co nie pozwala na stosowanie niejawnych konwersji konstruktorowych.

# QObject, rodzic, dzieci, GUI

- Każdy QObject posiada najwyżej jednego rodzica klasy QObject oraz potencjalnie dużą liczbę potomków (dzieci), również klasy QObject.
- Każdy QObject przechowuje wskaźniki do swoich potomków w liście typu QObjectList (dokładniej: QList<QObject\*>).
- Każdy rodzic zna adresy swoich potomków, każdy potomek zna adres swojego rodzica.
- Wszystkie elementy GUI w Qt pochodzą od klasy QWidget, a ta jest potomkiem QObject.
- Relacje rodzic-dziecko w przypadku GUI są łatwe do zrozumienia i odpowiadają hierarchii GUI.

# Okno z przyciskiem zamykania - wersja 1

- Wykorzystanie *slotu* jako *funkcji przypisanej do sygnału* — rozwiązane analogiczne do procedury obsługi zdarzenia (VCL, VisualStudio).

The image illustrates the process of connecting a button's click signal to a slot function in Qt. It consists of four main parts:

- Qt Designer:** A window titled "Wpisz tutaj" containing a button labeled "Quit". A context menu is open over the button, with the "Przejdź do slotu..." option highlighted.
- Przejdź do slotu dialog:** A dialog box titled "Przejdź do slotu" with a list of signals. The "clicked()" signal from "QAbstractButton" is selected.
- Code Editor:** A code editor showing the implementation of the `MainWindow` class. The `on_pushButton_clicked()` slot function is defined to call `close()`.
- Class Declaration:** A snippet of the `MainWindow` class declaration showing the `private slots:` section with `void on_pushButton_clicked();`.

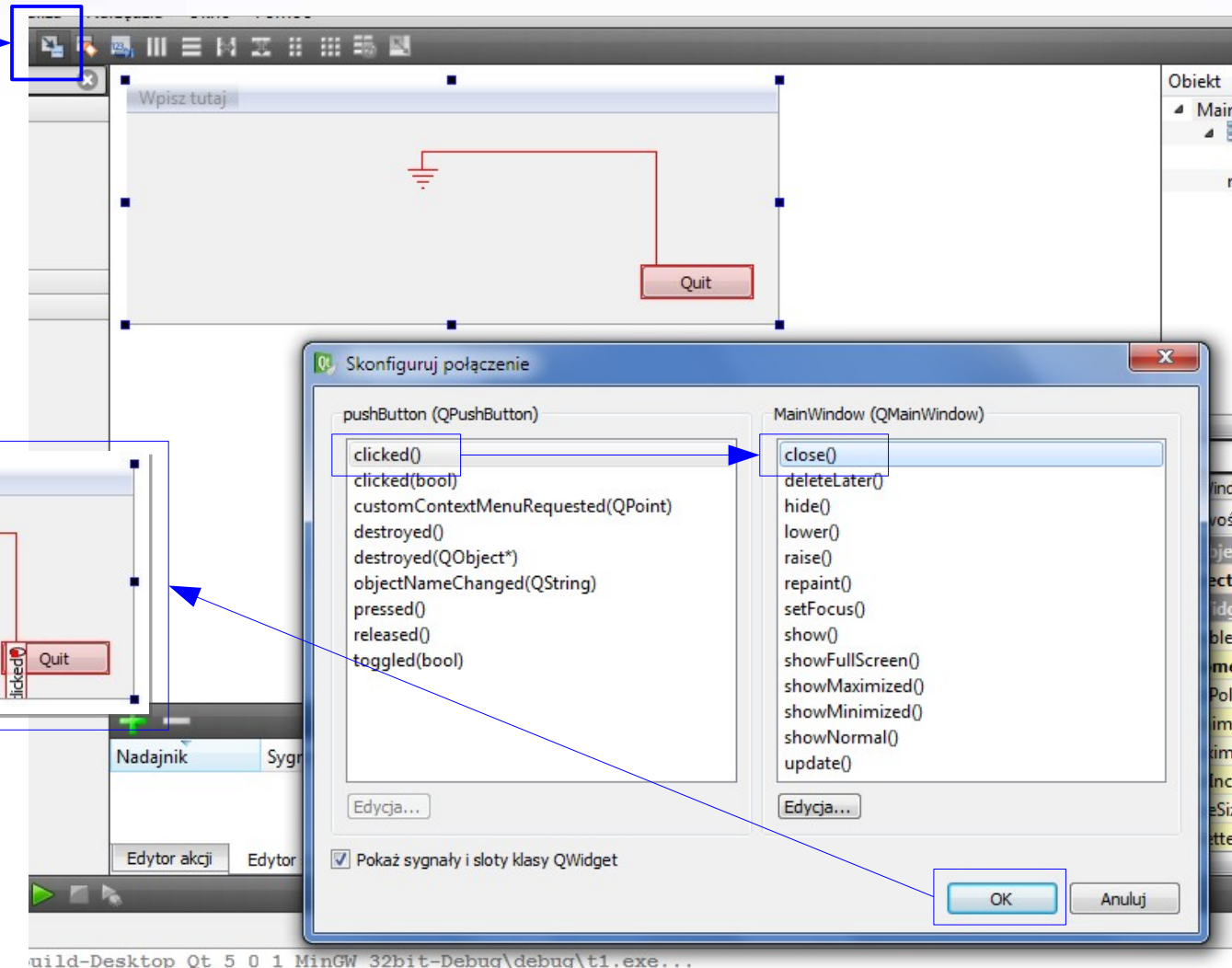
```
11 MainWindow::~MainWindow()
12 {
13     delete ui;
14 }
15
16
17 void MainWindow::on_pushButton_clicked()
18 {
19     close();
20 }
21
22
23 class MainWindow :
24 {
25     Q_OBJECT
26
27 public:
28     explicit MainWindow(QWidget *parent = 0);
29     ~MainWindow();
30
31 private slots:
32     void on_pushButton_clicked();
33
34 private:
35     Ui::MainWindow *ui;
36 };
37
```



