Материал предыдущего семестра

- Базовые типы данных: целые числа, вещественные числа, строки.
- Создание и использование переменных.
- Условные переходы.
- Циклы.
- Функции.
- Объектно-ориентированное программирование: классы, объекты, поля, методы. Наследование.
- Программные модули.
- Предпочитаемая среда программирования.

Графический интерфейс

Графический интерфейс пользователя (ГИП, англ. GUI, Graphic User Interface) — современный способ взаимодействия прикладного ПО и пользователя, обеспечивающий интуитивную и понятную работу с приложением.

Простота работы пользователя требует дополнительной работы со стороны программиста.

- UI — User Interface
- UX — User Experience

Графический интерфейс

Окно — прямоугольная область экрана, выделенная приложению для отображения интерфейса. Окно обслуживается и оформляется операционной системой. Окно может занимать весь экран.

Виджет — элемент интерфейса, имеющий визуальное представление и предназначенный для взаимодействия с пользователем. Из отдельных виджетов собирается интерфейс приложения.

Устройства ввода: мышь, клавиатура, сенсорный экран, перо, джойстик.

Буфер обмена (в том числе drag&drop). Системные события.

Графический интерфейс

Каждая графическая операционная система (ОС) предлагает свой вариант реализации графического интерфейса, зачастую несовместимого с другими ОС. Также существуют программные библиотеки, работающие поверх интерфейсов ОС, реализующие типовые элементы интерфейса: кнопки, надписи и т. д.

Существуют библиотеки, реализующие построение интерфейса для разных операционных систем: Qt, GTK, WxWidgets и т. д.

В пределах такой библиотеки каждому элементу интерфейса соответствует программная сущность. Например, класс.

Дополнительно обеспечивается связь между виджетами, позволяющая реализовать бизнес-логику приложения.

Библиотека Qt

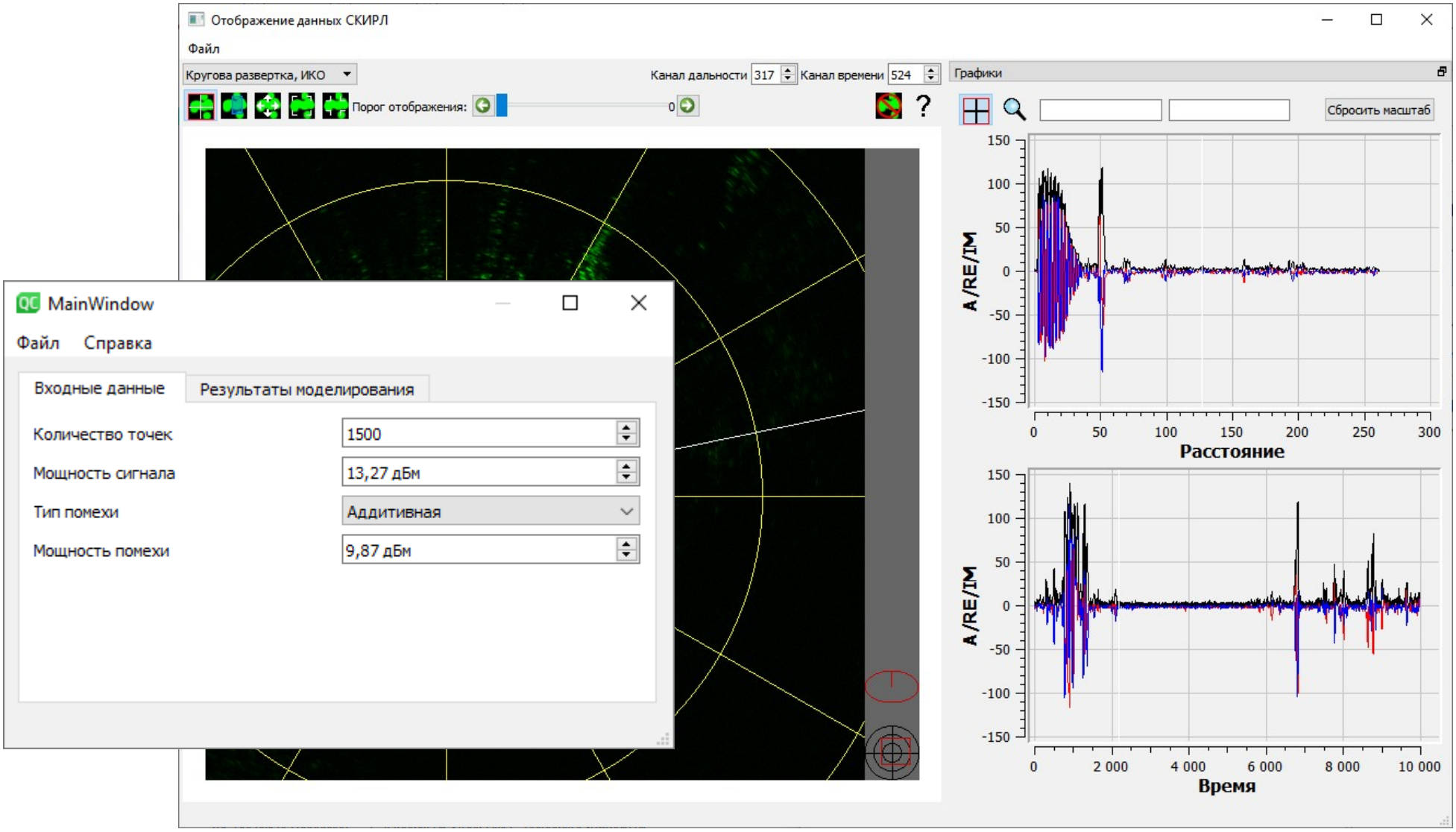
Qt — это кросс-платформенная библиотека, разработанная на языке C++, реализующая задачи построения приложений с графическим интерфейсом пользователя.

