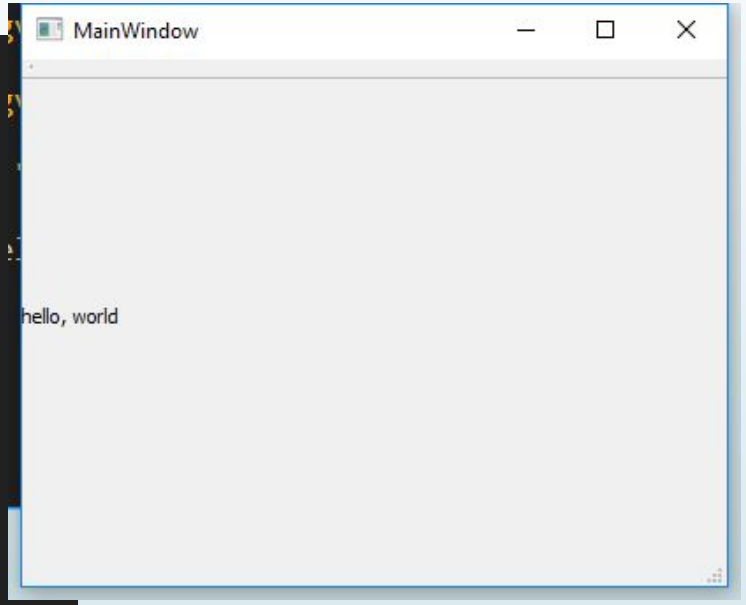
Hello, Qt!

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    QLabel label("Hello, Qt!");

    w.setCentralWidget(&label);
    w.show();

    return a.exec();
}
```



Модули Qt

- QtCore
- QtGui
- QtWidgets
- QtNetwork
- QtOpenGL
- QtWebKit
- QtWebKitWidgets
- ….

QtCore

- Является базовым и не содержит классы относящиеся к интерфейсу пользователя. Содержит:
  - классы контейнеры: QList, QVector, QMap, QString…
  - классы для ввода/вывода информации: QIODevice, QTextStream, QFile…
  - классы для программирования многопоточности: QThread, QWaitCondition, QMutex
  - классы для работы с таймерами: QBasicTimer, QTimer
  - QDate, QTime
  - QObject
  - QEvent
  - …..

QCoreApplication

- Создается в приложении только один раз
- Срок жизни = время работы программы
- Управляет событиями между приложением и ОС
- Передает и представляет аргументы командной строки
- Может быть унаследован
- Методы могут быть переопределены

QtGui

- Интеграция с оконной системой
- QWindow
  - позволяет получить ввод данных от пользователя
  - позволяет производить графические операции и рисование на своей поверхности

QGuiApplication

- Содержит механизм цикла событий
  - получение доступа к буферу обмена
  - инициализация настроек приложения (например, палитра)
  - управление формой курсора

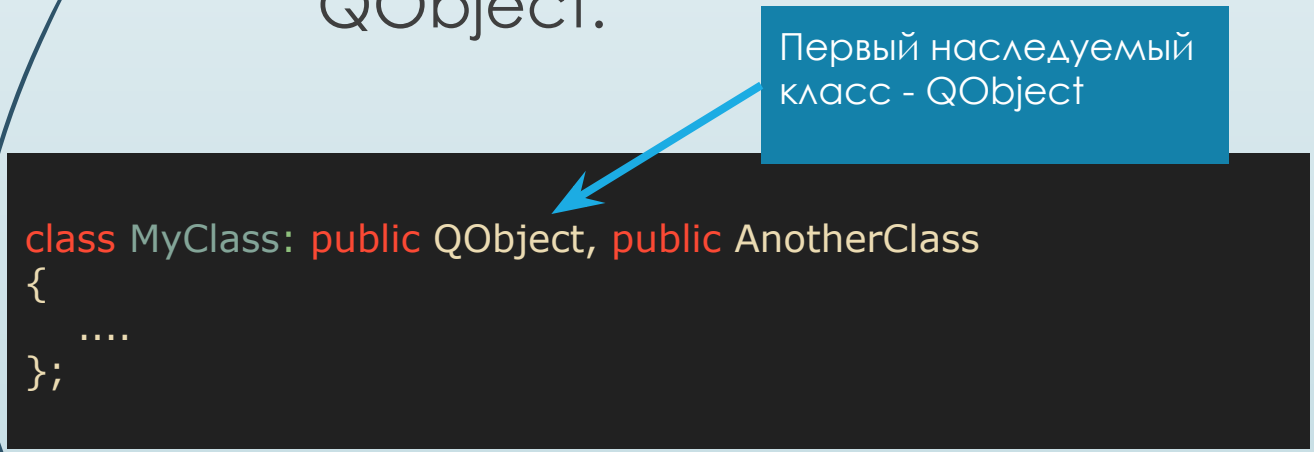
QWidgets

- Содержит классы виджетов:
  - QVBoxLayout, QHBoxLayout
  - QLabel
  - QPushButton, QCheckBox, QRadioButton
  - QScrollBar
  - QComboBox
  - QMainWindow
  - QDialog
  - ….

QObject

- QObject - основной, базовый класс. Большинство классов Qt являются его наследниками.

- Классы, имеющие слоны и сигналы должны быть унаследованы от QObject.



QObject

Поддерживает:

- сигналы и слоты
- таймеры
- события и их фильтрацию
- приведение типов
- ….

QWidget

- QWidget унаследован от QObject.
- Виджет без предка = виджет верхнего уровня.
- При уничтожении родительского виджета - дочерние уничтожаются
- Если родительский виджет стал невидимым/недоступным - дочерние перенимают состояние