# Okno z przyciskiem zamykania - wersja 2

- Wykorzystanie powiązania *sygnał-slot* — rozwiązane z wykorzystaniem QtDesigner'a.

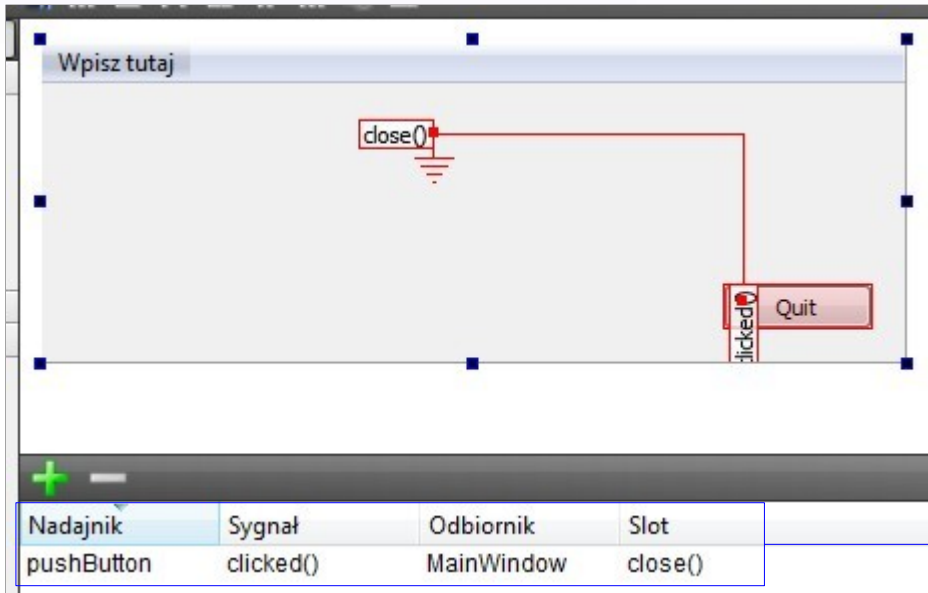
Tryb edycji powiązań sygnał-slot



uild-Desktop Qt 5 0 1 MinGW 32bit-Debug\debug\t1.exe...

