

ООП

Семестр 2. Лекция 2

Кафедра ИВТ и ПМ

2018



План

Прошлые темы

qDebug. Отладочный вывод

QObject

Core classes

QWidget

Outline

Прошлые темы

qDebug. Отладочный вывод

QObject

Core classes

QWidget

Сигналы и слоты

- ▶ Что такое сигнал?



Сигналы и слоты

- ▶ Что такое сигнал?

Сигнал метод вызываемый во время события.

Что такое слот?



Сигналы и слоты

- ▶ Что такое сигнал?

Сигнал метод вызываемый во время события.

Что такое слот?

слот - метод, принимающий сигнал.



Сигналы и слоты

- ▶ Как объявляются сигналы и слоты?
- ▶ В каком классе можно объявлять сигналы и слоты?

Сигналы и слоты

- ▶ Как объявляются сигналы и слоты?
- ▶ В каком классе можно объявлять сигналы и слоты?
В классе построенном на основе QObject
- ▶ Какие макросы нужно использовать при описании этого класса?

- ▶ Как объявляются сигналы и слоты?
- ▶ В каком классе можно объявлять сигналы и слоты?
В классе построенном на основе QObject
- ▶ Какие макросы нужно использовать при описании этого класса?

Макрос `Q_OBJECT` позволяет moc (met object compiler) транслировать код класса на чистый C++.



Сигналы и слоты

```
class A : public QObject{
    Q_OBJECT
    ...
signals:
    void my_signal();  // реализация не нужна
};

class B : public QObject{
    Q_OBJECT
    ...
public slots:
    void my_slot();  // нужно реализовать
};
```



Сигналы и слоты

- ▶ Как соединить сигнал со слотом?



- ▶ Как соединить сигнал со слотом?

```
QObject::connect(button, SIGNAL(clicked(bool)),  
                 label, SLOT(clear()));
```

или

```
QObject::connect(button, &QPushButton::pressed,  
                 label, &QLabel::hide);
```



Сигналы и слоты

- ▶ Как организовать вызов сигнала и слота с одинаковым фактическим параметром?



Сигналы и слоты

- ▶ Как организовать вызов сигнала и слота с одинаковым фактическим параметром?
Фактический параметр будет одинаковый для сигнала и слота если у них одинаковы формальные параметры.

```
QObject::connect(spinBox, &QSpinBox::valueChanged,  
                label,    &QLabel::setNum);
```



Outline

Прошлые темы

qDebug. Отладочный вывод

QObject

Core classes

QWidget

Отладочный вывод

В целях изучения Qt полезно использовать так называемый отладочный вывод.

Информация будет напечатана в консоль. Для GUI приложений вывод отладчика можно просмотреть в Qt Creator.

- ▶ Для отладочного вывода используется объект класса `QDebug`.
- ▶ Этот объект объявлен в модуле `QDebug`
- ▶ Чтобы получить к нему доступ используется функция `QDebug()`
- ▶ Для вывода используется оператор «



Отладочный вывод

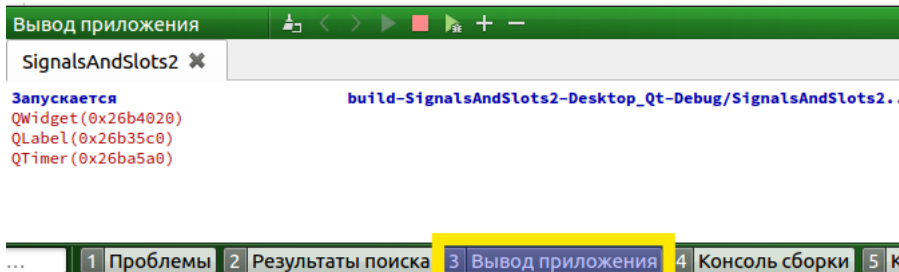
Преимущество вывода через QDebug перед cout:

- ▶ QDebug может выводить все объекты унаследованные от QObject
- ▶ QDebug может выводить на экран тип объекта при задействованном позднем связывании.



Отладочный вывод

```
QObject * o = new QWidget();  
qDebug() << o; // должен быть использован указатель  
o = new QLabel();  
qDebug() << o;  
o = new QTimer();  
qDebug() << o;
```



Outline

Прошлые темы

qDebug. Отладочный вывод

QObject

Core classes

QWidget

QObject

```
QT += core
```

```
#include <QObject>
```

QObject - базовый виртуальный класс для всех остальных классов в Qt.

