

# Qt

## Введение

Кафедра ИВТ и ПМ

2019

# Outline

## Трудности создания программ с GUI

### Qt

Структура проекта

Классы Qt

QObject

Подходы к созданию приложений с GUI в Qt

### Вопросы

- ▶ В стандартную библиотеку C++ не входят средства для удобного и быстрого создания приложений с GUI<sup>1</sup>
- ▶ Создавать окна и элементы интерфейсы можно пользуясь встроенными API операционной системы<sup>2</sup>. Например WinAPI

---

<sup>1</sup>graphical user interface, GUI - Графический интерфейс пользователя (ГИП)

<sup>2</sup>API (программный интерфейс приложения) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах.

# Создание окна

```
#include <windows.h>

const char g_szClassName[] = "myWindowClass";

// Step 4: the Window Procedure
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch(msg)
    {
        case WM_CLOSE:
            DestroyWindow(hwnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;

    //Step 1: Registering the Window Class
    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = 0;
    wc.lpfnWndProc = WndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = g_szClassName;
    wc.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
```

```
if(!RegisterClassEx(&wc))
{
    MessageBox(NULL, "Window Registration Failed!", "Error!",
        MB_ICONEXCLAMATION | MB_OK);
    return 0;
}

// Step 2: Creating the Window
hwnd = CreateWindowEx(
    WS_EX_CLIENTEDGE,
    g_szClassName,
    "The title of my window",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
    NULL, NULL, hInstance, NULL);

if(hwnd == NULL)
{
    MessageBox(NULL, "Window Creation Failed!", "Error!",
        MB_ICONEXCLAMATION | MB_OK);
    return 0;
}

ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

// Step 3: The Message Loop
while(GetMessage(&Msg, NULL, 0, 0) > 0)
{
    TranslateMessage(&Msg);
    DispatchMessage(&Msg);
}
return Msg.wParam;
}
```

Создание окна с помощью WinAPI

# Проблема

- ▶ Использование API операционной системы для создания GUI - это громоздкий код, даже для создания одного пустого окна  
Такой подход замедляет разработку
- ▶ Для создания приложения желательно использовать готовые шаблоны
- ▶ C++ компилируется для разных ОС.
- ▶ Но код для создания приложений с GUI на разных ОС отличается (если использовать средства ОС напрямую). Отсюда платформозависимость.
- ▶ Решение - использование библиотек с готовыми элементами интерфейса (приспособленных для нескольких ОС)

# Решение

Для создания приложений с GUI используются сторонние *фреймворки*, не входящие в стандартную библиотеку C++.

Для Windows

- ▶ Windows Presentation Foundation (WPF)<sup>3</sup>

Кроссплатформенные

- ▶ **Qt**
- ▶ GTK+
- ▶ wxWidgets

---

<sup>3</sup>входит в состав .NET Framework

**Фреймворк** (framework — остов, каркас, структура) — программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

- ▶ Как правило фреймворк состоит из классов, функций.

# фреймворк и библиотека

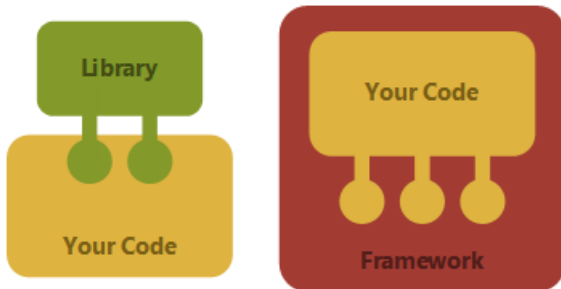
- ▶ Библиотека определяет как пользовательский<sup>4</sup> код будет с ней взаимодействовать, архитектуру определяет программист.
- ▶ фреймворк определяет архитектуру программы.
- ▶ Можно говорить, что фреймворк определяет структуру программы
- ▶ Функции библиотеки вызываются пользовательским кодом.
- ▶ Фреймворк вызывает пользовательский код.
- ▶ фреймворк может содержать в себе множество библиотек различного назначения.