# Okno z przyciskiem zamykania - wersja 2

- Informacje o powiązaniu *sygnał-slot* zapisywane są do pliku definicji okna głównego *mainwindow.ui*.



```
mainwindow.ui*
8     </rect>
9     </property>
0     </widget>
1 </widget>
2 <layoutdefault spacing="6" margin="11"/>
3 <resources/>
4 <connections>
5 <connection>
6     <sender>pushButton</sender>
7     <signal>clicked()</signal>
8     <receiver>MainWindow</receiver>
9     <slot>close()</slot>
0 <hints>
1 <hint type="sourcelabel">
2     <x>350</x>
3     <y>130</y>
4 </hint>
5 <hint type="destinationlabel">
6     <x>196</x>
7     <y>46</y>
8 </hint>
9 </hints>
0 </connection>
```

# Okno z przyciskiem zamykania - wersja 3

- Wykorzystanie powiązania *sygnał-slot* — wykorzystaniem makra *connect* w kodzie programu.

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

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(close()));
}

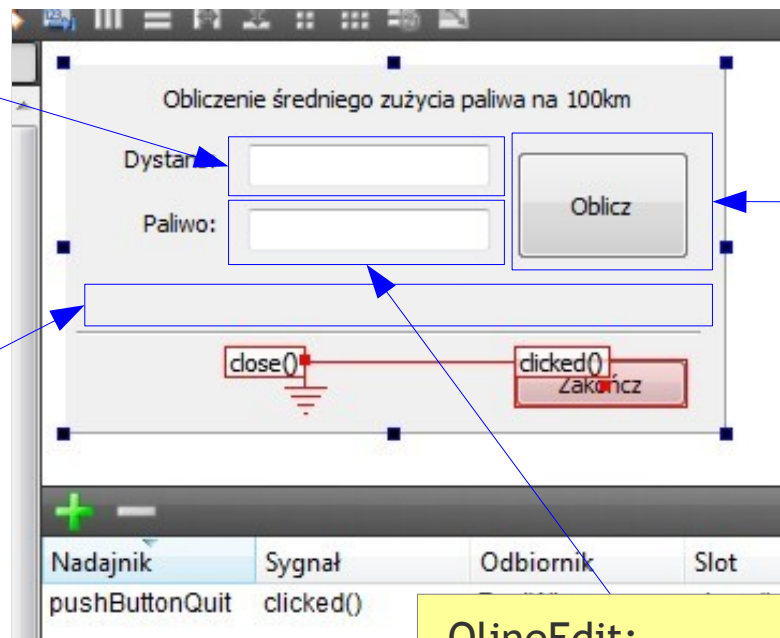
MainWindow::~MainWindow()
{
    delete ui;
}
```

# Program „Średnie spalanie” – wersja 1

- Obliczanie ile paliwa średnio zużył pojazd na dystansie 100km. Potrzebne informacje: przejechany dystans i ilość zużytego paliwa.
- Interfejs użytkownika:

QlineEdit:  
lineEditDistance

Pusty QLabel: labelResult



QPushButton:  
pushButtonCalculate

QlineEdit:  
lineEditFuel

# Program „Średnie spalanie” – wersja 1

- Główne okno aplikacji jest obiektem klasy *FuelWin*, pochodnej w stosunku do *QDialog*.