Любой класс построенный на основе QObject должен содержать макрос Q_OBJECT.

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



QObject

Соединение и разъединение сигналов и слотов

- ▶ connect - соединения сигнала и слота (статический метод)

```
connect(const QObject *sender,  
        PointerToMemberFunction signal,  
        const QObject *receiver,  
        PointerToMemberFunction method)
```

- ▶ disconnect - разрыв связи между сигналом и слотом

```
disconnect(const QObject *receiver,  
           const char *method = Q_NULLPTR) const
```



QObject

Конструктор

```
QObject(QObject *parent = Q_NULLPTR)
```

В конструкторе QObject можно указать ссылку на владельца данного объекта.

При уничтожении владельца, автоматически будут вызваны деструкторы всех его дочерних¹ объектов.

Это даёт возможность не заботиться о уничтожении многих объектов создаваемых динамически.

Например владельцем всех элементов интерфейса будет класс главного окна.

¹под дочерними объектами понимаются не наследники, а агрегированные объекты, т.е. объекты время жизни которых зависит от времени жизни родительского объекта



QObject

- ▶ `const QObjectList &QObject::children() const`
- ▶ `QObject *QObject::parent() const`

Так как все классы строятся на основе виртуального QObject то тип возвращаемых элементов будет определён во время выполнения программы.



О документации

В документации в описание каждого класса включены свойства (**properties**). Это в основном закрытые поля класса, доступ к которым возможен только с помощью методов.

Как правило методы возвращающие значения названы так же как и свойства, а методы устанавливающие значения начинаются с префикса `get`.

objectName

Свойство objectName содержит строку (QString) - имя *объекта*.

Это свойство можно использовать чтобы найти определённый объект по имени.

По умолчанию объект имеет пустое имя.

Методы доступа:

```
QString      objectName() const  
void        setObjectName(const QString &name)
```



QMetaObject

В Qt каждый класс содержит метаинформацию о самом себе.

Эта метаинформация содержится в классе QMetaObject. Она включает:

- ▶ `className()` - имя класса
- ▶ информацию о классе предоставленную разработчиком с помощью макроса `Q_CLASSINFO`
- ▶ Информацию о методах (их количество, названия, ...)



Q_CLASSINFO

Данная метаинформация о классе представлена в виде пар имя-значение.

В примере приведена информация об авторе класса и ссылка на сайт.

```
class MyClass : public QObject
{
    Q_OBJECT
    Q_CLASSINFO("Author", "Pierre Gendron")
    Q_CLASSINFO("URL", "http://www.my-organization.qc.ca")

public:
    ...
};
```



QMetaObject

```
QLabel label("Hello World!");
```

```
qDebug() << label.metaObject()->className();
```

```
qDebug() << label.metaObject()->methodCount();
```

Вывод:

QLabel

44



Outline

Прошлые темы

qDebug. Отладочный вывод

QObject

Core classes

QWidget

Рекомендации

- ▶ Перед решением задачи и написанием собственного кода следует проверить документацию на наличие подходящих классов.
- ▶ Перед использованием класса следует познакомиться с его документацией.
- ▶ Если не существует готовых решений, то следует изучить **лучшие практики** (best practice).



Справка Qt

F1 - вызов справки по классу (или функции), на который установлен курсор.

Справка по классу обычно состоит из

- ▶ общего описания класса
- ▶ Properties - списка свойств (полей класса и методов доступа к ним),
- ▶ Public Functions - открытых методов,
- ▶ Public Slots - открытых методов, которые вызываются в ответ на события.
- ▶ Закрытых методов
- ▶ Detailed Description - подробного описания класса, в котором могут быть приведены примеры его использования.



Основные классы Qt

Qt содержит свои версии классов стандартной библиотеки C++ и множество других, которые можно использовать для хранения данных, работы с файловой системой, операционной системой, временем и т.д.

Однако стоит помнить о том, что использующее эти классы приложение должно поставляться вместе с библиотеками qt.



Проблема бананов, обезьян и джунглей

Проблема с ОО-языками заключается в том, что они тянут за собой всё своё окружение. Вы хотели всего лишь банан, но в результате получаете гориллу, держащую этот банан, и все джунгли впридачу.

—Джо Армстронг, создатель Erlang



Core classes

Файл проекта:

```
QT += core
```

Почти все Qt классы (не только основные) содержатся в одноимённых заголовочных файлах.

