

## СЛАЙД 1

### РАЗРАБОТКА ПРИЛОЖЕНИЙ НА C++

#### Раздел 1. Основы Qt Quick

#### Тема № 3. Декларативный язык программирования QML.

## СЛАЙД 2

Занятие № 3/1

### ЛЕКЦИЯ № 3

#### Основы языка QML

1. Знакомство языком QML.	
2. Основные визуальные типы.	
3. Дорога к зачёту.	

## СЛАЙД 3

### ПЕРВЫЙ УЧЕБНЫЙ ВОПРОС. ЗНАКОМСТВО ЯЗЫКОМ QML.

#### 1.1. Состав языка QML

Модуль Qt QML определяет и реализует инфраструктуру языка и механизма, а также предоставляет API, позволяющий разработчикам приложений расширять язык QML с помощью пользовательских типов и интегрировать код QML с JavaScript и C++. Модуль Qt QML предоставляет как QML API и C++ API.

Обратите внимание, что, хотя модуль Qt QML предоставляет язык и инфраструктуру для приложений QML. Модуль Qt Quick предоставляет множество визуальных компонентов, поддержку представления модели, структуру анимации и многое другое для создания пользовательских интерфейсов.

Типы QML в Qt QML доступны через QtQml импорт. Чтобы использовать типы, добавьте в файл. qml следующий оператор импорта:

```
import QML
```

Чтобы связать модуль, добавьте эту строку в свой qmake.pro файл:

```
QT += qml
```

## СЛАЙД 4

### QML и типы QML

Модуль Qt QML содержит:

- структуру QML;
- важные типы QML, используемые в приложениях.

В дополнении к базовым типам QML, модуль поставляется со следующими типами объектов QML:

- Component;
- QObject;
- Binding;
- Connections;
- Timer;

О них мы будем говорить в следующих лекциях. А сейчас рассмотрим базовые типы данных.

## СЛАЙД 5

### 1.2. Базовые типы

QML поддерживает ряд базовых типов.

Базовый тип – это тип, который ссылается на простое значение, такое как **int** или **string**.

Это контрастирует с типами объектов QML, которые относятся к объекту со свойствами, сигналами, методами и так далее.

В отличие от ТИПА ОБЪЕКТА, БАЗОВЫЙ ТИП нельзя использовать для объявления объектов QML: невозможно, например, объявить объект **int{}** или объект **size{}**.

Базовые типы могут использоваться для обозначения:

- одного значения (например, **int** относится к одному числу, **var** может относиться к одному списку элементов);
- значения, содержащего простой набор пар «свойство-значение» (например, **size** относится к значению с атрибутами **width** и **height**).

Когда переменная или свойство имеют базовый тип и присваиваются другой переменной или свойству, создается копия значения. В JavaScript это значение называется примитивным значением.

## СЛАЙД 6

### Поддерживаемые базовые типы

Некоторые базовые типы поддерживаются движком по умолчанию и не требуют использования оператора импорта, в то время как другие требуют, чтобы клиент импортировал модуль, который их предоставляет.

Все базовые типы, перечисленные ниже, могут использоваться в качестве типа свойства в документе QML со следующими исключениями:

- list** (список) должен использоваться в сочетании с типом объекта QML;
- enumeration** (перечисление) нельзя использовать напрямую, так как перечисление должно быть определено зарегистрированным типом объекта QML.

## СЛАЙД 7

### Базовые типы, предоставляемые языком QML

Базовые типы, изначально поддерживаемые в языке QML, перечислены ниже:

<b>bool</b>	Двоичное значение true/false
<b>double</b>	Число с десятичной запятой, сохранённое с двойной точностью
<b>enumeration</b>	Именованное значение перечисления
<b>int</b>	Целое число, например, 0, 1 или -30
<b>list</b>	Список объектов в QML
<b>real</b>	Число с десятичной запятой
<b>string</b>	Текстовая строка произвольной формы
<b>url<sup>1</sup></b>	Определитель положения ресурса
<b>var</b>	Обобщённый тип свойства

## СЛАЙД 8

### Базовые типы, предоставляемые модулями QML

Модули QML (QtQuick<sup>2</sup> и Silica<sup>3</sup>) могут расширять язык QML дополнительными базовыми типами.