---

<sup>4</sup>под пользователем понимается программист использующий фреймворк



# Фреймворк и библиотека



# Фреймворк для создания программ с GUI

- ▶ Как правило при работе с фреймворком (для создания приложений с GUI<sup>5</sup>) программисту предоставляется один основной класс представляющий главное окно программы, *наследника* от которого нужно реализовать.
- ▶ Бизнес логика приложения как правило реализуется с помощью агрегирования пользовательских классов, сторонних библиотек, других классов фреймворка, а также реализации методов основного класса.
- ▶ Главные методы основного класса генерируются автоматически средствами IDE (при создании проекта по шаблону), а агрегирование классов представляющих собой элементы интерфейса - с помощью дизайнера форм.

---

<sup>5</sup>фреймворки могут использоваться и для решения других задач - например создания веб-приложений

# Пример использования фреймворка Qt

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
```

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

```
// Создаётся новый класс для главного окна на основе существующего
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
```

```
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    // добавляются или переопределяются методы
```

```
private:
    Ui::MainWindow *ui;
    // добавляются поля класса
};
```

```
#endif // MAINWINDOW_H
```

# Пример использования фреймворка Qt

mainwindow.cpp

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

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

```
#include <random>
```

```
MainWindow::MainWindow(QWidget *parent) :
```

```
    QMainWindow(parent), ui(new Ui::MainWindow){
```

```
    ui->setupUi(this);
```

```
    // Объект ui содержит все классы-элементы интерфейса,
```

```
    // расположенные на главном окне.
```

```
    // класс для ui генерируется автоматически.
```

```
}
```

```
void MainWindow::on_pushButton_clicked(){
```

```
    // Пользовательский код
```

```
    ui->label_num->setText( QString::number(rand()) );
```

```
}
```

# Пример использования фреймворка Qt

main.cpp

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

    return a.exec();
}
```

# Событийно-ориентированное программирование

- ▶ В приложениях с GUI пользователь в большей степени определяет порядок выполнения программы чем в консольных программах.
- ▶ Здесь приложение реагирует на действия пользователя, а не только пользователь на действия программы
- ▶ Подход создания программ, при котором её выполнение определяется событиями (действиями пользователя, операционной системы, сетью и т.д) называется **событийно-ориентированным программированием**<sup>6</sup>.
- ▶ Пользователь такой программы фактически вызывает методы класса описывающего главное окно нажимая на кнопки, вводя данные, работая с элементами интерфейса.