```
QWidget(QWidget* parent = nullptr, Qt::WindowFlags f = Qt::WindowFlags());
```

Указатель на родительский виджет

Флаги

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

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


    //установка флагов
    w.setWindowFlags(Qt::Window | Qt::WindowTitleHint);

    //установка названия окна
    w.setWindowTitle("First Qt Lection");

    QLabel label("hello, world");

    //устанавливаем виджет label как основной
    w.setCentralWidget(&label);

    //отображаем окно
    w.show();

    //возвращаем результат выполнения
    return a.exec();
}
```

QAbstractButton

- QAbstractButton - базовый класс всех кнопок:
  - QPushButton
  - QCheckBox
  - QRadioButton

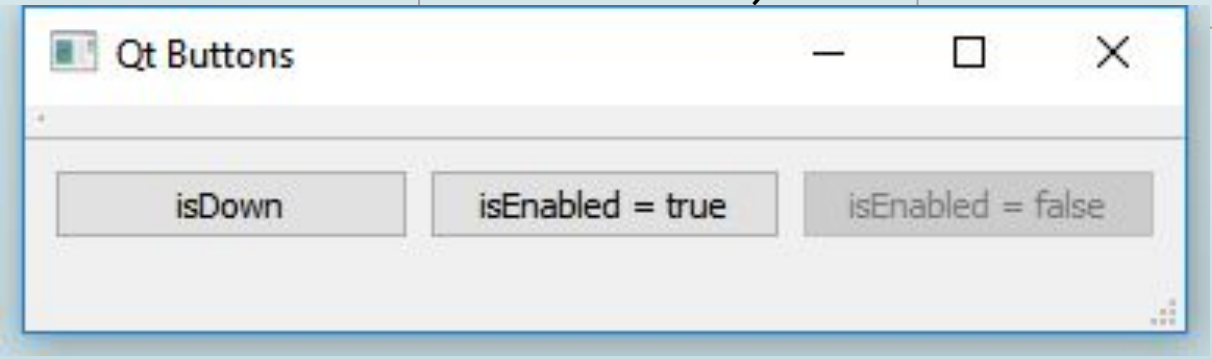
- QAbstractButton - реализует методы общие для всех кнопок

QAbstractButton сигналы

- `pressed()` - на кнопку нажали
- `released()` - кнопку отпусти
- `toggled()` - изменение состояние кнопки, имеющей статус выключателя
- `clicked()` (*Pressed & Released*) - нажатие на кнопку