<sup>1</sup> url – унифицированный указатель ресурса – это адрес ресурса, размещенного в Интернете. Теоретически, каждый URL указывает на уникальное местоположение в Интернете. Это могут быть веб-адреса, CSS-файлы, изображения или другие медиа

<sup>2</sup> QtQuick – стандартная библиотека для написания QML-приложений.

<sup>3</sup> Silica –предоставляет дополнительные типы, предназначенные для создания приложений с внешним видом, поведением и уникальными возможностями, соответствующими стилю стандартных приложений ОС Аврора.

Например, базовые типы, предоставляемые модулем QtQuick, перечислены ниже:

<b>color</b>	Значение цвета
<b>date</b>	Значение даты
<b>font</b>	Значение шрифта
<b>matrix4x4</b>	Матрица с 4 строками и 4 столбцами
<b>point</b>	Значение с атрибутами x и y
<b>quaternion</b>	Тип кватерниона <sup>4</sup> со скалярными атрибутами x и y
<b>rect</b>	Значение с атрибутами x и y, то есть, ширины и высоты
<b>size</b>	Значение с атрибутами ширины и высоты
<b>vector2D</b>	Тип Vektor2D имеет атрибуты x и y
<b>vector3D</b>	Тип Vektor3D имеет атрибуты x, y и z
<b>vector4D</b>	Тип Vektor3D имеет атрибуты x, y, z и w

Глобальный объект Qt предоставляет полезные функции для управления значениями базовых типов.

**Import Statement: `import QtQml`**

Объект Qt является глобальным объектом со служебными функциями, свойствами и перечислениями.

Он не является экземпляром. Чтобы использовать его, необходимо вызвать члены глобального объекта Qt напрямую. Например:

```
import QtQuick 2.0

Text {
    color: Qt.rgb(1, 0, 0, 1)
    text: Qt.md5("hello, world")
}
```

В настоящее время только модули QML, предоставляемые Qt, могут предоставлять свои собственные базовые типы, однако это может измениться в будущих версиях Qt QML. Чтобы использовать типы, предоставляемые определенным модулем QML, клиенты должны импортировать этот модуль в свои документы QML.

## СЛАЙД 9

### Поведение при изменении свойства для базовых типов

Некоторые базовые типы имеют свойства: например, тип **font** имеет свойства **pixelSize**, **family** и **bold**. В отличие от свойств типов объектов, свойства базовых типов не предоставляют собственных сигналов об изменении. Создать обработчик сигнала изменения свойства можно только для самого свойства базового типа:

<sup>4</sup> Кватернион – это система гиперкомплексных чисел, образующая векторное пространство размерностью четыре над полем вещественных чисел. являются альтернативой методам матрицы, которые обычно используются для трехмерных поворотов. Кватернион представляет ось в трехмерном пространстве и поворот вокруг этой оси.

```

Text {
    // недопустимо!
    onFont.pixelSizeChanged: doSomething()

    // тоже недопустимо!
    font {
        onPixelSizeChanged: doSomething()
    }

    // а это нормально
    onFontChanged: doSomething()
}

```

Однако имейте в виду, что сигнал изменения свойства для базового типа выдается всякий раз, когда изменяется *любой* из его атрибутов, а также когда изменяется само свойство. Возьмем, к примеру, следующий код:

```

Text {
    onFontChanged: console.log("font changed")

    Text { id: otherText }

    focus: true

    // изменение любого из атрибутов font или присваивание свойству
    // другого значения font вызовет обработчик onFontChanged
    Keys.onDigit1Pressed: font.pixelSize += 1
    Keys.onDigit2Pressed: font.b = !font.b
    Keys.onDigit3Pressed: font = otherText.font
}

```

Свойства объектного типа, наоборот, генерируют свои собственные сигналы об изменении, а обработчик сигнала изменения свойства для свойства объектного типа вызывается только тогда, когда свойству присваивается значение другого объекта.