Например QPoint:

```
##include <QPoint>
```

Заголовочные файлы в некоторых других классов (по большей части это разного рода виджеты) могут находиться в отдельных каталогах фреймворка:

```
##include <QtWidgets/QLabel>
```

Заголовочный файл и модуль Qt указывается для каждого класса в документации.



Core classes

Классы предназначены для работы с данными, файловой системой, временем, исключением и т.п. содержатся в ядре фреймворка - модуле core.

Некоторые из **core classes**

QSize QRect QPoint

QString QVector QStringList QStack QSet QPair QMap QList

QTime QDate QTimer

QException

QRegExp

QUrl

QTextStream QFile

QMessageLogger



QString

Класс для хранения строк в Unicode кодировке.

```
QString str;
```

```
// Число -> строка
```

```
str.setNum(1234);           // str == "1234"
```

```
// Строка -> число
```

```
float x = str.toFloat();
```

```
// Число -> строка. Статический метод
```

```
QString s = QString::number(42.05)
```

```
// Возвращает строку без пробелов в начале и конце
```

```
str = str.trimmed();
```

```
// в std::string
```

```
std::string s = str.toStdString()
```



Информация о каталогах и их структуре.

```
// Упрощает работу с путями к файлам
QDir directory("Documents/Letters");
QString path = directory.filePath("contents.txt");
QString name = directory.dirName();
QString absPath = directory.absoluteFilePath("contents.txt");

// текущая папка
QDir cdir = QDir::current();
QString cdir_path = QDir::currentPath();
// Папка пользователя
QDir home = QDir::home();
```



QDir

```
// проверка на существование
if ( dir.exists() ){... }

// Список имён файлов
QStringList dd = d.entryList();

// сменить папку (текущая папка для прогр. не меняется)
home.cd("another-dir");

// изменить текущую папку программы
if ( QDir::setCurrent("another-dir") ){
    // папка поменялась
}
```



QFile

Чтение и запись данных в файлы.

```
QFile f("myfile");

if ( !f.open(QFile::WriteOnly) )
    // не удалось открыть файл ;
;
char *data = "some data 12345 \n";
f.write(data, strlen(data));
f.close();

if ( !f.open(QFile::ReadOnly) )
    // не удалось открыть файл ;

char buf[1024];
qint64 r = f.readLine(buf, 1024);
if ( r!=-1 )
    // прочитано r байт
```



QTextStream

Упрощает работу с текстовыми файлами.

```
// создаём класс-файл
QFile data("output.txt");
// Открываем файл для записи
if (data.open(QFile::WriteOnly)) {

    // Для удобства записи разных типов данных
    // в текстовый файл
    // используем этот класс
    QTextStream out(&data);

    out << "Result: " << qSetFieldWidth(10) << left << 3.14
    // writes "Result: 3.14          2.7          "
}
```



QTextStream

Упрощает работу с текстовыми файлами.

```
QFile data("output.txt");
if (data.open(QFile::ReadOnly)) {
    QTextStream in(&data);
    QString str = in.readLine();
    // уберём лишние пробелы в начале и в конце
    str = str.trimmed();
    // заменим повторяющиеся пробелы на один
    unsigned n;
    do{ n = str.length();
        str = str.replace("  ", " ");
    } while (n!=str.length());

    // разделим строку по пробелам
    QStringList sl = str.split(" ");
    x = sl[1].toFloat();
    x = sl[3].toFloat();
```



QTimer

```
QTimer *timer = new QTimer;  
QObject::connect(timer, &QTimer::timeout,  
                 [](){qDebug() << ".";});  
timer->setInterval(1500); // в миллисекундах  
timer->start();
```

```
// Таймер с одиночным срабатыванием  
QTimer::singleShot(200, объект, SLOT(метод));  
// после срабатывания будет вызван  
// метод указанного объекта
```

В примере использована лямбда функция, но соединить сигнал таймера timeout можно с любой функцией или методом.



Outline

Прошлые темы

qDebug. Отладочный вывод

QObject

Core classes

QWidget

QWidget - основной класс для всех элементов интерфейса пользователя.

Он принимает события мыши и клавиатуры, рисует самого себя на экране.

Все классы виджетов наследуются от QWidget, поэтому они имеют много общих методов и полей.



Виджеты

При создании окна с несколькими элементами интерфейса один из виджетов должен быть главным.

Такая иерархия достигается за счёт агрегации подчинённых виджетов в главный.