---

<sup>6</sup>такое же подход использовался и на слайде 4

# Outline

Трудности создания программ с GUI

Qt

- Структура проекта

- Классы Qt

  - QObject

- Подходы к созданию приложений с GUI в Qt

Вопросы

# Приложения построенные на основе QT



Bitcoin Core



Google Earth



KeePass



Mathematica



GnuOctave



KStars



LibreCAD

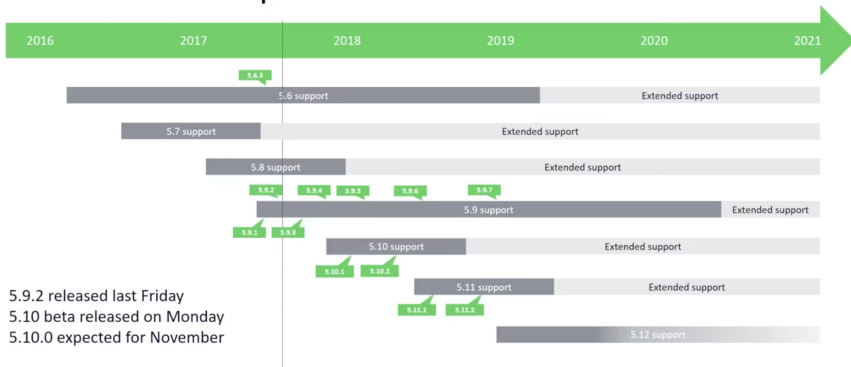


MARBLE

<https://resources.qt.io/customer-stories-all>



## Release roadmap



# Документация QT

- ▶ Документация есть в Qt Creator
- ▶ <http://doc.qt.io>
- ▶ Qt 5.X. Профессиональное программирование на C++. Макс Шлее. 2015 г. 928 с.
- ▶ Документация на русском доступно только для 4-й версии<sup>7</sup>  
[doc.crossplatform.ru/qt](http://doc.crossplatform.ru/qt)
- ▶ См. также *Примеры* и *Учебники* на вкладке *Начало* в QtCreator


---

<sup>7</sup>последняя на данный момент (2019 год.) версия фреймвока - 5.  
Версии 4 и 5 во многом похожи

# Особенности Qt

- ▶ простой API
- ▶ Поддержка разных платформ: Windows, Linux, MacOS, Android и др.
- ▶ Средства для работы с сетью, БД, потоками, файловой системой и т.п.
- ▶ STL-совместимая библиотека контейнеров
- ▶ Система сигналов и слотов<sup>8</sup>
- ▶ Возможно использование декларативного языка (QML) и JavaScript для создания интерфейсов пользователя
- ▶ Встроенная в IDE Qt Creator справка и примеры приложений
- ▶ Модули для работы с языками C#, Python, PHP и др.

---

<sup>8</sup>Сигналы и слоты реализованы также в библиотеке boost. 

# Особенности Qt

## Сигналы и слоты

- ▶ Qt расширяет возможности C++
- ▶ Классы построенные с использованием Qt могут отправлять друг другу *сигналы*.
- ▶ Причём, обработка механизма обмена сообщениями полностью лежит на Qt
- ▶ Для отправки *сообщения* класс использует специальный метод - сигнал.
- ▶ При вызове сигнала будет вызван назначенный ему слот - метод другого объекта.

# Outline

Трудности создания программ с GUI

Qt

Структура проекта

Классы Qt

QObject

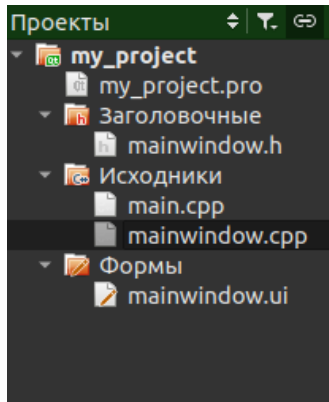
Подходы к созданию приложений с GUI в Qt

Вопросы

# Структура проекта

На примере шаблона на основе Qt Widgets

- ▶ pro файл - файл проекта.  
Содержит:
  - ▶ имена файлов проекта
  - ▶ имена используемых компонентов (например Qt)
  - ▶ параметры проекта
- ▶ Файлы исходных кодов и заголовочные файлы
- ▶ Файлы форм (\*.ui)



# Пример файла проекта для приложения с GUI

```
QT += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = example_gui
TEMPLATE = app

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    mainwindow.h

FORMS += \
    mainwindow.ui
```

## \*.pro - файл проекта

Для настройки параметров проекта нужно задать значения переменным проекта:

- ▶ **CONFIG** - общие настройки проекта и компилятора
- ▶ **QT** - список модулей Qt используемых проектом<sup>9</sup>
- ▶ **FORMS** - список форм, которые должны быть обработаны user interface compiler (uic).
- ▶ **HEADERS** - список заголовочных файлов
- ▶ **SOURCES** - список файлов исходных кодов
- ▶ **TEMPLATE** - шаблон приложения (application, library, plugin)
- ▶ **TARGET** - имя проекта (по умолчанию включает имя заданное в мастере создания проекта)

Как правило настройка проекта производится добавлением (оператор +=) в переменную требуемых значений.

