

# ООП

## Языки описания пользовательского интерфейса

Кафедра ИВТ и ПМ  
ЗабГУ

2018

# План

Прошлые темы

Язык описания UI  
QML

Ссылки

# Outline

Прошлые темы

Язык описания UI  
QML

Ссылки

# Прошлые темы

- ▶ Что такое бизнес-логика?
- ▶ Что такое шаблон проектирования *Модель-Представление*?
- ▶ Что такое сигнал?
- ▶ Что такое слот?
- ▶ Какие классы могут содержать сигналы и слоты?
- ▶ Что такое UI?
- ▶ Что такое API?

# Outline

Прошлые темы

Язык описания UI

QML

Ссылки

# Front-end и back-end

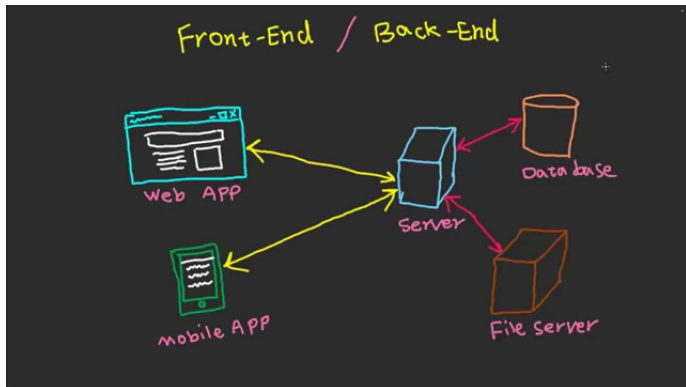
**Фронтенд** (Front-end) — клиентская сторона ПО. Как правило к front-end относят разработку интерфейсов пользователя и т.п., например создание дизайн-макета сайта, вёрстку сайта, создание скриптов описывающих поведение пользовательского интерфейса.

# Front-end и back-end

**Бекенд** (англ. back-end) — программно-аппаратная часть, ядро сайта или программы. Включает в себя бизнес-логику.

Back-end создает, некоторое API, которое использует front-end. Таким образом front-end разработчик не должен знать особенностей реализации сервера, а back-end разработчик реализацию front-end.

# Front-end и back-end



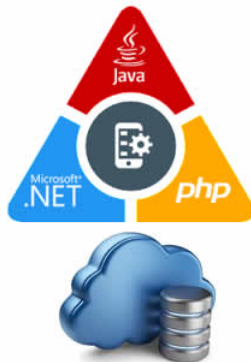


# Front-end и back-end

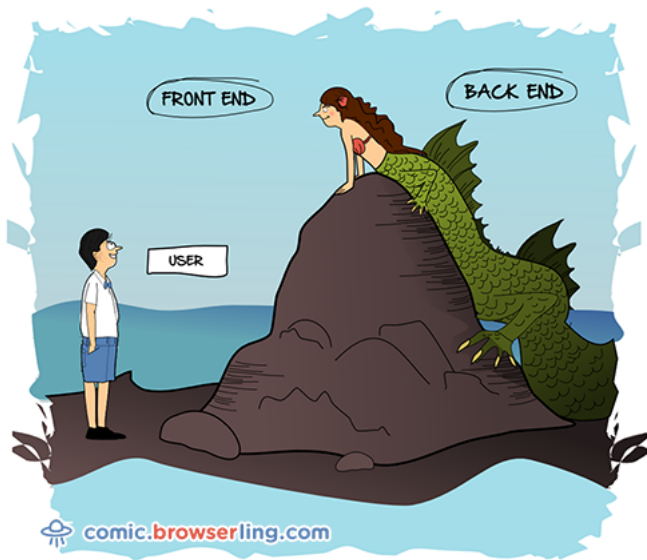
## FRONT END



## BACK END



## Front-end и back-end



Front end vs. Back end.

UI - интерфейс пользователя.

- ▶ то, что пользователь видит
- ▶ то, как пользователь должен пользоваться программой

UI - интерфейс пользователя.