Если виджеты создаётся в дизайнерае форм Qt Creator'a, то эта связь устанавливается автоматически во время генерации `cpp` файла из `ui` файла формы.



Виджеты

Если виджеты создаются вручную, то может понадобится у подчинённых виджетов при вызове конструктора передать параметром указатель на основной виджет.

```
QWidget::QWidget(QWidget *parent = Q_NULLPTR)
```

Если подчинённые виджеты добавляются сначала на компоновщик, а уже потом на основной виджет. То они автоматически становятся дочерними виджетами по отношению к основному.

Таким образом все виджеты находящиеся в окне, в конечном итоге агрегируются в основной виджет.

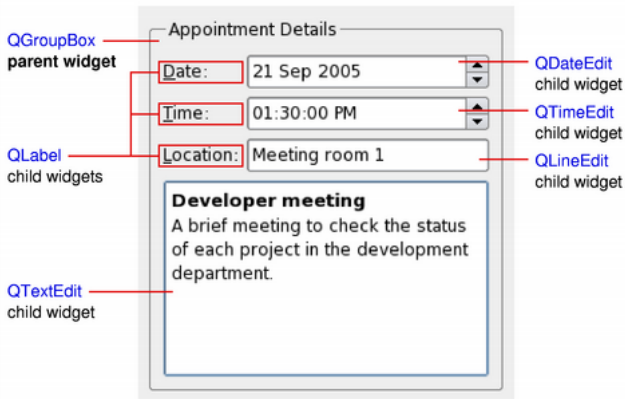


Виджеты

```
QWidget w;  
QVBoxLayout *layoyt = new QVBoxLayout();  
  
QPushButton *b = new QPushButton("PushMe");  
QPushButton *b2 = new QPushButton("Show Label");  
QLabel *l = new QLabel("I am Label");  
  
layoyt->addWidget(b);  
layoyt->addWidget(l);  
layoyt->addWidget(b2);  
w.setLayout(layoyt);  
  
qDebug() << w.children();  
// (QVBoxLayout(0xa8f4f0), QPushButton(0xd3e7c0),  
// QLabel(0xc70c70), QPushButton(0xc728d0))
```



Основным виджетом может выступать либо пустой виджет, либо другие виджеты-контейнеры. Например QTabWidget, QGroupBox и другие.



QWidget содержит свойства отвечающие за размер и положение элемента интерфейса пользователя, однако ручная работа с ними в большинстве случаев не рекомендуется.

За изменение размеров элемента интерфейса пользователя (виджетов) должен отвечать отдельный класс - компоновщик (layout), который будет автоматически менять ширину, высоту и положение виджета в зависимости от размеров окна.

В дизайнера форм можно задавать ограничения размера в контекстном меню виджета.




Поля класса лучше всего изменять в дизайнере форм QtCreator'a.

При изменении свойств основного виджета (на котором расположены другие виджеты), аналогично изменяются и свойства всех дочерних. Так например можно изменить шрифт одновременно во всём окне.

По каждому из свойств доступна справка:
выделить свойство -> F1



Свойство	Значение
▼ QObject	
objectName	centralWidget
▼ QWidget	
enabled	<input checked="" type="checkbox"/>
▶ geometry	[(0, 39), 736 x 487]
▶ sizePolicy	[Preferred, Preferred, 0, 0]
▶ minimumSize	0 x 0
▶ maximumSize	16777215 x 16777215
▶ sizeIncrement	0 x 0
▶ baseSize	0 x 0
palette	Унаследованная
▶ font	A [Ubuntu, 11]
cursor	 Arrow
mouseTracking	<input type="checkbox"/>
tabletTracking	<input type="checkbox"/>
focusPolicy	NoFocus
contextMenuPolicy	DefaultContextMenu
acceptDrops	<input type="checkbox"/>
▶ toolTip	
toolTipDuration	-1
▶ statusTip	
▶ whatsThis	
▶ accessibleName	
▶ accessibleDescription	
layoutDirection	LeftToRight



События QWidget

При изменении некоторых свойств виджета вызываются *обработчики* соответствующего события.

Например при изменении размера (вызов `resize(w, h)`) окна вызывается слот

```
resizeEvent(QResizeEvent *event);
```

Некоторые обработчики событий

```
void QWidget::showEvent(QShowEvent *event)
```

```
void QWidget::hideEvent(QHideEvent *event)
```

```
// перерисовка виджета
```

```
void QWidget::paintEvent(QPaintEvent *event)
```

```
void QWidget::closeEvent(QCloseEvent *event)
```

По умолчанию обработчики не имеют реализации или не выполняют никакой работы, но их можно определить внутри класса.