Qt предоставляет набор классов, реализующих различные элементы интерфейса, либо выполняющих внутренние служебные функции.

Библиотека предоставляется вместе со средой разработки Qt Creator и доступна на сайте http://qt.io.

Существует порт Qt для Python: PyQt. В PyQt доступны все классы Qt на языке Python с незначительными отличиями от оригинала.

Библиотека Qt



Объекты Qt

Поля объектов Qt закрыты от внешней модификации (private). Модификация объектов идет при помощи методов.

QObject — базовый класс для всех объектов Qt. Обеспечивает идентификацию виджетов, их взаимосвязь.

- **objectName()** — программно изменяемое имя объекта
- **parent()** — родительский объект, при удалении родительского объекта автоматически удаляются все дочерние
- **children()** — список дочерних объектов, автоматически удаляемых при удалении текущего

Виджеты Qt

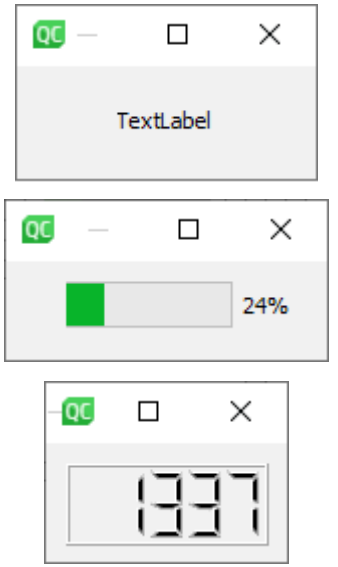
QWidget — базовый класс для всех виджетов. Работает в экранных координатах: ось X слева направо, ось Y сверху вниз. Базовая точка виджета в левом верхнем углу.