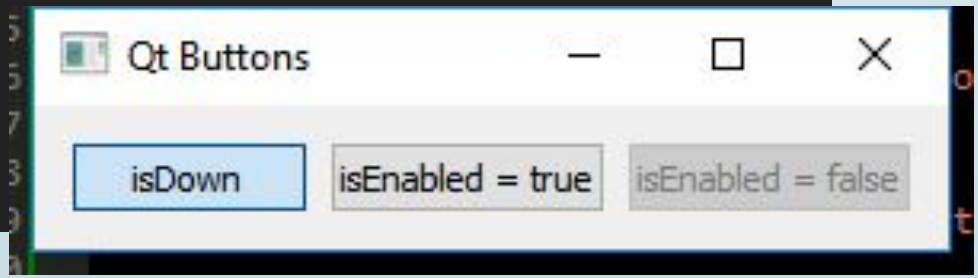
QAbstractButton состояния

`isDown()`	`true` - кнопка нажата	`setDown()`
`isChecked()`	`true` - кнопка включена(выбрана)	`setChecked()`
`isEnabled()`	`true` - если кнопка доступна (реагирует на действия пользователя)	`setEnabled()`



QPushButton

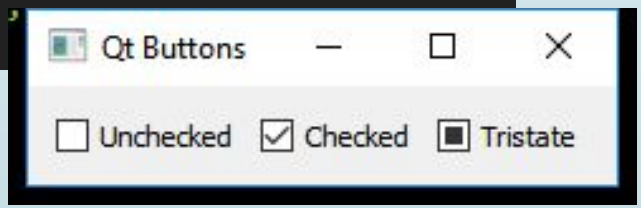
```
//создание кнопки
QPushButton *button1 = new QPushButton("isDown");


 //выставляем состояние кнопки isDown = true
 button1->setDown(true);

QPushButton *button2 = new QPushButton("isEnabled = true");

QPushButton *button3 = new QPushButton("isEnabled = false");


/* выставляем состояние кнопки isEnabled = false, кнопка
* становится недоступна пользователю.
* Изменить состояние на isEnabled = true можно только из
* программы */
button3->setEnabled(false);
```



QCheckBox

- Может быть выбрано сразу несколько QCheckBox
- Может быть в состояние:
  - Checked()
  - Unchecked()
  - partiallyCheked()

```
QCheckBox *ch1 = new QCheckBox("Unchecked");
QCheckBox *ch2 = new QCheckBox("Checked");
ch2->setChecked(true);

QCheckBox *ch3 = new QCheckBox("Tristate");
ch3->setTristate(true);
ch3->setCheckState(Qt::PartiallyChecked);
```



QRadioButton

- Можно выбрать только одну кнопку
- Может быть в состоянии:
  - Checked
  - Unchecked
- Можно объединять в группы QGroupBox

QRadioButton

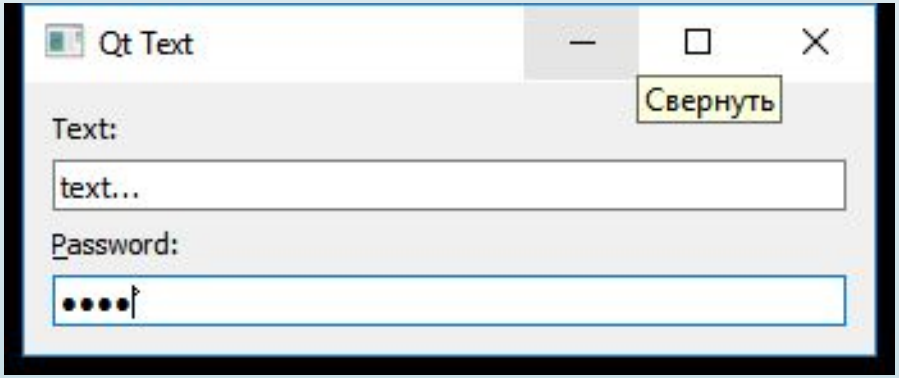
```
//объект для объединения виджетов в группу
 QGroupBox *groupBox = new QGroupBox();
 //слой для распложения группы радио-кнопок
 QVBoxLayout *vbox1 = new QVBoxLayout();

 QRadioButton *radio1 = new QRadioButton("Radio button 1");
 QRadioButton *radio2 = new QRadioButton("Radio button 2");
 QRadioButton *radio3 = new QRadioButton("Radio button 3");

 //выставляем кнопку в состояние "Выбрана"
 radio1->setChecked(true);

 //добавляем радио-кнопки на слой
 vbox1->addWidget(radio1);
 vbox1->addWidget(radio2);
 vbox1->addWidget(radio3);


 //устанавливаем слой для группы
 groupBox->setLayout(vbox1);

 //добавляем слой с радио-кнопками на основной слой
 layout->addWidget(groupBox);
```