- ▶ то, что пользователь видит
- ▶ то, как пользователь должен пользоваться программой

*Дизайн - это не то, как предмет выглядит, а то, как он работает.*

—Стив Джобс

- ▶ Кто должен проектировать интерфейс пользователя?

- ▶ Кто должен проектировать интерфейс пользователя?
- ▶ Почему это не программист, который понимает как устроена программа?

- ▶ Кто должен проектировать интерфейс пользователя?
- ▶ Почему это не программист, который понимает как устроена программа?
- ▶ Пользователю нет дела до внутреннего устройства программы, для него - программа просто инструмент (или даже помощник) который должен решать задачу пользователя
- ▶ Какой самый простой способ узнать, хорош ли интерфейс пользователя вашей программы?

- ▶ Кто должен проектировать интерфейс пользователя?
- ▶ Почему это не программист, который понимает как устроена программа?
- ▶ Пользователю нет дела до внутреннего устройства программы, для него - программа просто инструмент (или даже помощник) который должен решать задачу пользователя
- ▶ Какой самый простой способ узнать, хорош ли интерфейс пользователя вашей программы?
- ▶ Спросить другого человека.  
Желательно того, кто будет ей пользоваться.  
Желательно не программиста (потому, что у них своё понимание программ)
- ▶ А если никого рядом нет?



- ▶ Кто должен проектировать интерфейс пользователя?
- ▶ Почему это не программист, который понимает как устроена программа?
- ▶ Пользователю нет дела до внутреннего устройства программы, для него - программа просто инструмент (или даже помощник) который должен решать задачу пользователя
- ▶ Какой самый простой способ узнать, хорош ли интерфейс пользователя вашей программы?
- ▶ Спросить другого человека.  
Желательно того, кто будет ей пользоваться.  
Желательно не программиста (потому, что у них своё понимание программ)
- ▶ А если никого рядом нет?
- ▶ *Придумайте* другого человека. Дайте ему имя, представьте его цели, зачем ему нужна программа

**Язык описания интерфейса пользователя** (User Interface, UI) - язык разметки<sup>1</sup> предназначенный для рисования и описания графического интерфейса пользователя (GUI).

Такие языки переносят идея повторного использования кода в область создания интерфейса пользователя, позволяют детально описывать внешний вид элементов интерфейса и иногда их поведение.

---

<sup>1</sup>примеры других языков разметки: HTML, TEX, XML

- ▶ **QML** (Qt Modeling Language) - декларативный<sup>2</sup> язык программирования, основанный на JavaScript, предназначенный для дизайна приложений, делающих основной упор на пользовательский интерфейс.
- ▶ **XAML** (eXtensible Application Markup Language) — основанный на XML язык разметки для декларативного программирования приложений, разработанный Microsoft. XAML также может описывать логику приложений.