```
#ifndef FUELWIN_H
#define FUELWIN_H
#include <QDialog>

namespace Ui {
class FuelWin;
}

class FuelWin : public QDialog
{
    Q_OBJECT

public:
    explicit FuelWin(QWidget *parent = 0);
    ~FuelWin();
private slots:
    void on_pushButtonCalculate_clicked();

private:
    Ui::FuelWin *ui;
};

#endif // FUELWIN_H
```

Prototyp funkcji aktywowanej  
obliczającej średnie spalanie

# Program „Średnie spalanie” – wersja 1

- Obliczenia zrealizowane w funkcji (slocie) statycznie powiązanej z sygnałem generowanym przez przycisk *pushButtonCalculate*.

```
void FuelWin::on_pushButtonCalculate_clicked()
{
    QString resultStr;
    bool convOK;
    float dist = ui->lineEditDistance->text().toFloat( &convOK );
    float fuel = ui->lineEditFuel->text().toFloat( &convOK );

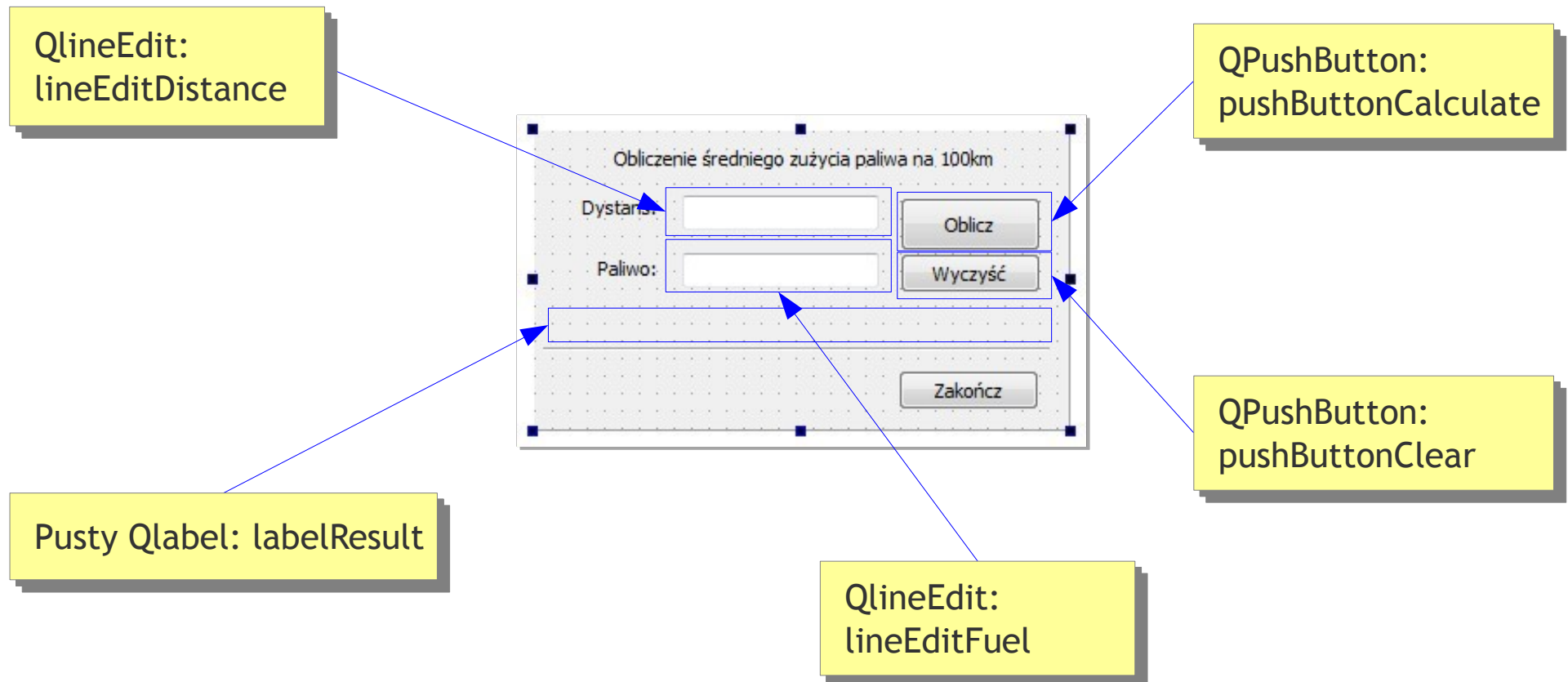
    if( convOK && dist != 0 )
    {
        resultStr.setNum( ( fuel * 100 ) / dist, 'f', 2 );
        resultStr = "Średnie spalanie: " + resultStr;
    }
    else
        resultStr = "Błędne dane";

