

# Qt

## Введение

Кафедра ИВТ и ПМ

2021

# Outline

Трудности создания программ с GUI

Qt

Структура Qt проекта

Подходы к созданию приложений с GUI в Qt

- Динамическое создание интерфейса пользователя

- Использование QML

- Создание GUI в редакторе форм

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

# Проблемы

- ▶ **Проблема 1:** 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.
- ▶ код для создания приложений с GUI на разных ОС отличается (если использовать средства ОС напрямую).  
Отсюда **проблема 2:** 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 <sup>3</sup>

- ▶ WinAPI
- ▶ WindowsForms
- ▶ Windows Presentation Foundation (WPF)
- ▶ Universal Windows Platform (UWP)
- ▶ WinUI

Кроссплатформенные

- ▶ **Qt** (последняя на декабрь 2021 версия 6.2.2)
- ▶ wxWidgets
- ▶ GTK+
- ▶ Другие: [slant.co/topics/12868/ cross-platform-c-gui-toolkits](https://slant.co/topics/12868/cross-platform-c-gui-toolkits)

---

<sup>3</sup>коротки обзор фреймворков от Microsoft: [habr.com/ru/post/566352/](https://habr.com/ru/post/566352/)

**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

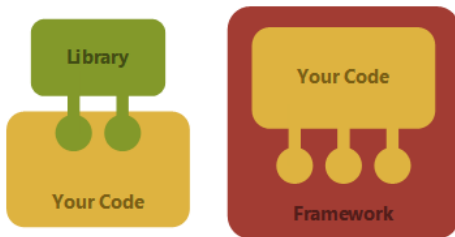
- ▶ Библиотека определяет как пользовательский<sup>4</sup> код будет с ней взаимодействовать, архитектуру самой программы определяет программист.
- ▶ Фреймворк определяет архитектуру программы.
- ▶ Функции библиотеки вызываются пользовательским кодом.
- ▶ Фреймворк вызывает пользовательский код.
- ▶ 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;}           // класс содержащий элементы UI