QLineEdit

- Однострочное поле ввода
- Не стоит использовать для ввода более чем одной строки
- Сигналы:
  - `textChanged()`
  - `textEdited()`
  - `returnPressed()`
- `setEchoMode(QLineEdit::Password)` - режим ввода пароля

QLineEdit

```
QLabel *label_text = new QLabel("Text:");
//однострочное текстовое поле
QLineEdit *edit_text = new QLineEdit();
//связываем лайбл и текстовое поле
label_text->setBuddy(edit_text);


QLabel *label_password = new QLabel("&Password:");
QLineEdit *edit_password = new QLineEdit();
label_password->setBuddy(edit_password);
edit_password->setEchoMode(QLineEdit::Password);
```

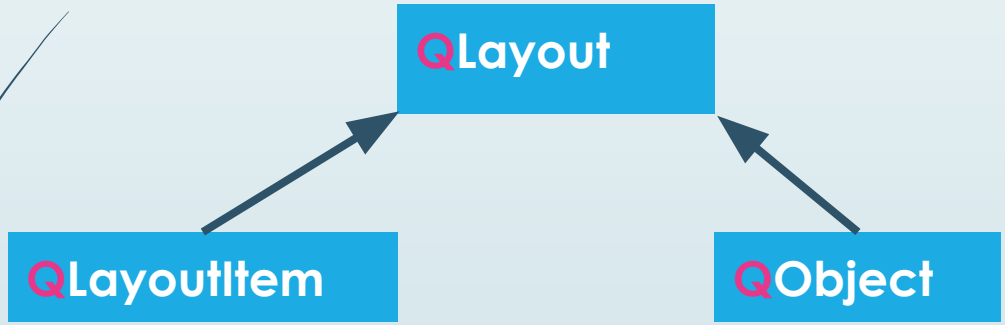


Расположение виджета

```
QWidget base;
QWidget *blue_widget = new QWidget(&base);
QPalette palette;

palette.setColor(blue_widget->backgroundRole(), Qt::blue);

//    auto x = blue_widget->backgroundRole();
//    palette.setColor(QPalette::ColorRole::Background, Qt::blue);

//устанавливаем палитру для виджета
blue_widget->setPalette(palette);
//заполнение области цветом
blue_widget->setAutoFillBackground(true);
//изменение размера виджета
blue_widget->resize(40, 40);
//изменение расположения виджета
blue_widget->move(10, 50);
```

Размещение элементов

- `Layouts` - классы компоновки, определяют расположение виджетов относительно друг друга



Layout

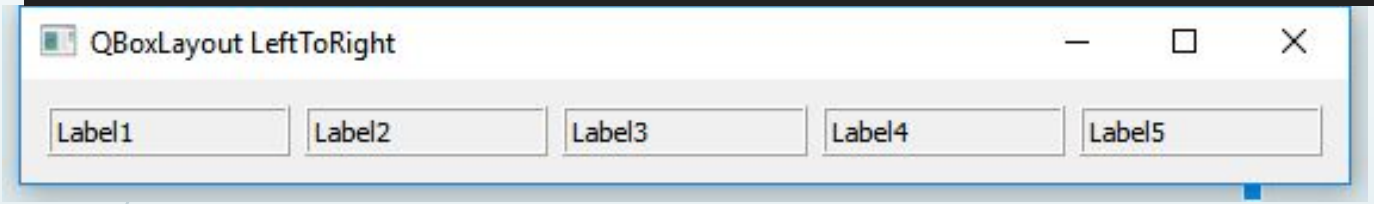
- Отвечают за правильное размещение виджетов
- Отвечают за присвоение объектам предков

QBoxLayout

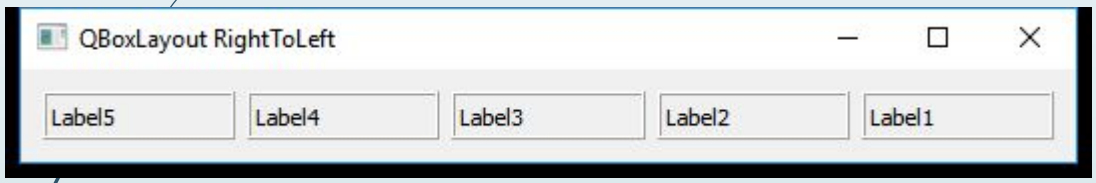
- LeftToRight - горизонтальное размещение, слева направо
- RightToLeft - горизонтальное размещение, справа налево
- TopToBottom - вертикальное размещение, сверху вниз
- BottomToTop - вертикальное размещение, снизу вверх