Класс QWidget содержит базовые методы для отображаемых объектов:

- **x()**, **y()**, **width()**, **height()** - параметры геометрии
- **move()**, **resize()** - изменение геометрии
- **setVisible()**, **visible()**, задает и узнает видимость
- **setEnabled()**, **enabled()** - переключает и узнает активность; неактивный виджет отображается серым и не реагирует на действия пользователя
- **focus()**, **setFocus()**, **clearFocus()** - работа с фокусом виджета; виджет в фокусе получает события клавиатуры

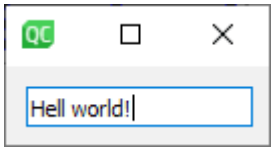
Виджеты вывода данных

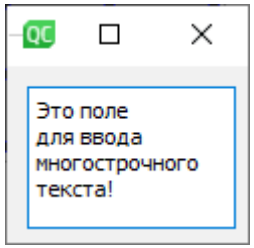
- **QLabel** — надпись в окне приложения;
- **QProgressBar** — заполняемая шкала прогресса;
- **QLCDNumber** — отображение числа, стилизованное под ЖК экран

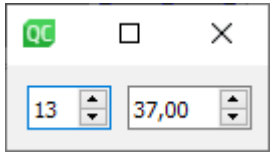


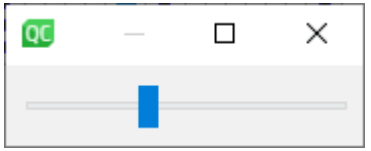
Виджеты ввода данных

- **QLineEdit** — однострочное поле ввода;
- **QTextEdit** — многострочное поле ввода;
- **QSpinBox**, **QDoubleSpinBox** — ввод вещественных и целых чисел
- **QHorizontalScrollBar**, **QVerticalScrollBar** — полоса прокрутки
- **QHorizontalSlider**, **QVerticalSlider** - движок



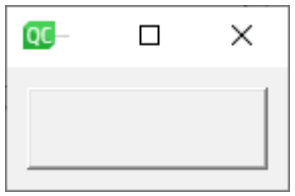


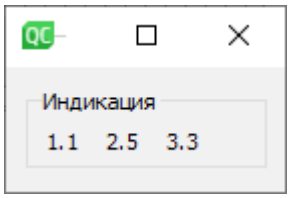


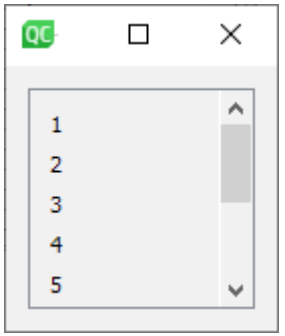


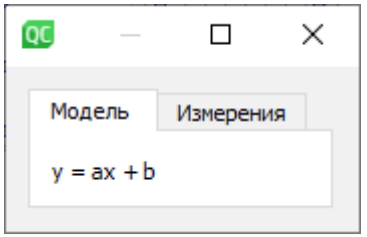
Виджеты-контейнеры

- **QWidget** — невидимая прямоугольная область без оформления;
- **QFrame** — область с оформлением границы;
- **QGroupBox** — область с оформленной границей и подписью;
- **QScrollArea** — область прокрутки;
- **QTabWidget** — вкладки.



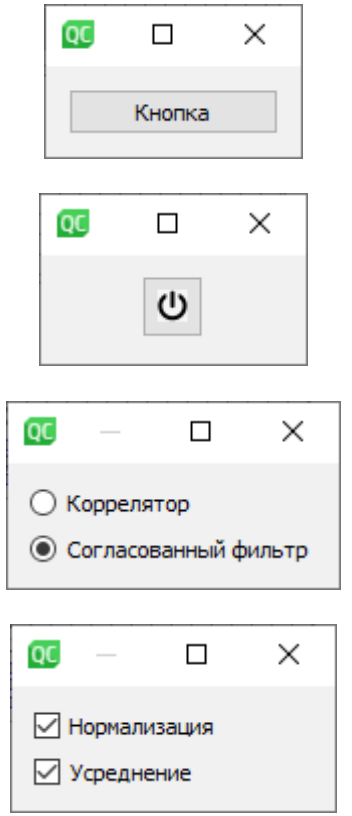






Кнопки

- **QPushButton** — обычная кнопка;
- **QToolButton** — квадратная кнопка с иконкой;
- **QRadioButton** — кнопка для взаимоисключающего выбора;
- **QCheckBox** — кнопка для выбора («галочка»).