// 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, в основном обработчики событий
    void on_pushButton_clicked();
private:
    Ui::MainWindow *ui;
    // добавляются поля класса, в том числе реализующие Модель (см. MVC)
};
#endif // MAINWINDOW_H
```

Исходный код создан в Qt Creator: New - Application (Qt) | Qt Widget Application

# Пример использования фреймворка Qt

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <random>

// конструктор главного окна
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),    // явный вызов конструктора базового класса
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    // Объект ui содержит все классы-элементы интерфейса,
    // расположенные на главном окне.
    // класс для ui генерируется автоматически из дизайна окна
}

// обработчик события -- клик на кнопку
// обработчик создан в дизайнера форм
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 Qt проекта

Подходы к созданию приложений с GUI в Qt

- Динамическое создание интерфейса пользователя

- Использование QML

- Создание GUI в редакторе форм

Вопросы

# Приложения построенные на основе QT



Bitcoin Core



Google Earth



KeePass



KStars



LibreCAD



Mathematica



GnuOctave

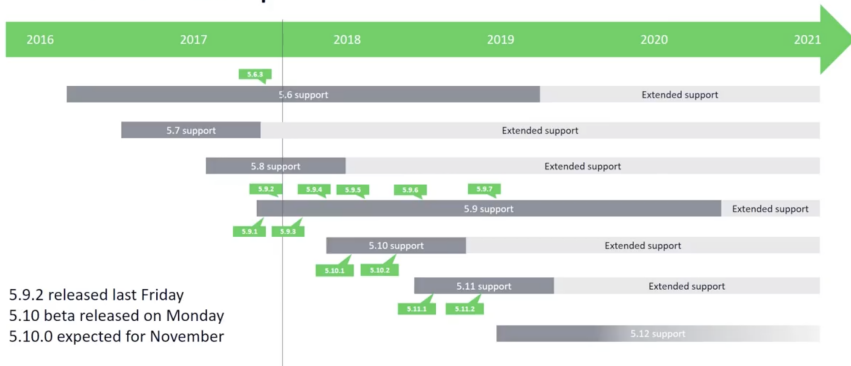


MARBLE

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



## Release roadmap



# Документация QT

- ▶ Документация есть в Qt Creator
- ▶ <http://doc.qt.io>
- ▶ Qt 5.10. Профессиональное программирование на C++.Макс Шлее. 2018 г. 1072 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 (2021 год.) версия фреймвока - 6.  
Версии 4 и 5, 6 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 Qt проекта

Подходы к созданию приложений с GUI в Qt

- Динамическое создание интерфейса пользователя

- Использование QML

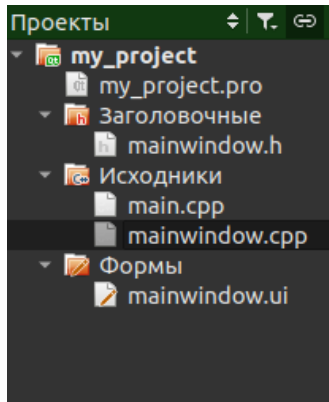
- Создание GUI в редакторе форм

Вопросы

# Структура проекта

На примере шаблона на основе Qt Widgets

- ▶ pro файл – главный файл проекта.  
Содержит:
  - ▶ список имён файлов проекта
  - ▶ список 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`



# Файл проекта

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++17, c++19 – включение поддержки соответствующих стандартов (может понадобится задать дополнительно: `QMAKE_CXXFLAGS += -std=c++19`)
- ▶ thread – включить поддержку потоков

## Файл проекта: библиотеки Qt

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

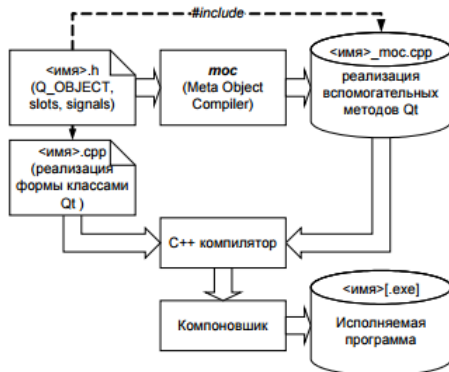
По умолчанию, QT уже включает модули core и gui.

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 compiler
- ▶ 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
};

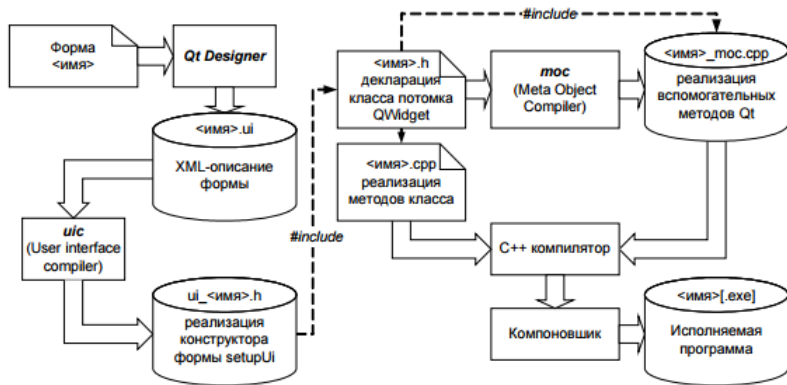
namespace Ui {
    class MainWindow: public Ui_MainWindow {};
} // namespace Ui

QT_END_NAMESPACE
#endif // UI_MAINWINDOW_H

MainWindow->setObjectName(QStringLiteral("MainWindow"));
MainWindow->resize(400, 300);
menuBar = new QMenuBar(MainWindow);
```