<sup>9</sup>используются скомпилированные модули: заголовочный файл (подключается как обычно) и файл, полученный компиляцией `src`



Некоторые значения для переменной CONFIG:

- ▶ qt - проект использует Qt
- ▶ release, debug - режим сборки (обычно не указывается, а выбирается в IDE)
- ▶ console - построение консольного приложения
- ▶ c++11, c++14 - включение поддержки соответствующих стандартов (может понадобится задать дополнительно: `QMAKE_CXXFLAGS += -std=c++14`)
- ▶ thread - включить поддержку потоков

## Файл проекта

Некоторые значения для переменной QT:

- ▶ core - базовый модуль Qt, необходимый каждому Qt приложению
- ▶ gui - включает базовые средства для создания приложений с GUI
- ▶ widgets - включает элементы интерфейса пользователя на основе QWidget
- ▶ quick - включает элементы интерфейса, построенного на основе QML
- ▶ opengl - поддержка opengl
- ▶ network - поддержка сети
- ▶ xml - поддержка XML

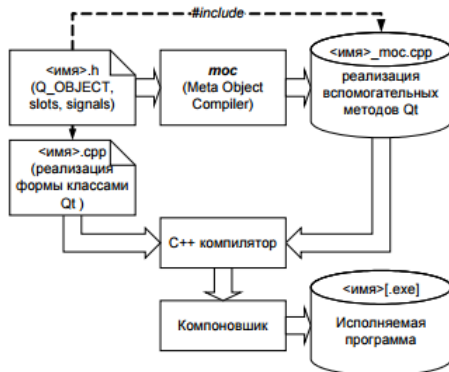
По умолчанию, QT уже включает модули core и gui.

При необходимости модуль можно исключить, например при создании консольного приложения: `QT -= gui`

## Сборка Qt проекта

- ▶ Механизм сигналов и слотов не является частью языка C++
- ▶ поэтому исходные коды сначала транслируются в чистый C++, а затем уже компилируются.
- ▶ Транслированием занимается отдельная программа - moc - meta object compiller
- ▶ На выходе moc получается C++ код, который можно компилировать отдельными компиляторами: gcc (MinGW), MSVC
- ▶ Заголовочные файлы, с классами использующими метаобъектную систему Qt должны включать макрос Q\_OBJECT, чтобы moc транслировал их в чистый C++

# Сборка Qt проекта



## ui файл

- ▶ UI файл описывает графический интерфейс пользователя.
- ▶ Для каждого окна используется отдельный UI файл
- ▶ Это XML файл, в котором содержатся названия и свойства всех элементов интерфейса (например размеры главного окна и его заголовков)
- ▶ Для изменения этого файла используется Дизайнер в QtCreator
- ▶ Во время сборки проекта содержимое преобразуется в класс на C++ (файл `ui_mainwindow.h` в каталоге сборки)
- ▶ Полученный класс агрегирует указатели на элементы интерфейса (которые являются готовыми классами Qt).

# Пример UI файла

## Исходный XML код

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>MainWindow</class>
  <widget class="QMainWindow" name="MainWindow">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>400</width>
        <height>300</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>MainWindow</string>
    </property>
    <widget class="QWidget" name="centralWidget"/>
    <widget class="QMenuBar" name="menuBar">
      <property name="geometry">
        <rect>
          <x>0</x>
          <y>0</y>
          <width>400</width>
          <height>22</height>
        </rect>
      </property>
    </widget>
    <widget class="QToolBar" name="mainToolBar">
      <attribute name="toolBarArea">
        <enum>TopToolBarArea</enum>
      </attribute>
      <attribute name="toolBarBreak">
        <bool>false</bool>
      </attribute>
    </widget>
    <widget class="QStatusBar" name="statusBar"/>
  </widget>
  <layoutdefault spacing="6" margin="11"/>
  <resources/>
  <connections/>
</ui>
```

# Пример UI файла

## Код полученный из UI файла