Сложные виджеты

- **QListWidget** — вывод элементов в виде списка. Элемент списка — объект класса **QListWidgetItem**.
- **QTableWidget** — вывод элементов в виде таблицы, снабженной горизонтальным и вертикальным заголовками. Элементы таблицы и заголовков — объекты класса **QTableWidgetItem**.
- **QTreeWidget** — вывод элементов в виде дерева. Элемент дерева — объект класса **QTreeWidgetItem**.
- Аналогичные классы **QListView**, **QTableView**, **QTreeView** работают поверх модели данных. Например, QTreeView может работать поверх **QFileSystemModel** для отображения дерева файлов на диске.

Вспомогательные классы

- **QString** — хранит строку в кодировке UTF-16. Основной класс для работы со строками в Qt. Элемент строки QString — класс **QChar**.
- **QByteArray** — байтовый массив. Элемент массива — **char**.
- **QList**, **QVector**, **QQueue**, **QStack**, **QMap**, **QHash** — шаблонные классы-контейнеры.
- **QRect**, **QSize**, **QPoint** — классы для работы с геометрией в целых числах.
- **QRectF**, **QSizeF**, **QPointF** — классы для работы с геометрией в вещественных числах.

Вспомогательные классы

- **QIODevice** — базовый класс для систем ввода-вывода данных:
  - **QFile** — файловый ввод-вывод;
  - **QTcpSocket**, **QUdpSocket** — сетевое соединение по протоколам TCP и UDP соответственно;
  - **QSerialPort** — последовательный порт.
- **QDir** — работа с файловой системой.
- **QImage**, **QPixmap** — работа с растровыми изображениями.
- **QColor**, **QBrush**, **QPen**, **QPainter** — рисование на поверхности виджета или на изображении в памяти.
- **QPicture** — векторное изображение.

Вспомогательные классы

- **QThread** — программный поток выполнения.
- **QMutex** — мьютекс для синхронизации потоков
- **QProcess** — отдельный системный процесс, запускаемый приложением. Унаследован от QIODevice.
- **QSqlDatabase**, **QSqlQuery** — работа с базой данных SQL (MySQL, MSSQL, SQLite и т.д.)

Раскладка виджетов

Поддержание внешнего вида приложения независимо от размеров окна — рутинная, но необходимая задача.

Qt предлагает механизм раскладок (Layout), позволяющий ее решить.

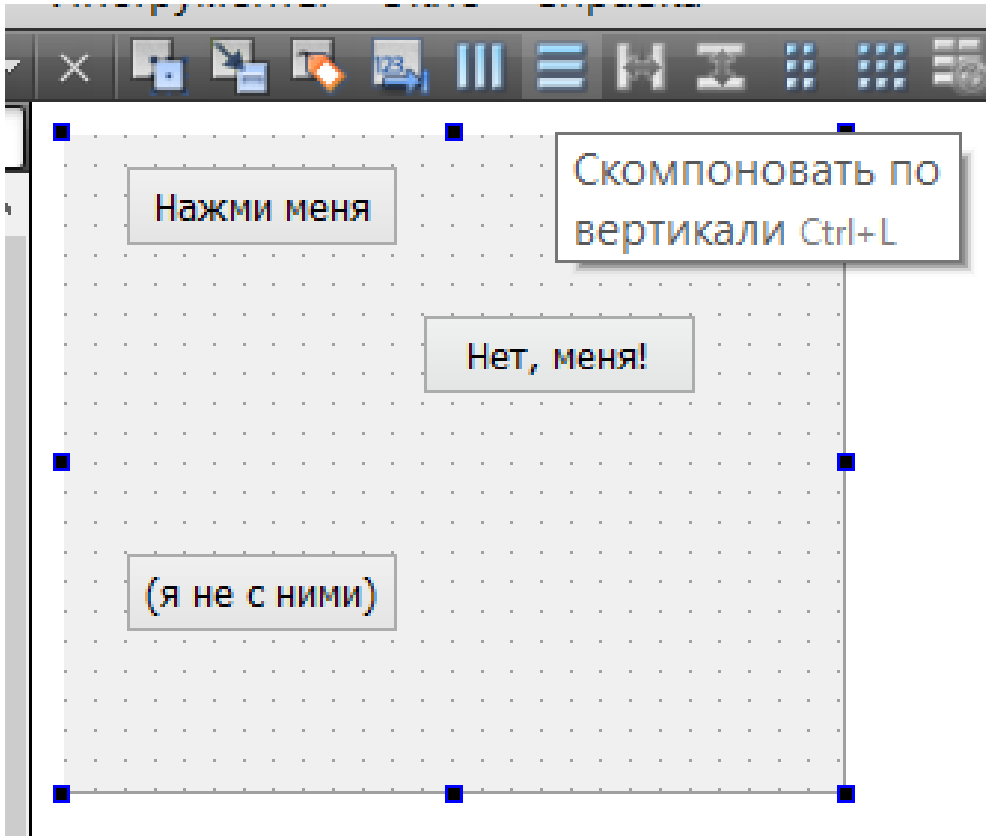
Раскладка — это специальный служебный объект, который прикрепляется к виджету и расставляет его дочерние виджеты в соответствии с правилами раскладки. При этом раскладка учитывает минимальные, максимальные и предпочитаемые размеры дочерних виджетов.