# Сборка Qt проекта

## Приложения с GUI



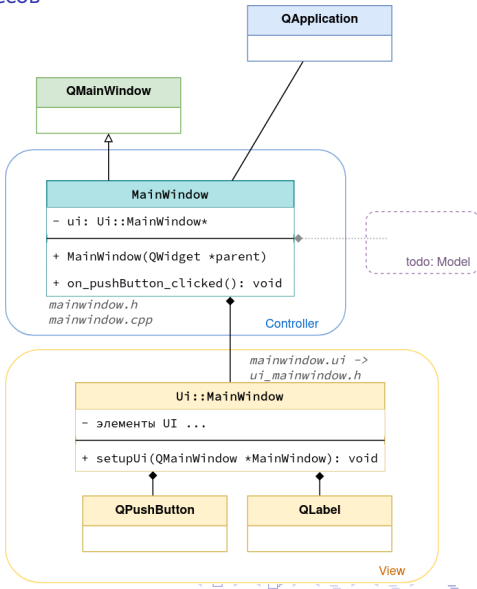
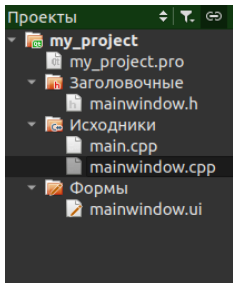


# Сборка Qt проекта

- ▶ 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 и диаграмма классов



# 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 Qt проекта

Подходы к созданию приложений с GUI в Qt

Динамическое создание интерфейса пользователя

Использование QML

Создание GUI в редакторе форм

Вопросы

# Структура приложений с GUI

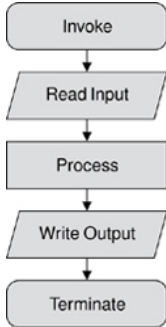
- ▶ Приложение с GUI создаются согласно парадигме событийно-ориентированного программирования
- ▶ Такая программа должна реагировать на события создаваемые пользователем и операционной системой
- ▶ Программа содержит главный цикл – **цикл событий** (event loop)
- ▶ Он отвечает за получение и передачу событий
- ▶ Большинство событий обрабатывает главное окно программы посредством методов класса главного окна.

# Структура приложений с GUI

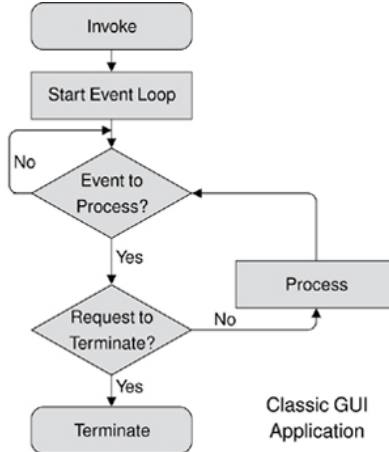
- ▶ Главный цикл приложения в псевдокоде

```
initialize()
while message != quit
    message := get_next_message()
    process_message(message)
end while
```

- ▶ Фреймворки для созданий приложений с GUI уже содержат в себе реализацию главного цикла  
В Qt главный цикл реализован в методе `exec()` классе `QApplication`
- ▶ Задача программиста – создать обработчики событий (как правило в классе главного окна)



Classic  
Batch-processing  
Application



Classic GUI  
Application

# Outline

Трудности создания программ с GUI

Qt

Структура Qt проекта

Подходы к созданию приложений с GUI в Qt

- Динамическое создание интерфейса пользователя

- Использование QML

- Создание GUI в редакторе форм

Вопросы



# 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:

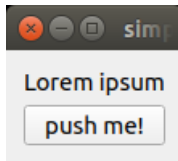
[github.com/VetrovSV/OOP/tree/master/examples/Qt/SignalsAndSlots2](https://github.com/VetrovSV/OOP/tree/master/examples/Qt/SignalsAndSlots2)

# 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();
}
```



# Undefined reference

Кроме заголовочных файлов, к проекту нужно подключать соответствующие модули Qt.

Если этого не сделать, то компилятор укажет на то, что используемые имена не определены.