```
/*
** Form generated from reading UI file 'mainwindow.ui'
** Created by: Qt User Interface Compiler version 5.11.0
** WARNING! All changes made in this file will be lost when
** recompiling UI file!
*/

#ifdef UI_MAINWINDOW_H
#define UI_MAINWINDOW_H
#include <QtCore/QVariant>
#include <QtWidgets/QApplication>
#include <QtWidgets/QMainWindow>
#include <QtWidgets/QMenuBar>
#include <QtWidgets/QStatusBar>
#include <QtWidgets/QToolBar>
#include <QtWidgets/QWidget>

QT_BEGIN_NAMESPACE

class Ui_MainWindow
{
public:
    QMenuBar *menuBar;
    QToolBar *mainToolBar;
    QWidget *centralWidget;
    QStatusBar *statusBar;

    void setupUi(QMainWindow *MainWindow)
    {
        if (MainWindow->objectName().isEmpty())
            MainWindow->setObjectName(QStringLiteral("MainWindow"));
        MainWindow->resize(400, 300);
        menuBar = new QMenuBar(MainWindow);

        mainToolBar = new QToolBar(MainWindow);
        mainToolBar->setObjectName(QStringLiteral("mainToolBar"));
        MainWindow->addToolBar(mainToolBar);
        centralWidget = new QWidget(MainWindow);
        centralWidget->setObjectName(QStringLiteral("centralWidget"));
        MainWindow->setCentralWidget(centralWidget);
        statusBar = new QStatusBar(MainWindow);
        statusBar->setObjectName(QStringLiteral("statusBar"));
        MainWindow->setStatusBar(statusBar);
        retranslateUi(MainWindow);
        QMetaObject::connectSlotsByName(MainWindow);
    } // setupUi

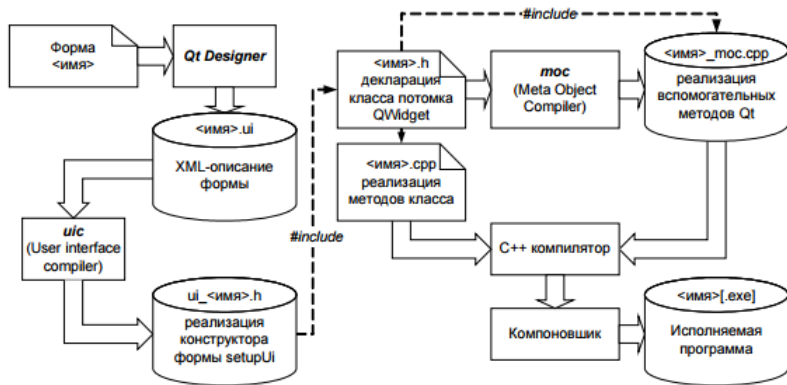
    void retranslateUi(QMainWindow *MainWindow)
    {
        MainWindow->setWindowTitle(QApplication::translate(
    } // retranslateUi
};

namespace Ui {
    class MainWindow: public Ui_MainWindow {};
} // namespace Ui

QT_END_NAMESPACE
#endif // UI_MAINWINDOW_H
```

# Сборка Qt проекта

## Приложения с GUI



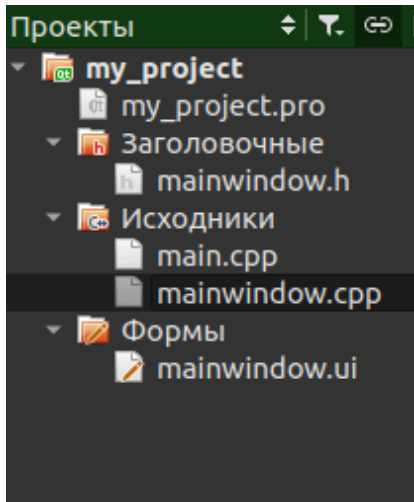


# Сборка Qt проекта

- ▶ Все файлы использующие классы Qt сначала транслируются в код на чистом C++
- ▶ Файл формы (\*.ui) - это обычный XML файл
- ▶ Этот файл тоже транслируется в C++ код
- ▶ На выходе из него получается класс представляющий всё содержимое окна
- ▶ Этот класс агрегируется (поле ui) в класс MainWindow, который описывает уже программист

# Пример использования фреймворка Qt

## Структура проекта



# Пример использования фреймворка Qt

mainwindow.cpp

