

# Qt

## Введение

Кафедра ИВТ и ПМ

2017



# Фреймворк

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

## Фреймворк и библиотека

Библиотека определяет как пользовательский код будет с ней взаимодействовать, архитектуру определяет программист.

Фреймворк определяет архитектуру программы.

Функции библиотеки вызываются пользовательским кодом.

Фреймворк вызывает пользовательский код.

Фреймворк может содержать в себе множество библиотек различного назначения.



# Фреймворк

Как правило при работе с фреймворком (для создания приложений с GUI) программисту предоставляется один основной класс - главное окно, наследника от которого нужно реализовать.

Бизнес логика приложения как правило реализуется с помощью агрегирования пользовательских классов и других классов фреймворка, а также реализации методов основного класса.

Методы основного класса генерируются автоматически средствами IDE, а агрегирование классов представляющих себя элементы интерфейса - с помощью дизайнера форм.



## Пример использования фреймворка Qt

```
#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, р
```

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

```
}
```

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

```
    // Пользовательский код
```

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

```
}
```



# Зачем нужны фреймворки?

- ▶ В стандартную библиотеку языка как правило не входят модули для быстрого построения приложений с GUI.
- ▶ Использование платформозависимого кода (например Win API<sup>1</sup>) для создания интерфейса - не эффективно.
- ▶ Необходимо каждый раз использовать один и тот же шаблон кода для разработки нового приложения.

---

<sup>1</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.



# Фреймворки для C++

- ▶ Qt
- ▶ Microsoft Foundation Classes (MFC)
- ▶ Windows Forms (составная часть .NET)
- ▶ GTK+
- ▶ wxWidgets
- ▶ Fast, Light Toolkit (FLTK)



# Приложения построенные на основе QT



Bitcoin Core



Google Earth



KeePass



Mathematica



GnuOctave



KStars



LibreCAD



MARBLE



## Особенности Qt

- ▶ простой API
- ▶ Поддержка разных платформ: Windows, Linux, MacOS, Android и др.
- ▶ Классы для работы с сетью, БД, потоками, файловой системой и т.п.
- ▶ STL совместимая библиотека контейнеров
- ▶ Система сигналов и слотов
- ▶ 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.





# Метаобъектная система

**Метаобъектная система** — часть ядра фреймворка для поддержки в C++ таких возможностей, как сигналы и слоты для коммуникации между объектами в режиме реального времени и динамических свойств системы.

Метаобъектная система содержит: класс `QObject`, макрос `Q_OBJECT` и утилиту `moc` (метаобъектный компилятор).

**QObject** — это базовый класс для всех Qt-классов.

Макрос **Q\_OBJECT** используется для включения метаобъектных функций в классах и на этапе компиляции работает как препроцессор, который преобразует применения макроса в исходный код C++.



# 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, на вызов другого в режиме реального времени.

Сигналы и слоты реализованы также в библиотеке boost.



**Событийно-ориентированное программирование** (event-driven programming) — парадигма программирования, в которой выполнение программы определяется событиями — действиями пользователя (клавиатура, мышь), сообщениями других программ и потоков, событиями операционной системы (например, поступлением сетевого пакета).

Программа написанная в соответствии с этой парадигмой содержит обработчик событий (как правило выполняющийся в отдельном потоке) и код, вызываемый обработчиком.

При использовании фреймворков для создания GUI программисту необходимо описать только реакцию на события, а обработчик предоставляется самим фреймворком.



# Структура проекта

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



## \*.pro - файл проекта

Описание и настройка параметров проекта происходит заданием значений переменных проекта:

- ▶ **CONFIG** - общие настройки проекта и компилятора
- ▶ **QT** - список модулей Qt используемых проектом
- ▶ **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 (+=) в переменную требуемых значений.



# 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++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`



## 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

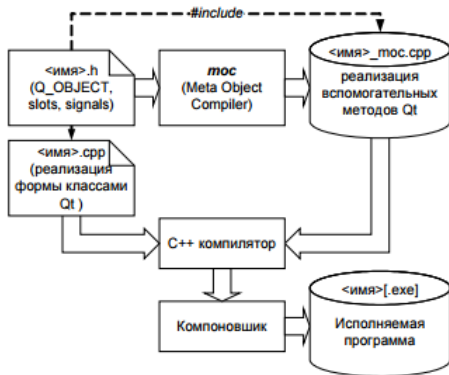
FORMS += \
    mainwindow.ui
```





# Работа компилятора qmake

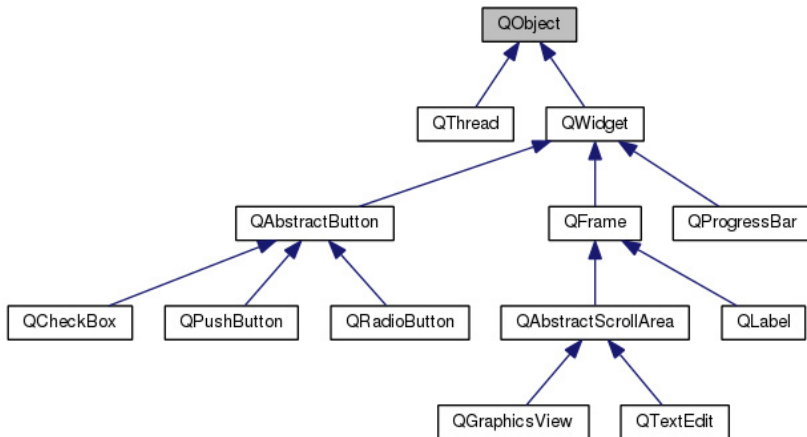
Механизм сигналов и слотов не является частью языка C++, поэтому исходные коды сначала транслируются в чистый C++, а затем уже компилируются в бинарный файл.





# 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"



# Подходы к созданию приложений с GUI в Qt

- ▶ Создание GUI динамически, во время запуска или выполнения программы.  
Подходит для небольших программ.
- ▶ Создание GUI в редакторе форм, Qt Creator автоматически генерирует соответствующие классы и отношения между ними.
- ▶ Использование декларативного языка описания QML и JavaScript.  
Гибкий инструмент описания и настройки внешнего вида GUI.



# Hello World

Пример простейшего консольного приложения Qt.

файл проекта

```
QT -= gui    # отключим поддержку GUI
```

# настройка проекта

```
CONFIG += c++11 console    # поддержка C++11, консольное приложение
```

```
CONFIG -= app_bundle      # отключение
```

```
SOURCES += main.cpp
```

файл исходных кодов

```
#include <QCoreApplication>
```

```
#include <iostream>
```

```
int main(int argc, char *argv[]){
```

```
    QCoreApplication a(argc, argv);
```

```
    std::cout << "Hello world\n";
```

```
    return a.exec();
```

```
}
```



# 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