```
In function `main':  
/home/sotona/w/lab/cpp/qtlab/build-qtlab-Desktop_Qt_5_9_1_GCC_64bit-Debug/helloworld.o  
❗ undefined reference to `QApplication::QApplication(int&, char**, int)'  
❗ undefined reference to `QLabel::QLabel(QString const&, QWidget*, QFlags<Qt::WindowType>)'  
❗ undefined reference to `QWidget::show()'  
❗ undefined reference to `QApplication::exec()'  
❗ undefined reference to `QLabel::~QLabel()'  
❗ undefined reference to `QApplication::~QApplication()'
```

# Outline

Трудности создания программ с GUI

Qt

Структура Qt проекта

Подходы к созданию приложений с GUI в Qt

Динамическое создание интерфейса пользователя

Использование QML

Создание GUI в редакторе форм

Вопросы

Использование декларативного языка описания QML и JavaScript.

Гибкий инструмент описания и настройки внешнего вида GUI.

Подробнее рассматривается в [OOP\\_2\\_8\\_UI\\_markup\\_language.pdf](#)

# Outline

Трудности создания программ с GUI

Qt

Структура Qt проекта

Подходы к созданию приложений с GUI в Qt

Динамическое создание интерфейса пользователя

Использование QML

Создание GUI в редакторе форм

Вопросы

# Создание GUI в редакторе форм

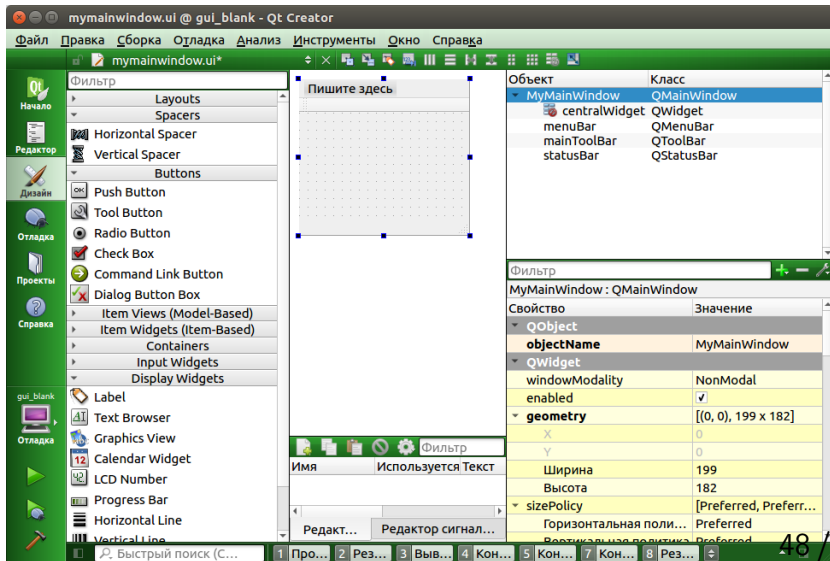
- ▶ Создание GUI в редакторе форм, Qt Creator автоматически генерирует соответствующие классы и отношения между ними.

Используются ui файлы. Универсальный подход.

Пример кода приведён на слайде 11 и последующих

# Создание интерфейса пользователя

Файл \*.ui - текстовый XML файл, но в QtCreator автоматически открывается в дизайнера форм.





# Создание интерфейса пользователя

Элементы интерфейса пользователя в Qt называются **виджетами** (widgets). Они представляют собой классы, которые имеют графическое представление (например в виде кнопки).

В самом начале форма - окно программы, уже включает в себя несколько элементов интерфейса пользователя.

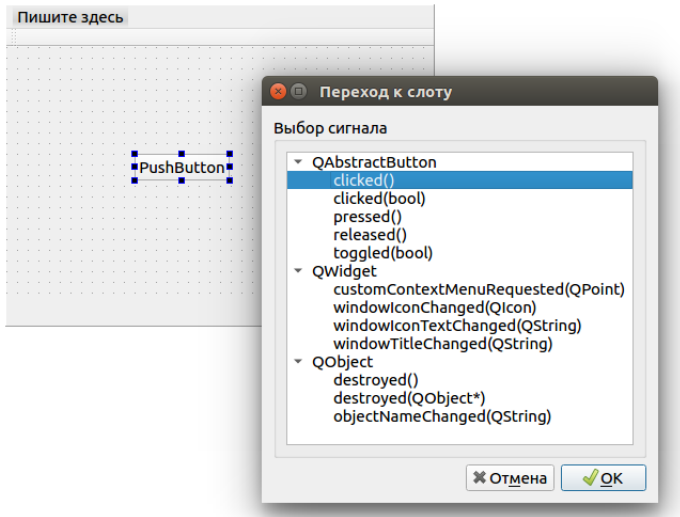
- ▶ menuBar - строка меню;
- ▶ mainToolBar - панель инструментов;
- ▶ statusBar - панель состояния;
- ▶ CentralWidget - основной виджет (пустое пространство на форме), который будет содержать остальные элементы интерфейса.

Все эти элементы интерфейса в шаблоне пусты.

## Создание обработчиков событий - слотов

- ▶ Обработчики событий в фреймворке Qt называются **слотами**.
- ▶ Каждый элемент интерфейса может реагировать на свои наборы событий. Они приведены в справке.
- ▶ Обработчики событий – это методы класса "главное окно" (MainWindow).
- ▶ Эти методы создаются в редакторе форм, и автоматически добавляются в класс. "Перейти к слоту" в контекстном меню элемента GUI  
Разработчику остаётся описать реакцию на событие внутри метода.
- ▶ Название методов создаётся их названия элемента GUI. Поэтому рекомендуется давать объектам – элементам интерфейса осмысленные имена, говорящие о их назначении или совершаемом действии.

# Создание обработчиков событий – слотов



## Создание обработчиков событий – слотов

mainwindow.h

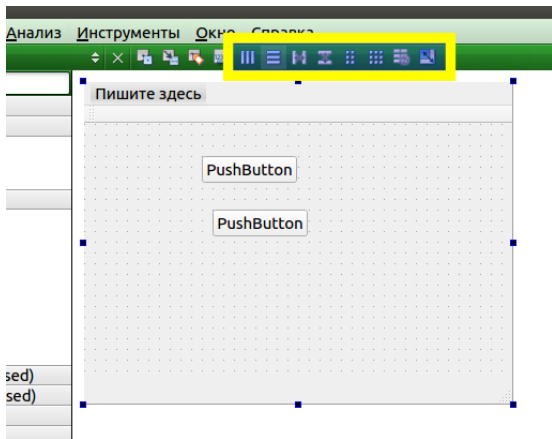
```
...  
private slots:  
    void on_pushButton_random_clicked();  
...
```

mainwindow.cpp