## СЛАЙД 10

### 1.3. Структура QML-документа

Документ QML – это строка, соответствующая синтаксису документа QML. Документ определяет тип объекта QML. Документ обычно загружается из ".qml" файла, хранящегося локально или удаленно, но может быть создан вручную в коде.

#### Структура документа QML

Документ QML состоит из двух разделов: раздела импорта и раздела объявления объекта. Раздел импорта в документе содержит операторы импорта, которые определяют, какие типы объектов QML и ресурсы JavaScript может использовать документ. Раздел объявления объекта определяет дерево объектов, которое будет создано при создании экземпляра типа объекта, определенного в документе.

Пример простого документа выглядит следующим образом:

```
import QtQuick 2.0

Rectangle {
    width: 300
    height: 200
    color: "blue"
}
```

## СЛАЙД 11

### Синтаксис документа QML

В разделе объявления объекта документа должна быть указана допустимая иерархия объектов с соответствующими QML синтаксис.

Объявление объекта может включать спецификацию пользовательского объекта атрибута.

Атрибуты метода объекта могут быть указаны как функции JavaScript, а атрибуты свойств объекта могут быть назначены свойством binding выражением.

## СЛАЙД 12

### Определение типов объектов с помощью документов QML

Как кратко описано в предыдущем разделе, документ неявно определяет тип объекта QML. Одним из основных принципов QML является возможность определять, а затем повторно использовать типы объектов. Это повышает удобство сопровождения кода QML, повышает удобочитаемость объявлений иерархии объектов и способствует разделению определения пользовательского интерфейса и реализации логики.

В следующем примере разработчик клиента определяет Button тип с документом в файле:

```
// Button.qml
import QtQuick 2.0

Rectangle {
    width: 100; height: 100
    color: "red"

    MouseArea {
        anchors.fill: parent
        onClicked: console.log("Button clicked!")
    }
}
```

Затем этот Button тип можно использовать в приложении:

```
// application.qml
import QtQuick 2.0

Column {
    Button { width: 50; height: 50 }
    Button { x: 50; width: 100; height: 50; color: "blue" }
    Button { width: 50; height: 50; radius: 8 }
}
```

## СЛАЙД 13

### Загрузка ресурсов и прозрачность сети

Важно отметить, что QML прозрачен для сети.

Приложения могут импортировать документы из удаленных путей так же просто, как и документы из локальных путей.

На самом деле, любому url свойству может быть назначен удаленный или локальный URL-адрес, а механизм QML будет обрабатывать любые задействованные сетевые подключения.

## СЛАЙД 14

### 1.4. Элементы QML-объекта

QML элемент, так же, как и элемент Qt, представляет собой совокупность блоков: графических (таких, как rectangle, image) и поведенческих (таких, как state, transition, animation). Эти элементы могут быть объединены, чтобы построить комплексные компоненты, начиная от простых кнопок и ползунков и заканчивая полноценными приложениями, работающими с интернетом.

QML элементы могут быть дополнены стандартными для JavaScript вставками путём встраивания .js файлов. Также они могут быть расширены C++ компонентами через Qt framework.

Элементы можно разделить на ВИЗУАЛЬНЫЕ и НЕВИЗУАЛЬНЫЕ.

ВИЗУАЛЬНЫЙ элемент (например, Rectangle, прямоугольник) имеет геометрическую форму и обычно представляет собой область на экране.

НЕВИЗУАЛЬНЫЙ элемент (например, таймер) обеспечивает общую функциональность, обычно используемую для управления визуальными элементами.