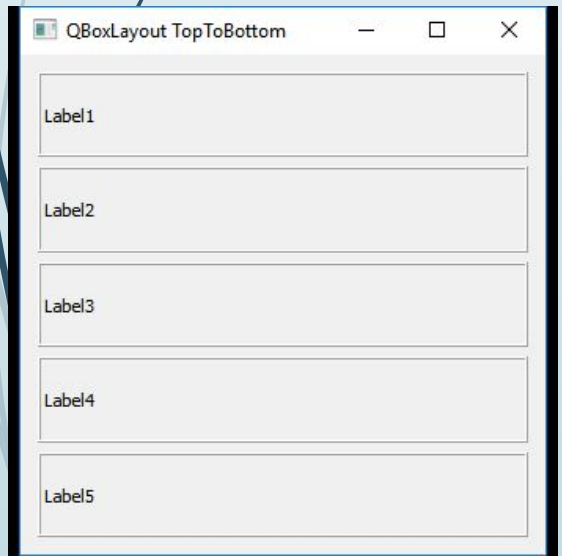
QBoxLayout

```
QBoxLayout *layout = new QBoxLayout(QBoxLayout::LeftToRight);
```




```
QBoxLayout *layout = new QBoxLayout(QBoxLayout::RightToLeft);
```





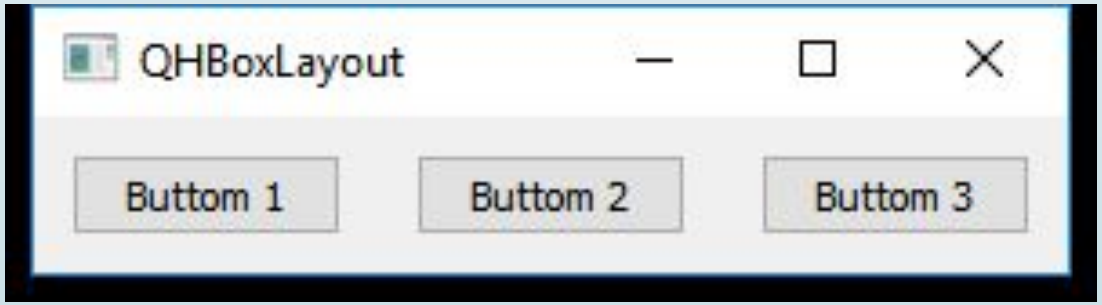

```
QBoxLayout *layout =
new QBoxLayout(QBoxLayout::TopToBottom);
```

QBoxLayout

```
  QBoxLayout *layout = new
QBoxLayout(QBoxLayout::TopToBottom);

   QLabel *label1 = new QLabel("Label1");
   label1->setFrameStyle(QFrame::Box | QFrame::Raised);

   QLabel *label2 = new QLabel("Label2");
   label2->setFrameStyle(QFrame::Box | QFrame::Raised);

   QLabel *label3 = new QLabel("Label3");
   label3->setFrameStyle(QFrame::Box | QFrame::Raised);

   QLabel *label4 = new QLabel("Label4");
   label4->setFrameStyle(QFrame::Box | QFrame::Raised);

   QLabel *label5 = new QLabel("Label5");
   label5->setFrameStyle(QFrame::Box | QFrame::Raised);

   layout->addWidget(label1);
   layout->addWidget(label2);
   layout->addWidget(label3);
   layout->addWidget(label4);
   layout->addWidget(label5);
```

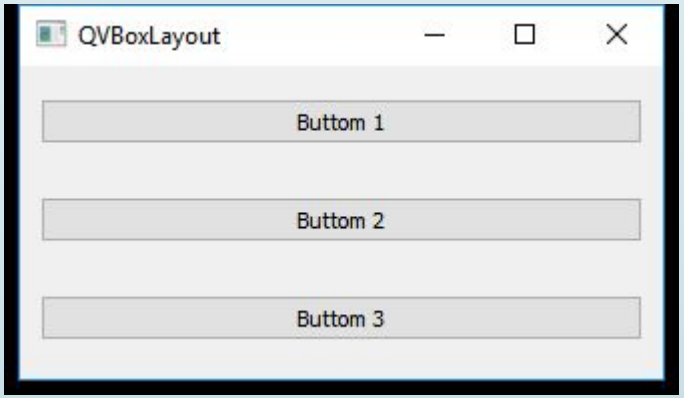
QHBoxLayout

```
QPushButton* buttom1 = new QPushButton("Buttom 1");
QPushButton* buttom2 = new QPushButton("Buttom 2");
QPushButton* buttom3 = new QPushButton("Buttom 3");

QHBoxLayout* hbox_layout = new QHBoxLayout;
hbox_layout->setMargin(10);
hbox_layout->setSpacing(20);
hbox_layout->addWidget(buttom1);
hbox_layout->addWidget(buttom2);
hbox_layout->addWidget(buttom3);
```