```
...  
MyMainWindow::MyMainWindow(QWidget *parent) :  
    QMainWindow(parent), ui(new Ui::MyMainWindow){  
    // все классы-элементы интерфейса агрегированы в ui  
    ui->setupUi(this);}  
  
void MyMainWindow::on_pushButton_random_clicked(){  
    // реакция на нажатие кнопки  
}  
...
```

# Компоновщики виджетов

**Менеджеры компоновки (Layouts)** - классы управляющие геометрией и расположением других виджетов.



## Компоновщики виджетов

Менеджеры компоновки создаются в дизайнере форм. Для подстройки размещения элементов интерфейса их выделяют и выбирают подходящий компоновщик.

Для контроля размера виджета используют настраивают значения полей `maximimize` и `minimize`.

Дополнительно для создания "пустоты" вокруг виджетов применяют "пружины" (Spacers).

Процесс компоновки виджетов идёт поэтапно, снизу вверх: сначала элементы интерфейса объединяются в небольшие группы, затем компонуются сами группы, наконец создаётся компоновщик для всего содержимого главного окна.

# Outline

Трудности создания программ с GUI

Qt

Структура Qt проекта

Подходы к созданию приложений с GUI в Qt

- Динамическое создание интерфейса пользователя

- Использование QML

- Создание GUI в редакторе форм

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 5.10. Профессиональное программирование на C++. Макс Шлее. 2018 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)