```
#include "mainwindow.h"

// подключим созданный из UI файла формы код
// Этот файл создаётся в процессе сборки проекта
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent), ui(new Ui::MainWindow){
    ui->setupUi(this);
    // Объект ui содержит все классы-элементы интерфейса,
    // расположенные на главном окне.
    // класс для ui генерируется автоматически.
}

void MainWindow::on_pushButton_clicked(){
    // Пользовательский код
    ui->label_num->setText( QString::number(rand()) );
}
```

# Outline

Трудности создания программ с GUI

Qt

Структура проекта

Классы Qt

QObject

Подходы к созданию приложений с GUI в Qt

Вопросы

# QObject

- ▶ Все классы в Qt - наследники класса QObject.
- ▶ Это виртуальный класс
- ▶ Для его использования нужно подключить модуль Qt
- ▶ Для этого нужно подключить  
QT += core

```
#include <QObject>
```

- ▶ Вместо <QObject> можно использовать другие файлы Qt, так как классы описанные в них, включают QObject
- ▶ QObject поддерживает механизм сигналов и слотов
- ▶ Документация:<http://doc.qt.io/qt-5/qobject.html#details>

# QObject

Любой класс построенный на основе QObject должен содержать макрос Q\_OBJECT.

```
class MtyClass : public QObject{  
    Q_OBJECT  
    ...  
};
```

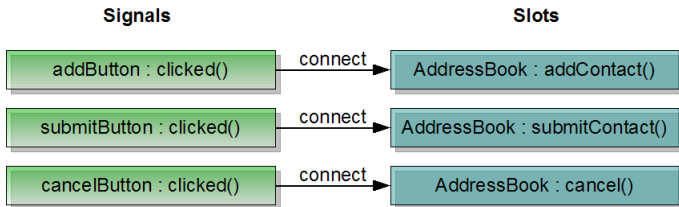
Этот макрос нужен для работы moc.

Если новый класс наследует несколько классов, включая QObject, то QObject должен приводится первым.

# Сигналы и слоты

- ▶ **Сигнал** (signal) - метод вызываемый во время события.
- ▶ **Слот** (slot) - метод принимающий сигнал.

Соединяя сигналы и слоты между собой с помощью специальной функции можно добиться автоматического вызова методов одного объекта (слотов), на вызов методов другого объекта (сигналов)



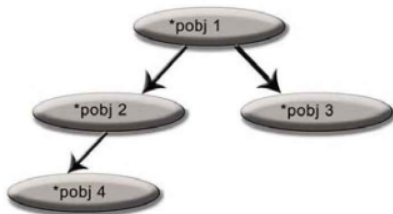
# QObject

- ▶ Содержит метаобъектную информацию (например, объект содержит в себе название класса и информации о наследовании)
- ▶ Механизм объединения объектов в иерархические структуры  
Объекту может быть назначен владелец (параметр `parent`), это избавит от ручного удаления объекта: он будет удалён владельцем, когда тот сам прекратит существование.
- ▶ ...



# Объектная иерархия