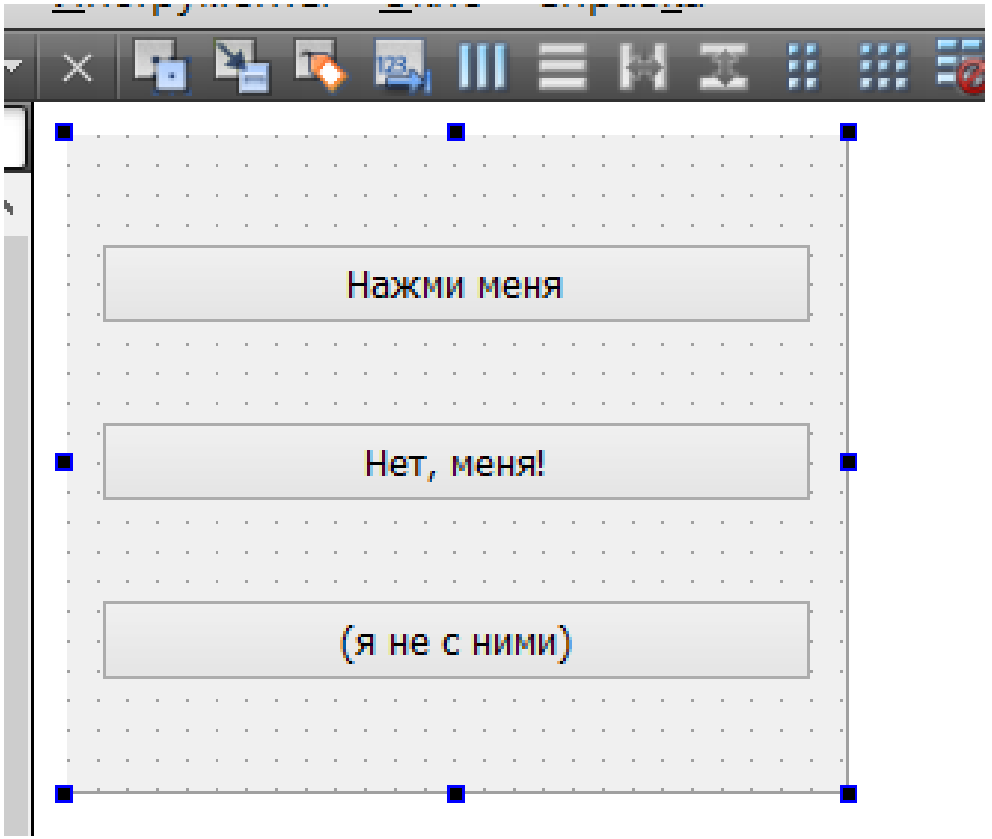
Например, раскладка "по горизонтали" расположит все дочерние виджеты в один ряд так, чтобы они занимали всю площадь родительского виджета.