    ui->labelResult->setText( resultStr );

    ui->lineEditDistance->setFocus( Qt::TabFocusReason );
}
```

# Program „Średnie spalanie” – wersja 2

- Dodatkowy przycisk czyszczenia elementów okna.
- Obliczenia i czyszczenie zrealizowane w funkcjach (slotach) dynamicznie powiązanych z sygnałami *clicked()* przycisków *pushButtonCalculate* oraz *pushButtonClear*.



# Program „Średnie spalanie” – wersja 2

- Definicja własnych slotów *calculate* i *clear* – prototypy:

```
. . .  
  
class FuelWin : public QDialog  
{  
    Q_OBJECT  
  
public:  
    explicit FuelWin(QWidget *parent = 0);  
    ~FuelWin();  
  
private slots:  
    void calculate();  
    void clear();  
  
private:  
    Ui::FuelWin *ui;  
};  
. . .
```



# Program „Średnie spalanie” – wersja 2, definicja slotów

```
void FuelWin::calculate()
{
    QString resultStr;
    bool convOK;
    float dist = ui->lineEditDistance->text().toFloat( &convOK );
    float fuel = ui->lineEditFuel->text().toFloat( &convOK );
    if( convOK && dist != 0 )
    {
        resultStr.setNum( ( fuel * 100 ) / dist, 'f', 2 );
        resultStr = "Średnie spalanie: " + resultStr;
    }
    else
        resultStr = "Błędne dane";
    ui->labelResult->setText( resultStr );
    ui->lineEditDistance->setFocus( Qt::TabFocusReason );
}

void FuelWin::clear()
{
    ui->lineEditDistance->setText( "" );
    ui->lineEditFuel->setText( "" );
    ui->labelResult->setText( "" );
    ui->lineEditDistance->setFocus( Qt::TabFocusReason );
}
```

# Program „Średnie spalanie” – wersja 2

- Powiązanie sygnałów i slotów:

```
FuelWin::FuelWin(QWidget *parent)
: QDialog(parent),
  ui(new Ui::FuelWin)
{
    ui->setupUi(this);

    connect( ui->pushButtonCalculate, SIGNAL(clicked()),
             this, SLOT(calculate()));
    connect( ui->pushButtonClear, SIGNAL(clicked()),
             this, SLOT(clear()));
}
```

# Program „Średnie spalanie” – wersja 2

- Obliczenia zrealizowane w funkcji (slocie) statycznie powiązanej z sygnałem generowanym przez przycisk *pushButtonCalculate*.

```
void FuelWin::on_pushButtonCalculate_clicked()
{
    QString resultStr;
    bool convOK;
    float dist = ui->lineEditDistance->text().toFloat( &convOK );
    float fuel = ui->lineEditFuel->text().toFloat( &convOK );

    if( convOK && dist != 0 )
    {
        resultStr.setNum( ( fuel * 100 ) / dist, 'f', 2 );
        resultStr = "Średnie spalanie: " + resultStr;
    }
    else
        resultStr = "Błędne dane";

    ui->labelResult->setText( resultStr );

    ui->lineEditDistance->setFocus( Qt::TabFocusReason );
}
```

Dziękuję za uwagę

Pytania? Polemiki?  
Teraz, albo:  
[roman.siminski@us.edu.pl](mailto:roman.siminski@us.edu.pl)