

## Laboratorium 10 Pomoc. Wprowadzenie do biblioteki QT

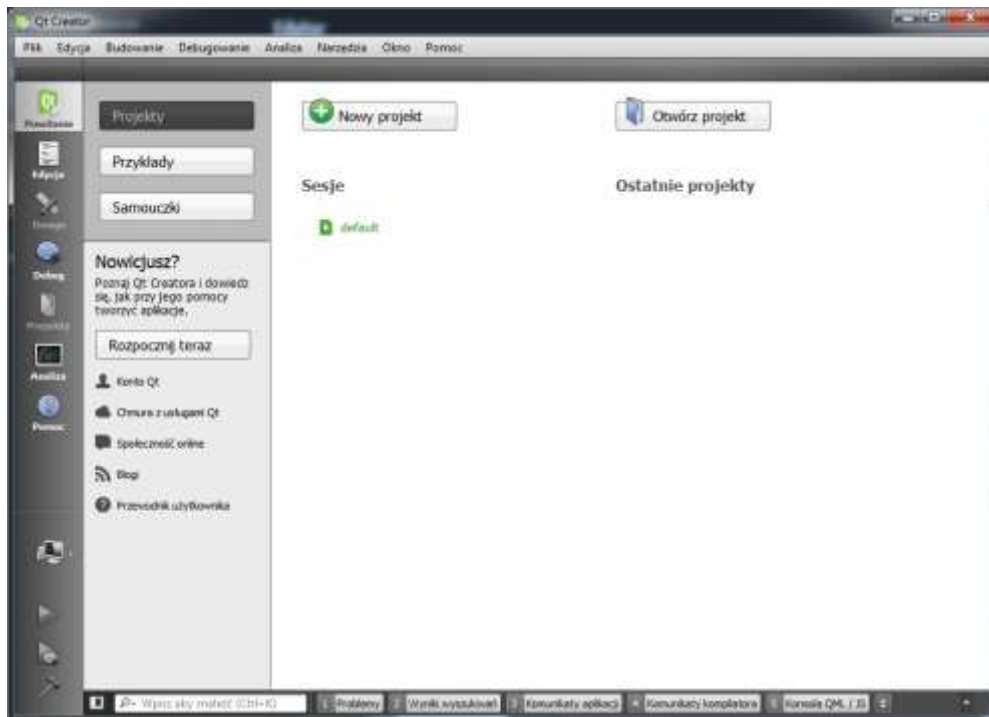
### Cel laboratorium:

Zapoznanie z możliwością wykorzystania biblioteki QT

**QT** to wieloplatformowy zestaw bibliotek i narzędzi dla języka C++ służących do tworzenia aplikacji z GUI (graficznym interfejsem użytkownika).

**QT Creator** - środowisko programistyczne IDE + GUI +RAD

### Interfejs QT Creatora:



### Rodzaje projektów:

- *Qt Widgets Application* - aplikacja desktopowa, główne okno bazujące na Designerze
- *Qt Quick Application*
- *Qt Quick Controls Application*
- *Qt Canvas 3D Application*
- *Qt Console Application*

### Pliki aplikacji:

- **nazwaProjektu.pro** - plik projektu
- **MainWindow.h** - definicja klasy okna głównego (class MainWindow w namespace Ui)
- **MainWindow.cpp** - implementacja metod klasy MainWindow
- **MainWindow.ui** - plik w formacie xml z opisem interfejsu
- **main.cpp** - plik z funkcją main() - tworzenie głównego okna programu i pętla obsługi zdarzeń