В Qt доступны раскладки "по горизонтали", "по вертикали" и "по сетке".

Допускается помещать одну раскладку внутрь другой.

Раскладка виджетов





Сигналы и слоты

Сигнал — это программная сущность, порождаемая объектом Qt при возникновении определенного события.

Например, кнопка QPushButton порождает сигнал clicked() при нажатии мышкой.

Сигнал оформляется как объявление функции в секции **signals**. Аргументы функции будут переданы вместе с сигналом.

```
class Keeper : public QObject
{
    Q_OBJECT
signals:
    void valueChanged(int v)
public:
    int _value;
}
```

Сигналы и слоты

Слот — это метод объекта Qt, который может быть соединен с сигналом. После соединения метод будет вызываться автоматически при возникновении сигнала.

Слот объявляется как метод в секции slots:

```
class Catcher : public QObject
{
    Q_OBJECT
public slots:
    void updateValue(int v)
    {
        _value = v;
    }
private:
    int _value;
}
```

Сигналы и слоты

Для соединения сигнала и слота необходимо вызвать метод connect:

```
int main(int argc, char** argv)
{
    Keeper k;
    Catcher c;
    QObject::connect(&k, SIGNAL(valueChanged(int)),
                     &c, SLOT(updateValue(int)));
    return 0;
}
```

При разработке графических приложений используются автослоты:

```
Class MainWindow : public QMainWindow
{
private slots:
    void on_pushButton_clicked();
}
```

Структура проекта Qt

Qt использует расширения языка, не входящие в стандарт, и поэтому применяет специальную двухэтапную систему сборки. На первом этапе специальные компиляторы Qt (MOC, RCC, UIC) преобразуют конструкции Qt к конструкциям C++. Далее приложение собирается обычным компилятором.

Для описания проекта используется язык QMake. Для сборки — одноименная система сборки, вызывающая компиляторы в нужном порядке.

Конечное приложение требует наличия библиотек Qt для запуска. Типовой объем собранного приложения с библиотеками — порядка 50 МБ.

Структура проекта Qt

Типовой проект графического приложения Qt на языке C++ содержит следующие файлы:

- **project.pro** — описание проекта на языке QMake.
- **mainwindow.ui** — описание интерфейса приложения, созданное в Qt Designer
- **mainwindow.h** — заголовочный файл основного класса программы
- **mainwindow.cpp** — реализация методов основного класса
- **main.cpp** — автоматически сгенерированный файл, осуществляющий запуск приложения.

Среда может создавать файл project.pro.user в процессе работы. Он не является частью проекта и содержит служебную информацию с конкретного компьютера.