<sup>2</sup>см. декларативное и императивное программирование

# XAML. Пример

## Окно с двумя кнопками

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
  <Grid Name="MainGrid">
    <StackPanel Name="StackPanel1">
      <Button Name="Button1">First Button</Button>
      <Button Name="Button2">Second Button</Button>
    </StackPanel>
  </Grid>
</Window>
```

# XAML

Создание приложения "Hello, world" (XAML)

[youtu.be/4DDonpLldLM](https://youtu.be/4DDonpLldLM) - Видеокурс Windows Presentation Foundation (WPF). Урок 1. Введение в WPF и XAML

# Outline

Прошлые темы

Язык описания UI  
QML

Ссылки

Доклад на конференции "C++ Siberia" о QML и Qt Quick:  
Преимущества и использование.

[youtube: Сергей Хомяков, QML Qt Quick на практике](#)

# Особенности QML

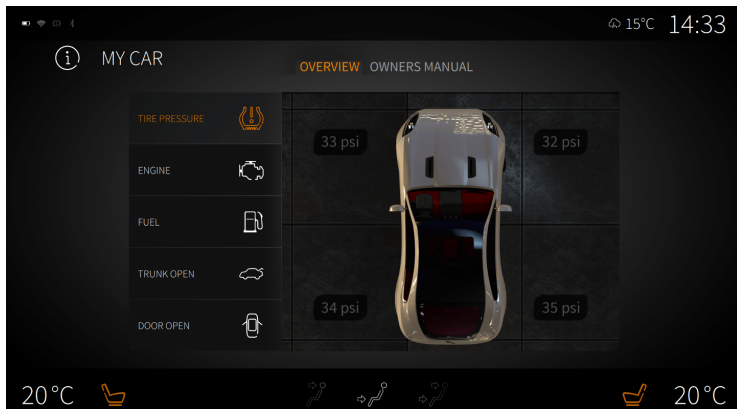
- ▶ Декларативный язык  
Код не компилируется.
- ▶ Синтаксис похож на JSON <sup>3</sup>
- ▶ Интегрируется с C++ кодом (с использованием Qt)
- ▶ Может включать JavaScript, HTML, CSS
- ▶ Структура QML файла - дерево из отдельных элементов UI

---

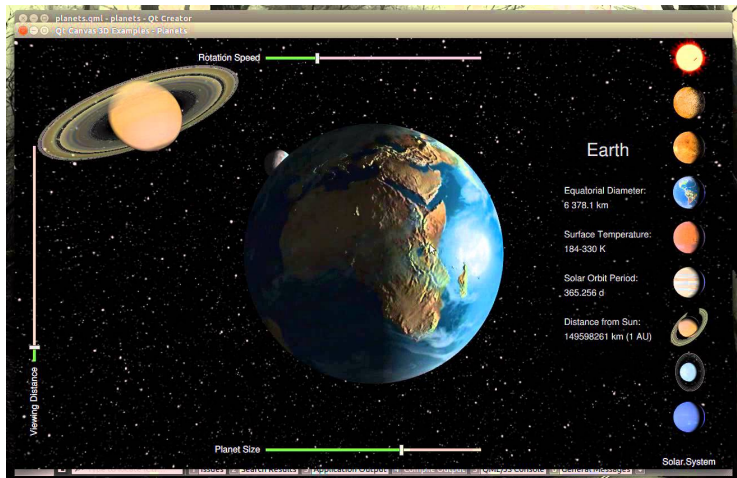
<sup>3</sup>[wikipedia: JSON](#)



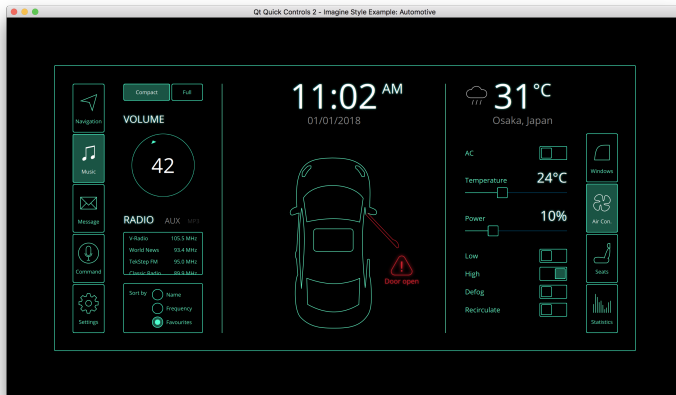
# Примеры QML интерфейсов



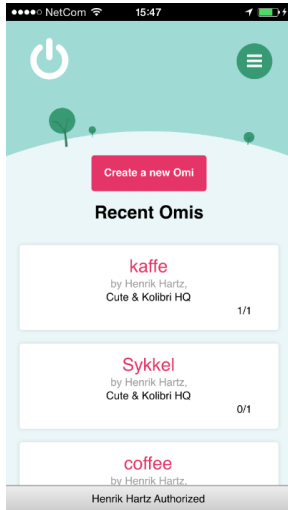
# Примеры QML интерфейсов



# Примеры QML интерфейсов



# Примеры QML интерфейсов

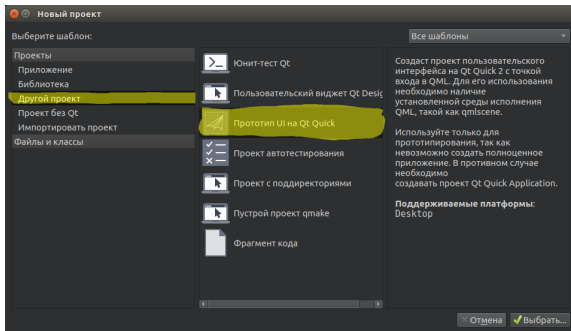


# Особенности QML

- ▶ Гибкий способ описания UI (внешнего вида, поведения, анимаций и т.д.)
- ▶ Использует встроенный в Qt Creator QLM дизайнер (Qt Quick Designer)  
[youtube.com/watch?v=cOViDcYWNCI](https://youtube.com/watch?v=cOViDcYWNCI)
- ▶ возможно использование отдельно от основного приложения для создания прототипов UI

# Создание QML проекта

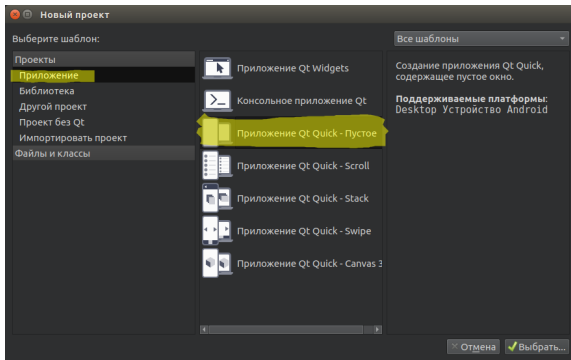
Создание QML проекта из одного QML файла.



Такой QML файл можно запускать в Qt Creator как самостоятельный проект или использовать QML файл в другом проекте.

# Создание QML проекта

## Создание проекта использующего QML и C++



# Создание QML проекта

## Пример

[github.com/VetrovSV/OOP/tree/master/examples/example\\_qml](https://github.com/VetrovSV/OOP/tree/master/examples/example_qml)



# Hello World

## Пример 2.

Файл описания интерфейса:

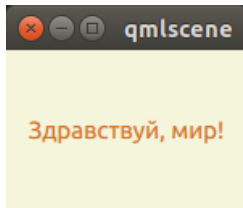
```
import QtQuick 2.0