```
QObject* pobj1 = new QObject;  
QObject* pobj2 = new QObject(pobj1);  
QObject* pobj4 = new QObject(pobj2);  
QObject* pobj3 = new QObject(pobj1);  
pobj2->setObjectName("the first child of pobj1");  
pobj3->setObjectName("the second child of pobj1");  
pobj4->setObjectName("the first child of pobj2");
```



Чтобы удалить все объекты достаточно удалить только владельца - obj1

Пример из книги Профессиональное программирование на C++. Макс Шлее.  
2015 г

# Объектная иерархия

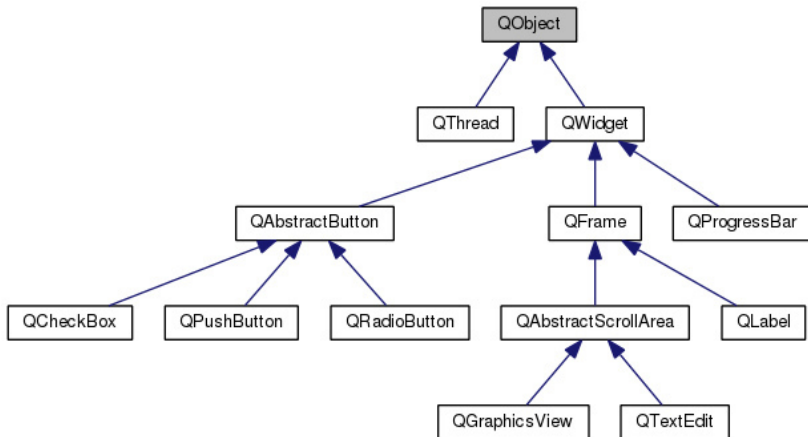
- ▶ Организация объектов в иерархию упрощает динамическое управление памятью
- ▶ Объекты которые управляют удалением других объектов называют родительскими (parent)  
10
- ▶ У классов Qt есть конструктор принимающий указатель на родителя
- ▶ Следует создавать Qt объекты динамически (с использованием оператора new)
- ▶ чтобы не удалять такие объекты вручную назначать им родителя, который сам позаботится об освобождении памяти

---

<sup>10</sup> не стоит путать с наследованием

# Иерархия классов

Фрагмент дерева иерархии классов



# Некоторые классы

- ▶ QApplication - класс взаимодействующий с ОС (обработка событий и т.п.)
- ▶ QWidget - базовый класс для элементов интерфейса (пустое окно)
- ▶ QMainWindow - основное окно программы
- ▶ QLabel - класс "Надпись"
- ▶ QSpinBox - класс "Числовое поле ввода"
- ▶ QPushButton - класс "Кнопка"
- ▶ QTextEdit - класс "Текстовое поле ввода"
- ▶ QTableWidgetItem - класс для представления табличных данных"

# Outline

Трудности создания программ с GUI

Qt

Структура проекта

Классы Qt

QObject

Подходы к созданию приложений с GUI в Qt

Вопросы

# Подходы к созданию приложений с GUI в Qt

- ▶ Создание GUI в редакторе форм, Qt Creator автоматически генерирует соответствующие классы и отношения между ними. Используются ui файлы. Универсальный подход. Пример на слайде 11 и далее)

# Подходы к созданию приложений с GUI в Qt

- ▶ Создание GUI динамически, во время запуска или выполнения программы.

Подходит для небольших программ. Окно конструируется в C++ коде

Пример:

<https://github.com/VetrovSV/OOP/blob/master/SignalsAndSlots2>

- ▶ Использование декларативного языка описания QML и JavaScript.

Гибкий инструмент описания и настройки внешнего вида GUI.

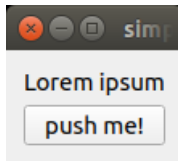
Подробнее рассматривается в *OOP<sub>28U</sub> markup language.pdf*

# Подходы к созданию приложений с GUI в Qt

## Динамическое создание окна