Файл QMake

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    mainwindow.h

FORMS += \
    mainwindow.ui
```

Структура класса MainWindow

Класс MainWindow содержит в себе объект **ui**. В данном объекте содержатся все виджеты, добавленные в Qt Designer и записанные в файле mainwindow.ui. Эти объекты доступны внутри любого метода класса.

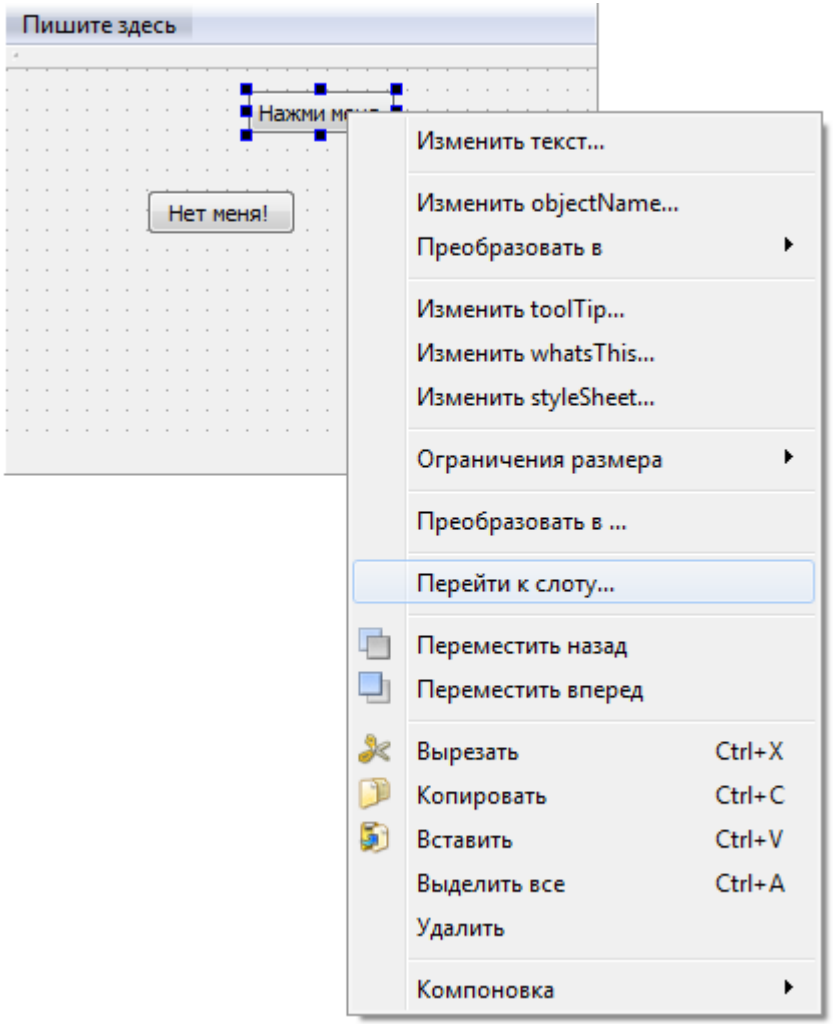
```
class MainWindow : public QMainWindow
{
    Q_OBJECT

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

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

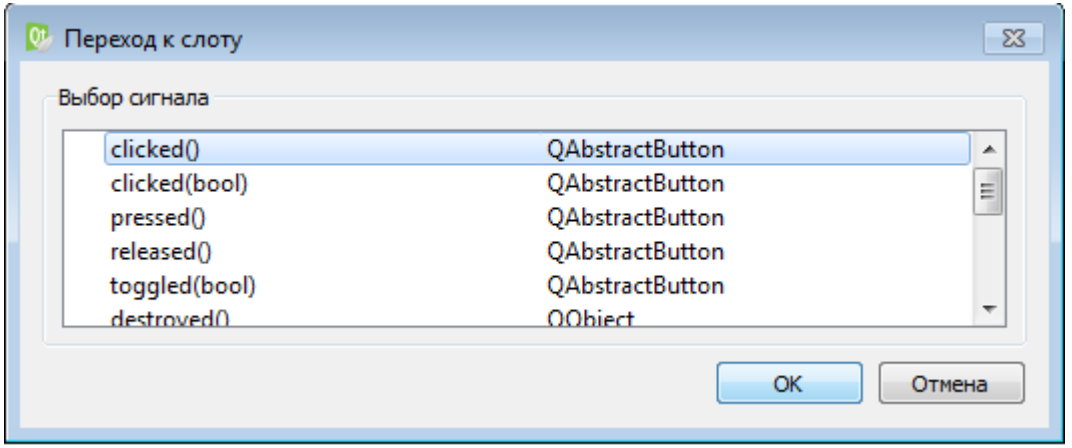
Структура класса MainWindow

Класс MainWindow содержит в себе объект **ui**. В данном объекте содержатся все виджеты, добавленные в Qt Designer и записанные в файле mainwindow.ui. Эти объекты доступны внутри любого метода класса.