Rectangle {
    color: "beige"
    width: 150; height: 100

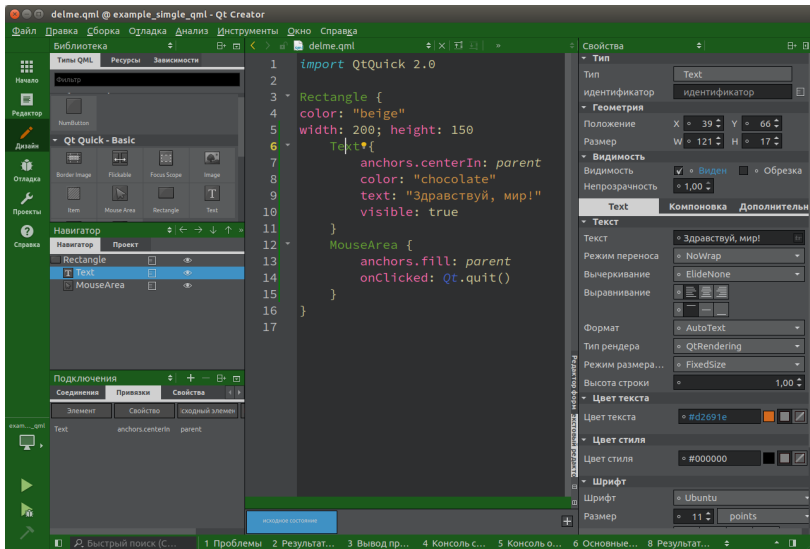
    Text {
        anchors.centerIn: parent
        color: "chocolate"
        text: "Здравствуй, мир!"
    }