Пример

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

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent), ui(new Ui::MainWindow){
    ui->setupUi(this);

    // Включить отслеживание мыши для данного класса (главного окна)
    setMouseTracking(true);

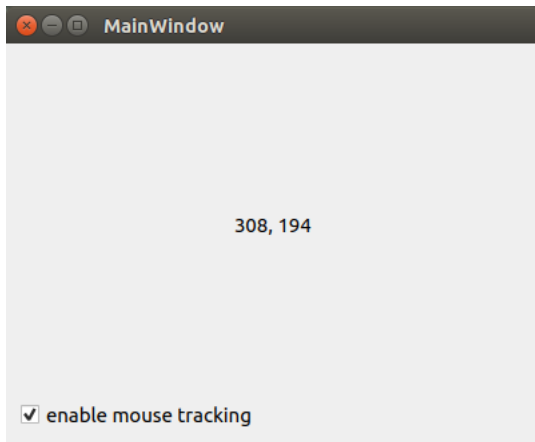
    // Если это не сделать, то обработчик движения мыши будет
    // вызываться только если нажата одна из кнопок мыши
    // Включить отслеживание мыши виджетом,
    // хранящим всё содержимое главного окна
    ui->centralWidget->setMouseTracking(true);
    ui->checkBox->setChecked(
        ui->centralWidget->hasMouseTracking() );}

void MainWindow::mouseMoveEvent(QMouseEvent *e){
    ui->label->setText( QString::number( e->x() ) + ", "
        + QString::number( e->y() ));}

void MainWindow::on_checkBox_stateChanged(int arg1){
    setMouseTracking(arg1);}
```



Пример



События QWidget

Аналогично для событий генерируемых пользователем (клик или движение мыши, нажатие клавиши и т.д.) QWidget содержит виртуальные методы.

```
void QWidget::mouseMoveEvent(QMouseEvent *event)
void QWidget::mousePressEvent(QMouseEvent *event)
void QWidget::mouseDoubleClickEvent(QMouseEvent *event)

void QWidget::keyPressEvent(QKeyEvent *event)

void QWidget::contextMenuEvent(QContextMenuEvent *event)

void QWidget::dragEnterEvent(QDragEnterEvent *event)
```

При вызове обработчика ему передаётся в параметр объект описывающий соответствующее событие. Например координаты мыши или код нажатой клавиши.



События QWidget

Эти не специфические события для виджета нужно определять в производном от него классе вручную.

Обработчики для событий *других* виджетов, расположенных на данном окне создаются из контекстного меню конкретного виджета в дизайнере форм (go to slot...).



Основные свойства QWidget

► Размер

```
width();    // ширина в пикселях
```

```
height();   // высота в пикселях
```

```
resize( int w, int h); // задать размер виджета
```

```
// задать и зафиксировать размер
```

```
setFixedSize(w, h)
```



Основные свойства QWidget

- ▶ **enabled** : bool

Свойство отвечающее за "включение/выключение" элемента интерфейса.

```
bool isEnabled() const
```

```
void setEnabled(bool)
```

- ▶ **visible** : bool

- виден ли элемент интерфейса пользователю. bool

```
isVisible() const
```

```
virtual void setVisible(bool visible)
```



Основные свойства QWidget

- Фокус ввода с клавиатуры

focus : bool

```
bool hasFocus() const
```

```
void setFocus()
```

// Можно ли устанавливать фокус ввода?

// Каким способом устанавливать фокус ввода?

```
focusPolicy() const
```

```
void setFocusPolicy(Qt::FocusPolicy policy)
```



Ссылки и литература

1. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений. 720 с. 2010 г. 700 страниц. Теория. Примеры на C++. Картинки! Вторая половина книги - примеры OOA и OOD с UML диаграммами.
2. MSDN - Microsoft Developer Network
3. Qt 5.X. Профессиональное программирование на C++. Макс Шлее. 2015 и более поздние издания г. 928 с. Книга периодически обновляется с выходом новых версий фреймворка Qt.
4. www.stackoverflow.com - система вопросов и ответов
5. draw.io — создание диаграмм.



Материалы курса

Слайды, вопросы к экзамену, задания, примеры

github.com/VetrovSV/OOP