## Ogólna struktura main.cpp

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

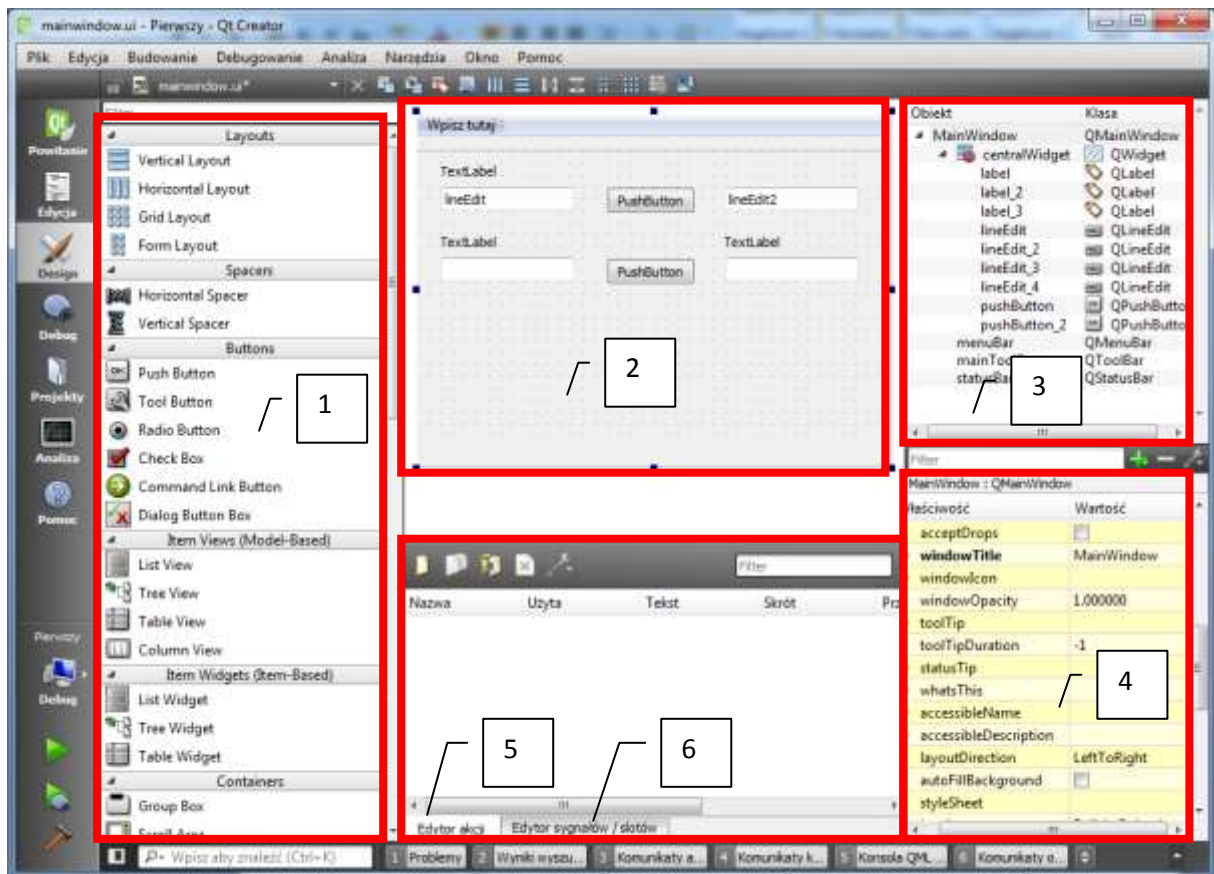
int main(int argc, char *argv[])
{
    QApplication a(argc, argv); //utworzenie obiektu klasy QApplication
    MainWindow w;                //utworzenie okna aplikacji
    w.show();                    //wymuszenie ukazania się okna

    return a.exec();             //uruchomienie pętli obsługi zdarzeń
}
```

### Etapy tworzenia aplikacji:

1. Projektowanie GUI (edycja pliku **MainWindow.ui**)
2. Programowanie procedur obsługi zdarzeń (edycja pliku **MainWindow.cpp**)

### 1. Projektowanie GUI



Okna IDE Designera:

1. Paleta komponentów
2. Projektant zawartości okna aplikacji
3. Hierarchia obiektów umieszczonych w oknie
4. Właściwości komponentów (obiektów)
5. Edytor akcji
6. Edytor sygnałów i slotów

Bazową klasą większości klas w QT jest **QObject** zawierająca wiele mechanizmów wykorzystywanych w QT: zdarzenia, sygnały, sloty, właściwości). Interfejs GUI jest budowany z dostępnych widgetów umieszczonych w hierarchii (Wszystkie elementy GUI w Qt pochodzą od klasy **QWidget**, a ta jest potomkiem **QObject**).

QObject posiada **właściwości** z metodami **get** (text, color, ...)/**set** (setText, setColor, ...) np.