    MouseArea {
        anchors.fill: parent
        onClicked: Qt.quit()
    }
}
```

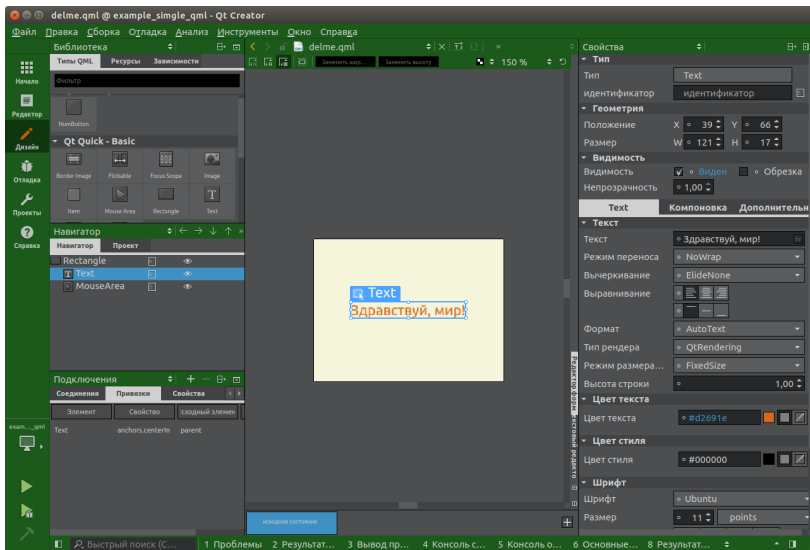
# Hello World



# Qt Quick Designer



# Qt Quick Designer



# Qt Quick Designer

<https://www.youtube.com/watch?v=cOViDcYWNCI> - работа в Qt Quick Designer

# Повторное использование QML

- ▶ Можно создавать свои элементы интерфейса
- ▶ Элементы интерфейса описанные с помощью QML можно сохранять в отдельных файлах, а затем использовать так, как будто это стандартные компоненты.
- ▶ В Qt Quick Designer для этого нужно выбрать элемент интерфейса в навигаторе затем сохранить в отдельном файле через контекстное меню.
- ▶ Имя созданного элемента интерфейса будет совпадать с именем файла, в котором он описан.

Чтобы интерфейс описанный на QML работал вместе с программой на C++ используется Qt Quick

**Qt Quick** – современная технология создания UI, особенностью которой является разделение декларативного описания дизайна интерфейса и императивной логики программирования.

QML является составляющей Qt Quick.

# Как посмотреть интерфейс программы на QML?

- ▶ с помощью Qt Creator
- ▶ с помощью программы **qmlscene**<sup>4</sup>

```
qmlscene my_beautiful_ui.qml
```

Однако qml файл не является отдельной программой, так как его необходимо интерпретировать.

---

<sup>4</sup>расположена в каталоге где установлен Qt, например:  
Qt5/5.11.0/gcc\_64/bin



# Как встроить интерфейс на QML в программу?

В Qt существуют классы, которые выполняют QML код:

- ▶ QQuickView

```
QQuickView *view = new QQuickView;  
view->setSource(QUrl::fromLocalFile("myqmlfile.qml"));  
view->show();
```

Перед использование qml файл должен быть добавлен к проекту. В этой программе не реализовано никакой бизнес логики вне QML файла.

В Qt существуют классы, которые выполняют QML код:

- ▶ QQmlApplicationEngine

```
// Загрузка и разбор qml файла  
QQmlApplicationEngine engine;  
engine.load(QUrl(QStringLiteral("qrc:/main.qml")));  
  
if (engine.rootObjects().isEmpty()) return -1;
```

В QML файле все элементы должны быть помещены  
внутри Window.

Перед использование qml файл должен быть добавлен к  
проекту. В этой программе не реализовано никакой бизнес  
логики вне QML файла.

Рекомендуется использовать именно этот класс.

# Элементы интерфейса