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->pushButton->setText("Нажми меня!");
}
```

Создание слота в Qt Creator





Создание слота в Qt Creator

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

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

Private slots:
    void on_pushButton_clicked();

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

```
Void mainWindow::on_pushButton_clicked()
{
    ui->lineEdit->setText("Clicked!");
}
```

Qt в Python: PyQt

PyQt — это порт Qt для использования его в языке Python.

Для установки PyQt необходимо использовать утилиту pip в командной строке (могут потребоваться права администратора):

```
c:\users\user> cd c:\python38-32\scripts
c:\python38-32\scripts> pip install pyqt5 pyqt5-tools
```

PyQt содержит все классы Qt за исключением некоторых отличий:

- Нельзя использовать Qt Creator, но доступна любая среда разработки Python и программа Qt Designer.
- Вместо QString напрямую используются строки Python.
- Вместо слотов используются обычные функции и методы.

Проект PyQt

Проект с использованием PyQt сотоит из следующих файлов:

- **mainwindow.ui** — интерфейс окна приложения, разработанный с использованием Qt Designer.
- **main.py** — сценарий проекта

```
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5 import uic

app = QApplication([])
ui = uic.loadUi("PyQtForm.ui")
ui.show()
exit(app.exec())
```

Объект ui содержит в себе все объекты, созданные в Qt Designer.

Сигналы и слоты в PyQt

Сигнал в PyQt выполнен в виде специального объекта. Объект сигнала содержит метод **connect**, позволяющий подключить к сигналу любой вызываемый метод или функцию.

```
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5 import uic

def onClick():
    ui.lineEdit.setText("Clicked!")

app = QApplication([])
ui = uic.loadUi("PyQtForm.ui")
ui.show()
ui.pushButton.clicked.connect(onClick())
exit(app.exec())
```

Краткое руководство по классам

Методы **x()**, **y()**, **width()** и **height()** возвращают текущие геометрические параметры виджета: координаты левого верхнего угла, ширину и высоту соответственно.

Метод **resize(int, int)** позволяет задать размеры виджета в пикселях.

Метод **move(int, int)** позволяет переместить виджет в указанные координаты.

Координаты виджета определяются внутри родительского виджета. Для всего окна координаты определяются относительно левого верхнего угла.

Метод **hide()** скрывает виджет.

Метод **show()** показывает виджет на экране, если он скрыт.

Метод **isVisible()** возвращает, отображается ли виджет.

Краткое руководство по классам

Метод **setEnabled(bool)** переключает активность виджета. Неактивный виджет отображается серым и не реагирует на действия пользователя, однако с ним можно взаимодействовать программно.

Метод **isEnabled()** возвращает, активен ли виджет.

Метод **setText(QString)** задает текст для отображения на виджетах, работающих с текстом. Например, QLabel, QLineEdit.

Метод **text()** возвращает строку, отображаемую на виджете, работающем с текстом.

Метод **setValue(…)** задает значение для виджета, работающего с числом (QSpinBox, QSlider). Тип значения зависит от виджета: int для целых чисел, double для вещественных.

Метод **value()** возвращает значение виджета, работающего с числом.

Краткое руководство по классам

Метод **setChecked(bool)** позволяет установить или снять отметку для виджетов выбора (QCheckBox, QRadioButton) или переключить "залипание" кнопки (QPushButton, QToolButton).

Метод **isChecked()** возвращает текущее состояние отметки или залипание кнопки.

Краткое руководство по классам

Класс **QString** используется в C++ для работы со строками в Qt.

При работе со строками необходимо знать кодировку файлов исходных кодов. Рекомендуется применять кодировку UTF-8 для всех файлов.

Для создания строки необходимо выполнить:

```
QString s("Hello");
QString ru = QString::fromUtf8("Привет!");
```

Строки поддерживают присвоение и арифметические операции:

```
QString s2 = s + " " + ru;
QStrig s3 = s;
s3 += " world!";
```

Краткое руководство по классам

Преобразование между числами и строками делается средствами QString. Для создания строки из числа необходимо использовать метод **QString::number**:

```
int a = 143;
double b = 225.76;
QString s = QString::number(a);
QString bb = QString::number(b);
```

Для создания числа из строки существуют методы **toInt()** и **toDouble()**:

```
QString v1("2312");
QString v2("3.1415926")
int a = v1.toInt();
double b = v2.toDouble();
```