QVBoxLayout

```
QPushButton* buttom1 = new QPushButton("Buttom 1");
QPushButton* buttom2 = new QPushButton("Buttom 2");
QPushButton* buttom3 = new QPushButton("Buttom 3");

QVBoxLayout* hbox_layout = new QVBoxLayout;
hbox_layout->setMargin(10);
hbox_layout->setSpacing(20);
hbox_layout->addWidget(buttom1);
hbox_layout->addWidget(buttom2);
hbox_layout->addWidget(buttom3);
```



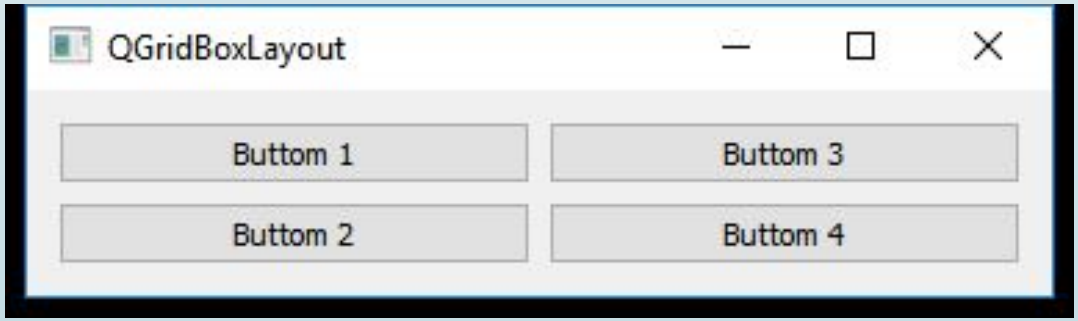
QGridLayout

```
QPushButton* buttom1 = new QPushButton("Buttom 1");
QPushButton* buttom2 = new QPushButton("Buttom 2");
QPushButton* buttom3 = new QPushButton("Buttom 3");
QPushButton* buttom4 = new QPushButton("Buttom 4");

QGridLayout* grid_layout = new QGridLayout;
```

Номер строки

Номер столбца

```
grid_layout->addWidget(buttom1, 0, 0);
grid_layout->addWidget(buttom2, 1, 0);
grid_layout->addWidget(buttom3, 0, 1);
grid_layout->addWidget(buttom4, 1, 1);
```



Слоты и сигналы

Средства, позволяющие эффективно производить обмен информацией о событиях между объектами.

- класс унаследованный от **`QObject`** может иметь сколько угодно слотов и сигналов
- сообщения, передаваемые через сигналы, могут иметь сколько угодно аргументов любого типа
- сигнал может соединяться с различным количеством слотов
- слот может принимать множество сигналов от множества объектов
- при уничтожении объекта все связи слот-сигнал этого объекта уничтожаются

Сигнал

- Сигнал - метод, осуществляющий пересылку сообщений.
- Сигналы определяются в классе как методы, только без реализации.
- Сигнал не обязательно соединять со слотом.
- Существуют готовые сигналы, также можно реализовывать свои.

Слот

- Метод, присоединяющийся к сигналу.
- В слотах нельзя использовать значения по умолчанию.
- Нельзя определять их как static.

Соединение

```
QObject::connect(
const typename QtPrivate::FunctionPointer<Func1>::Object *sender,
Func1 signal,
const QObject *context,
Func2 slot,
Qt::ConnectionType type = Qt::AutoConnection)
```

- **sender** - объект, отправляющий сигнал
- **signal** - сигнал, с которым устанавливается соединение
- **context** - указатель на объект, имеющий слот для обработки сигнала
- **slot** - функция, вызываемая при получении сигнала
- **type** - режим обработки