```
import QtQuick.Controls 2.2
```

[doc.qt.io/qt-5/qtquick-controls-qmlmodule.html](http://doc.qt.io/qt-5/qtquick-controls-qmlmodule.html) - список элементов интерфейса и документация

# Некоторые элементы интерфейса

- ▶ Window - главное окно приложения
- ▶ Item - базовый элемент интерфейса (аналог QWidget)
- ▶ Button
- ▶ TextField - поле ввода
- ▶ TextArea - многострочное поле ввода
- ▶ SpinBox - поле ввода для чисел
- ▶ Label
- ▶ Image
- ▶ ComboBox
- ▶ CheckBox

# Настройка отдельных элементов интерфейса

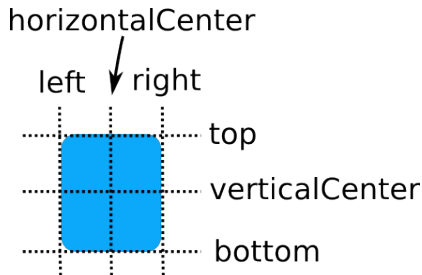
Так как все видимые элементы интерфейса так или иначе построены на основе **Item**, то у них есть общие наборы свойств

- ▶ id - идентификатор объекта
- ▶ x, y
- ▶ z - задаёт "глубину" расположения элемента интерфейса
- ▶ width, height
- ▶ color
- ▶ rotation - угол поворота (по часовой стрелке, в градусах)
- ▶ scale - масштаб
- ▶ visible
- ▶ anchors - составное свойство, описывающее привязку расположения и границ объекта (якоря)
- ▶ opacity - прозрачность (по умолчанию 1.0)

# anchors

**anchors** (якоря) используются для привязки размеров и положения объектов интерфейса к другим объектам.

Например можно привязать надпись к горизонтальному центру окна, или привязать правую границу одного элемента интерфейса, к левой другого.



см. документацию [doc.qt.io/qt-5/qtquick-positioning-anchors.html](http://doc.qt.io/qt-5/qtquick-positioning-anchors.html)

## anchors. Примеры

```
Rectangle { id: rect1; ... }
```

```
Rectangle {  
  id: rect2;  
  // левая граница второго прямоугольника привязана  
  // к правой границе первого  
  anchors.left: rect1.right;  
  // поле между границами  
  anchors.leftMargin: 5; ... }
```

Другие варианты задания якорей:

`anchors.fill: parent` - заполнить родительский объект

`centerIn: parent` - всегда в центре родительского объекта

`horizontalCenter: parent.horizontalCenter` - на  
вертикальной середине родительского объекта

## Layouts

Для управления взаимным положением элементов интерфейса также как и в Qt Designer используются специальные компоновщики.

```
Row { // Горизонтльное расположение
    spacing: 2
    Rectangle { color: "red"; width: 50; height: 50 }
    Rectangle { color: "green"; width: 20; height: 50 }
    Rectangle { color: "blue"; width: 50; height: 20 }
}
```



Здесь для динамического изменения размеров содержимого компоновщика, а не только контроля положения, нужно использовать RowLayout.



# Изображения. Image

```
Image {  
  id: img  
  anchors.fill: parent  
  // В компонент Image можно помещать не только  
  // предварительно добавленные в проект изображения  
  // но и изображения из Интернета  
  source:  
  "https://apod.nasa.gov/apod/image/1805/Pleiades_WiseAntonucci_960.jpg"  
}
```

# Image и BusyIndicator

Если используется изображение из сети, то можно снабдить программу индикатором загрузки

```
...
Image {
    id: img
    anchors.fill: parent
    source:
        "https://apod.nasa.gov/apod/image/1805/Pleiades_WiseAntonucci_960.jpg"
}