Рассмотрим НЕВИЗУАЛЬНЫЕ элементы.

## СЛАЙД 15

Существует ряд элементов QML, используемых для позиционирования объектов. Они называются ПОЗИЦИОНЕРАМИ, из которых модуль Qt Quick предоставляет следующее: Row, Column, Grid и Flow.

Рассмотрим некоторые из них.

Прежде чем углубиться в детали, рассмотрим некоторые вспомогательные элементы: красный, синий, зеленый, светлый и темный квадраты.

Каждый из этих компонентов содержит закрашенный прямоугольник размером 48x48 пикселей.

В качестве примера рассмотрим код RedSquare:

```
// RedSquare.qml

import QtQuick

Rectangle {
    width: 48
    height: 48
    color: "#ea7025"
    border.color: Qt.lighter(color)
}
```

Стоит обратить внимание на использование Qt.lighter(color) для получения более светлого цвета границы на основе цвета заливки. Мы будем использовать эти вспомогательные компоненты в следующих примерах, чтобы сделать исходный код более компактным и читабельным.

Стоит помнить, что каждый прямоугольник изначально имеет размер 48x48 пикселей.

## СЛАЙД 16

### 1.4.1. Элемент Column

**Column** — это тип, который размещает свои дочерние элементы вдоль одного столбца. Его можно использовать как удобный способ вертикального позиционирования ряда элементов без использования якорей.

Ниже приведен столбец, содержащий три прямоугольника различных размеров:

```
Column {
    spacing: 2
    Rectangle { color: "red"; width: 50; height: 50 }
    Rectangle { color: "green"; width: 20; height: 50 }
    Rectangle { color: "blue"; width: 50; height: 20 }
}
```

**Column** автоматически размещает эти элементы в вертикальном порядке, вот так:





Свойство **spacing** можно использовать для изменения расстояния между дочерними элементами.

Если элемент в столбце не виден или если его ширина или высота равны 0, элемент не будет размещен и не будет виден внутри столбца.

Кроме того, поскольку `column` автоматически позиционирует свои дочерние элементы по вертикали, дочерний элемент внутри столбца не должен устанавливать свое положение `y` или привязывать себя по вертикали, используя привязки `top`, `bottom`, `anchors.verticalCenter`, `fill` или `centerIn`.

Если вам необходимо выполнить эти действия, рассмотрите возможность размещения элементов без использования столбца.

Обратите внимание, что элементы в столбце могут использовать свойство `Positioner attached` для доступа к дополнительной информации о своем положении в столбце.

## СЛАЙД 17

Рассмотрим некоторые свойства элемента **Column**.

Свойства:

- add : Transition;***
- bottomPadding : real;***
- leftPadding : real;***
- move : Transition;***
- padding : real;***
- populate : Transition;***
- rightPadding : real;***
- spacing : real;***
- topPadding : real.***

Такие свойства, как :

- bottomPadding : real;***
- leftPadding : real ;***
- padding : real;***
- rightPadding : real;***
- topPadding : real.***

удерживают отступы вокруг содержимого.

Рассмотрим остальные свойства.

## СЛАЙД 18

### Свойство **add : Transition**

Это свойство содержит переход, который должен выполняться для элементов, добавленных в этот позиционер.

Для позиционера это относится к:

- элементы, которые создаются или повторно отображаются как дочерние элементы позиционера после того, как позиционер был создан;

- дочерние элементы, которые меняют свое свойство **Item::visible** с **false** на **true** и, таким образом, теперь видны .

## Свойство **move** : **Transition**

Это свойство позволяет выполнить переход для элементов, которые переместились внутри позиционера. Для позиционера это относится к:

- дочерние элементы, которые перемещаются при их перемещении из-за добавления, удаления или перестановки других элементов в позиционере;
- дочерние элементы, которые перемещаются из-за изменения размера других элементов в позиционере.