```
class QLabel : public QFrame
{
    Q_OBJECT          //makro 1
    Q_PROPERTY(QString text READ text WRITE setText) //makro2
public:
    QString text() const; //get, const - przekazuje wartość pola
public slots:
    void setText(const QString &); //set - przekazuje wartość przez
                                   //argument
};
```

#### Wykorzystanie właściwości:

- dostęp bezpośredni

```
QString text = label->text();
label->setText("Hello World!");
```

- dostęp przez meta info i metodę property

```
QString text = object->property("text").toString();
object->setProperty("text", "Hello World!");
```

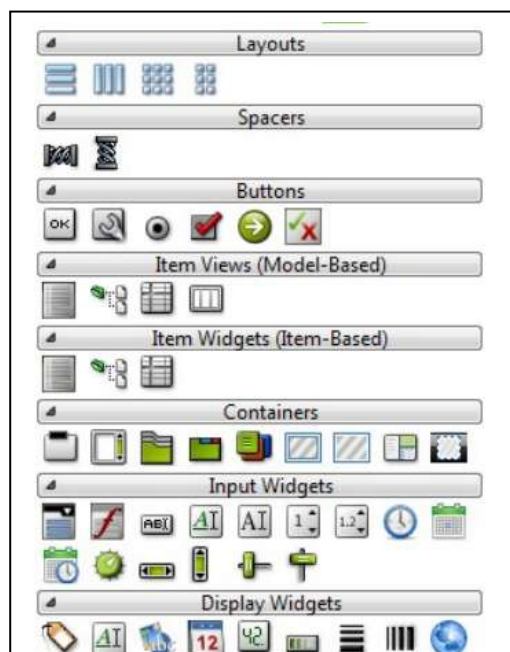
- określenie właściwości podczas uruchamiania

```
int QMetaObject::propertyCount();
QMetaProperty QMetaObject::property(i);
QMetaProperty::name/isConstant/isDesignable/read/write/...
```

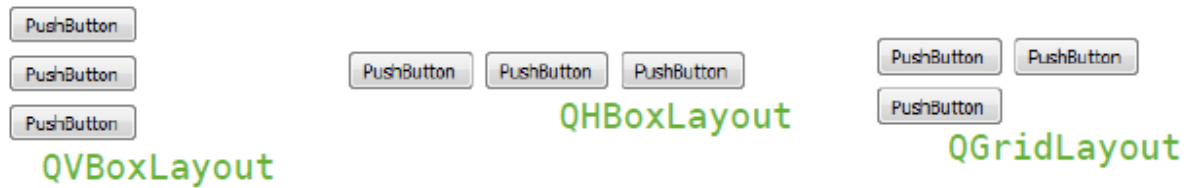
#### Popularne widżety:

#### Popularne klasy widżetów:

- QLabel
- QLineEdit
- QTextEdit
- QPushButton
- QRadioButton
- QComboBox
- QScrollBar
- QGroupBox
- ...



**Layout'y** - dopasowują widżety do rozmiaru okna, tekstu, ... Rodzaje layout'ów:



**Spacer Springs** ustalają puste przestrzenie

## 2. Programowanie procedur obsługi zdarzeń

Aplikacja - zbiór procedur obsługi zdarzeń.

Mechanizm **signals & slots** pozwala na obsługę reakcji na zdarzenia.

- **Sygnały** to **zdarzenia** zachodzące na obiektach interfejsu - funkcje bez ciała typu void
- **Sloty** to **funkcje obsługi tych zdarzeń**, funkcje przechwytyjące sygnał i wykonujące określone operacje - funkcje typu void.

Definicja sygnałów i slotów wewnątrz klasy:

```
class Qnazwa{
public slots:
signals:
}
```

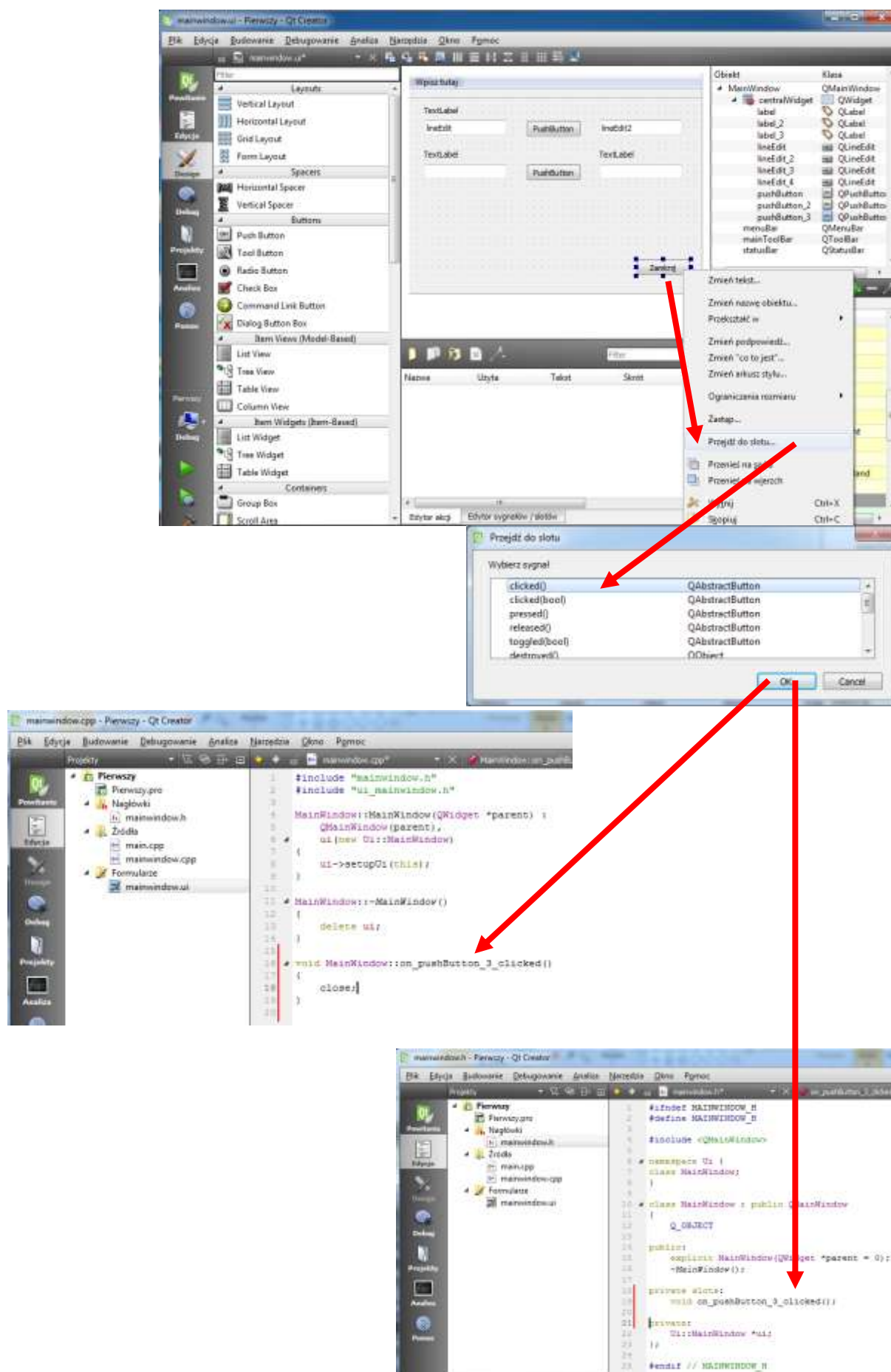
Łączenie sygnałów i slotów realizowane jest w funkcji `connect()`

```
connect(addr_nadawcy, SIGNAL( signal() ),
        addr_odbiornicy, SLOT( slot() ) );
```

gdzie:

- `addr_nadawcy (sender)`- wskaźnik do obiektu, który nadaje sygnał (np. `PushButton1`)
- `SIGNAL( signal() )` - sygnał wysyłany przez nadawcę
- `addr_odbiornicy (receiver)` - adres odbiorcy sygnału, który zawiera slot reagujący na sygnał (`this`)
- `SLOT( slot() )` - metoda uruchomiona po wysłaniu sygnału (`void on_PushButton_clicked()`)

Automatyczne tworzenie slotów: Wskazanie komponentu/prawy klawisz myszy **Przejdź do slotu**/wybór sygnału z listy -> automatyczne wygenerowanie funkcji slotu



### Aplikacje obliczeniowe

1. Pobranie tekstu z okna `lineEdit` i konwersja tekstu na liczbę  
- metody klasy `QString`: `toInt()`, `toFloat()`, `toDouble()`  
`float r = ui->lineEdit->text().toFloat();`
2. Wykonanie obliczeń  
`float pole=M_PI*pow(r,2);`
3. Konwersja liczby na tekst i przypisanie go do właściwości `text` okna `lineEdit_2`  
metoda statyczna `QString::number(float)`  
`ui->lineEdit_2->setText(QString::number(pole) );`