BusyIndicator {
    id: load_indicator
    anchors.centerIn: parent
    running: img.status === Image.Loading
}
...
```

# Image

Помимо якоря для изображения можно задавать *способ* заполнения родителя - `fillMode`:



[doc.qt.io/qt-5/qml-qtquick-image.html#fillMode-prop](http://doc.qt.io/qt-5/qml-qtquick-image.html#fillMode-prop)

# Градиент

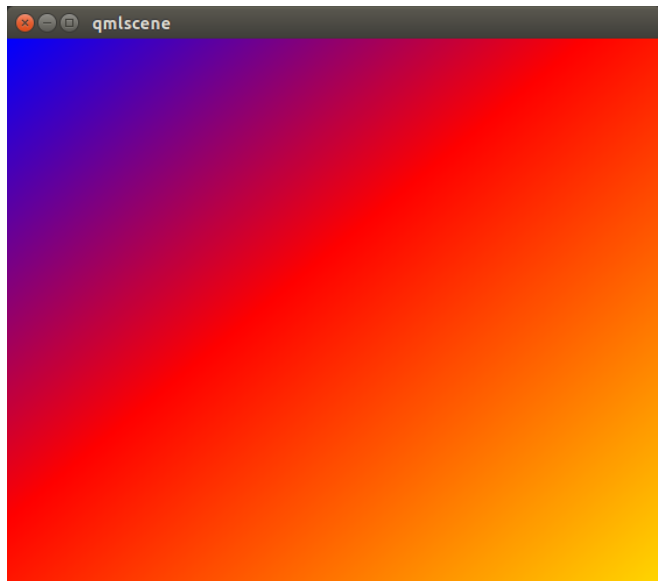
```
LinearGradient{
    anchors.fill: parent
    start: Qt.point(0,0)    // Начало градиента
    end: Qt.point(rect.width, rect.height) // конец градиента

    gradient: Gradient{
        // Список из точек градиента
        // расположение задаётся относительной координатой position
        // на прямой от start до end
        GradientStop{position: 0; color: "blue"}
        GradientStop{position: 0.4; color: "red"}
        GradientStop{position: 1; color: "yellow"} }
    }
```

Градиент изображается средствами CPU, не GPU поэтому его стоит использовать как статический элемент, чтобы не расходовать много процессорного времени.

Помимо использования отдельного компонента для градиента (например LinearGradient) свойство gradient можно задавать для различных элементов интерфейса. Однако такой градиент всегда вертикальный.

## Градиент. Пример



# Анимация

- ▶ Анимация является отдельным компонентом
- ▶ Анимация обычно включает в себя набор базовых анимаций, которые изменяют одно или несколько свойств конкретного объекта.

Например базовая анимация `NumberAnimation` может задавать изменение одного числового свойства объекта.

- ▶ Базовые анимации могут выполняться последовательно или параллельно внутри соответствующих компонентов
  - ▶ **`ParallelAnimation`**
  - ▶ **`SequentialAnimation`**
- ▶ Для запуска анимации нужно задать её свойство *running*  
`my_anim.running = true`

<http://doc.qt.io/qt-5/qtquick-usecase-animations.html>

# Анимация. Пример

Одновременное изменение ширины и высоты прямоугольника

```
ParallelAnimation {  
    id: my_anim  
  
    // изменение ширины компонента my_rect  
    // от 50 до 1000 пикселей  
    // в течении 350 миллисекунд  
    // easing.type задаёт функцию, согласно которой  
    // будет изменяться целевой параметр (width)  
    NumberAnimation{target: my_rect; property: "width";  
        easing.type: Easing.InCubic  
        from: 50; to: 1000; duration: 350 }  
  
    NumberAnimation{target: my_rect; property: "height";  
        easing.type: Easing.InCubic  
        from: 50; to: 1000; duration: 350 }  
  
    // дополнительно можно описать действия выполняемые после  
    // завершения анимации  
    onStopped: {my_rect.color = "red";}  
}
```

# Некоторые обработчики событий

Обработчики событий описываются подобно простым свойствам объекта, только значения свойства - код на Java Script

- ▶ **Component.onCompleted** - обработчик запускаемый при создании объекта интерфейса
- ▶ **onClicked** (для `MouseArea`<sup>5</sup>)
- ▶ **onEditingFinished** - закончено редактирование поля ввода
- ▶ **onValueChanged** - изменено значение `SpinBox`
- ▶ **onPressed** - обработчик нажатия кнопки

---

<sup>5</sup> невидимый объект, который может получать события мыши



# Обработчики событий. Пример