```
#include <QApplication>
#include <QPushButton>
#include <QLabel>
#include <QVBoxLayout>

int main(int argc, char** argv){
    QApplication a(argc, argv);
    QWidget main_widget; // Пустой виджет. Будет главным окном.
    // Кнопка
    QPushButton *button = new QPushButton("push me!", &main_widget);
    // Надпись
    QLabel *label = new QLabel("Lorem ipsum", &main_widget);
    // Вертикальный компоновщик элементов интерфейса
    QVBoxLayout * layout = new QVBoxLayout(&main_widget);
    layout->addWidget(label);
    layout->addWidget(button);
    main_widget.setLayout(layout);
    main_widget.show();
    return a.exec();
```





# Outline

Трудности создания программ с GUI

Qt

Структура проекта

Классы Qt

QObject

Подходы к созданию приложений с GUI в Qt

Вопросы

# Вопросы

- ▶ Что такое фреймворк?
- ▶ Qt - это объектно-ориентированный фреймворк?
- ▶ Как представлено окно приложения в программе (с точки зрения языка программирования)?
- ▶ Что такое событийно-ориентированное программирование?
- ▶ Какой класс является базовым для всех в Qt?
- ▶ Что можно сказать о реализации динамического полиморфизма в Qt?

# Ссылки и литература

- ▶ Qt Википедия
- ▶ OpenSource версия
- ▶ Qt wiki

## Книги:

- ▶ Qt 5.X. Профессиональное программирование на C++. Макс Шлее. 2015 г. 928 с. Книга периодически обновляется с выходом новых версий фреймворка Qt.
- ▶ Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++. Марк Саммерфилд

## Ссылки и литература. Другие фреймворки

- ▶ [itvdn.com/ru/video/wpf](http://itvdn.com/ru/video/wpf) - видеолекция: Введение в WPF и XAML

## Об установке Qt

При установке Qt и работе с IDE нельзя использовать кириллические (и иные кроме английских) пути к папкам и файлам.

Установленный комплект Qt (фреймворк и IDE) занимают 1-4 Гб в зависимости от ОС и выбранного компилятора.

Qt поставляется в виде библиотек (бинарных файлов и файлов исходных кодов) для разных компиляторов.

При установке нужно выбрать версию Qt для желаемого компилятора.

## Об установке Qt

qt.io - сайт Qt

Для скачивания доступно 2 версии: для коммерческого использования (Commercial) и Open Source версия. Вторая - бесплатна, распространяется под (L)GPL v3 лицензией.

По умолчанию доступен онлайн-установщик, но существует и его оффлайн версия [qt.io/offline-installers/](https://qt.io/offline-installers/)

# Об установке Qt

## Выбор компонентов

Выберите компоненты для установки. Для удаления уже установленных компонентов снимите отметки выбора. Уже установленные компоненты не будут

Имя компонента	Установл
▼ Preview	
▶ <input type="checkbox"/> Qt 5.10.1 snapshot	
▶ <input type="checkbox"/> Qt 5.11.0 Alpha snapshot	
▶ <input type="checkbox"/> Qt Creator 4.5.0-rc1	
▼ <input checked="" type="checkbox"/> Qt	1.0.8
▼ <input checked="" type="checkbox"/> Qt 5.10.0	5.10.0-0-
<input checked="" type="checkbox"/> Desktop gcc 64-bit	5.10.0-0-
<input checked="" type="checkbox"/> Android x86	5.10.0-0-
<input checked="" type="checkbox"/> Android ARMv7	5.10.0-0-
<input type="checkbox"/> Sources	
<input type="checkbox"/> Qt Charts	
<input type="checkbox"/> Qt Data Visualization	
<input type="checkbox"/> Qt Purchasing	
<input type="checkbox"/> Qt Virtual Keyboard	
<input type="checkbox"/> Qt WebEngine	
<input type="checkbox"/> Qt Network Authorization	
<input type="checkbox"/> Qt Remote Objects (TP)	
<input type="checkbox"/> Qt WebGL Streaming Plugin (TP)	
<input type="checkbox"/> Qt Script (Deprecated)	
▶ <input type="checkbox"/> Qt 5.9.4	
▶ <input type="checkbox"/> Qt 5.9.3	

### Qt 5.10.0

Этот компонент займёт приблизительно 1.14 ГБ на жестком диске.

# Материалы курса

Слайды, вопросы к экзамену, задания, примеры

[github.com/VetrovSV/OOP](https://github.com/VetrovSV/OOP)