```
import QtQuick 2.0

Rectangle {
    id: rect
    width: 100; height: 100

    MouseArea {
        anchors.fill: parent
        onClicked: {
            rect.color =
                Qt.rgb(Math.random(), Math.random(), Math.random(), 1)
        }
    }
}
```

[doc.qt.io/qt-5/qtqml-syntax-signals.html](http://doc.qt.io/qt-5/qtqml-syntax-signals.html)

## Обработка событий. Пример

Для описания реакции на нажатие клавиши предусмотрены специальные обработчики:

```
Keys.onUpPressed: rectangle.y -= 10  
Keys.onDownPressed: rectangle.y += 10  
Keys.onLeftPressed: rectangle.x += 10  
Keys.onRightPressed: rectangle.x -= 10
```

```
focus: true
```

Чтобы компонент с обработчиками перехватил нажатие клавиши он должен находится в фокусе ввода.

[doc.qt.io/qt-5/qtquick-usecase-userinput.html](http://doc.qt.io/qt-5/qtquick-usecase-userinput.html)

# Использование JavaScript

Для работы с математическими функциями используется модуль **Math**.

Этот модуль подключается автоматически.

для преобразования числа в строку используются *методы*

- ▶ `toString`
- ▶ `toFixed( число-десятичных-знаков )`

List of JavaScript Objects and Functions

<http://doc.qt.io/qt-5/qtqml-javascript-expressions.html>

# Использование JavaScript

для преобразования строки в число используются *функции*

- ▶ `parseInt(string, radix);`
- ▶ `parseFloat(string);`

# Использование JavaScript

- ▶ `console.log("hello")` - вывод в консоль (используется для отладки)

В QML можно загружать JavaScript файлы, которые могут содержать сложную логику работы UI. [Например D3.js](#)

## Переменные и функции

Если есть необходимость использовать глобальную переменную в QML её можно описать как свойство некоторого компонента:

```
property var aNumber: 100
```

Определение новых функций не отличается от такового в JavaScript

```
function factorial(a) {  
    a = parseInt(a);  
    if (a <= 0)  
        return 1;  
    else  
        return a * factorial(a - 1);  
}
```

[doc.qt.io/qt-5/qml-var.html](http://doc.qt.io/qt-5/qml-var.html)

[doc.qt.io/qt-5/qtqml-javascript-expressions.html](http://doc.qt.io/qt-5/qtqml-javascript-expressions.html)

# Взаимодействие C++ и QML

- ▶ Программист не реализует собственный класс представляющий главное окно приложения.
- ▶ Вместо этого задача программиста - создать набор классов с бизнес-логикой, которые будут подключены в QML файл и использованы там.
- ▶ Проектировщик интерфейса использует классы с бизнес логикой так, как будто это стандартные компоненты QML
- ▶ События генерируемые пользователем привязываются к классам с бизнес логикой.

см. пример [github.com/VetrovSV/OOP/tree/master/example\\_qml](https://github.com/VetrovSV/OOP/tree/master/example_qml)

# Взаимодействие C++ и QML

Перед использование C++ класса (производного от QObject) его применение нужно явно обозначить:

Программа интерфейс который описан с помощью QML будет выглядеть так:

```
...  
// сделать класс Calc доступным в QML коде  
qmlRegisterType<Calc>("calc", 1, 0, "Calc");  
// calc - имя модуля  
// 1,0 - версия модуля  
// Calc имя типа  
  
// Загрузка и разбор qml файла  
QQmlApplicationEngine engine;  
...
```



# Outline

Прошлые темы

Язык описания UI  
QML

Ссылки

## Документация:

- ▶ [Qt Documentation: QML](#)
- ▶ [Integrating QML and C++](#)

## Примеры и обзоры:

- ▶ [youtube: Вебинар по QML и QtQuick: часть первая](#)
- ▶ [youtube: Introduction to Qt – Intro to QML \(tutorial\)](#)
- ▶ [Начинаем работу Python + Qt5 + QML](#)
- ▶ [youtube: Сергей Хомяков, QMLQuick на практике](#)

# Материалы курса

Слайды, вопросы к экзамену, задания, примеры

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