### СЛАЙД 19

## Свойство **populate** : **Transition**

Это свойство содержит переход, который должен выполняться для элементов, являющихся частью этого позиционера на момент его создания. Переход выполняется при первом создании позиционера.

## Свойство **spacing** : **real**

**Spacing**- это величина в пикселях, оставленная пустой между соседними элементами.

Интервал по умолчанию равен 0.

### СЛАЙД 20

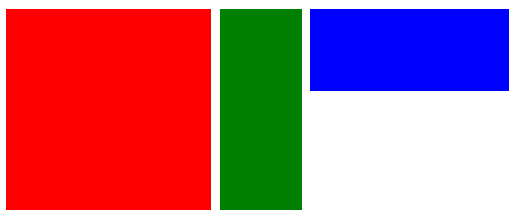
## 1.4.2. Элемент **Row**

**Row** - это тип, который позиционирует дочерние элементы вдоль одного ряда. Он может использоваться как удобный способ горизонтального позиционирования ряда элементов без использования якорей.

Ниже показан **Row**, содержащий три прямоугольника разного размера:

```
import QtQuick 2.0
Row {
    spacing: 2
    Rectangle { color: "red"; width: 50; height: 50 }
    Rectangle { color: "green"; width: 20; height: 50 }
    Rectangle { color: "blue"; width: 50; height: 20 }
}
```

**Row** автоматически размещает эти элементы в горизонтальном порядке, как показано ниже:



Если элемент в ряду не виден, или если его ширина или высота равны 0, элемент не будет выложен и не будет виден в ряду. Кроме того, поскольку ряд автоматически позиционирует свои дочерние элементы горизонтально, дочерний элемент в ряду не должен устанавливать свою позицию x или горизонтально привязывать себя с помощью

якорей `left`, `right`, `anchors.horizontalCenter`, `fill` или `centerIn`. Если вам необходимо выполнить эти действия, рассмотрите возможность позиционирования элементов без использования `Row`.

## СЛАЙД 21

Рассмотрим некоторые свойства элемента **Row**.

Свойства:

- bottomPadding** : **real**;
- leftPadding** : **real**;
- padding** : **real**;
- rightPadding** : **real**;
- topPadding** : **real**.

Эти свойства определяют подложку вокруг содержимого.

Рассмотрим остальные свойства.

### Свойство **add** : **Transition**

Это свойство содержит переход, который должен выполняться для элементов, добавленных в этот позиционер.

Для позиционера это относится к:

- элементы, которые создаются или переназначаются как дочерние элементы позиционера после его создания;
- дочерние элементы, которые меняют свое свойство **Item::visible** с **false** на **true** и, таким образом, становятся видимыми.

### Свойство **effectiveLayoutDirection** : **enumeration**

Это свойство определяет эффективное направление расположения строки.

При использовании присоединенного свойства **LayoutMirroring::enabled** для макетов локали визуальное направление расположения позиционера строки будет зеркально отражено. Однако свойство **layoutDirection** останется неизменным.

## СЛАЙД 22

### Свойство **layoutDirection** : **enumeration**

В этом свойстве указывается направление расположения строки.

Возможные значения:

- Qt.LeftToRight** (по умолчанию) - Элементы располагаются слева направо. Если ширина ряда задана явно, то левая привязка остается слева от ряда.
- Qt.RightToLeft** - Элементы располагаются справа налево. Если ширина строки явно задана, правая привязка остается справа от строки.

### Свойство **move** : **Transition**

Это свойство определяет переход для элементов, которые переместились в позиционере. Для позиционера это свойство применяется к:

- дочерние элементы, которые перемещаются, когда они смещаются из-за добавления, удаления или перестановки других элементов в позиционере;
- дочерние элементы, которые перемещаются из-за изменения размера других элементов в позиционере.

## СЛАЙД 23

### Свойство **populate** : **Transition**

Это свойство содержит переход, который будет выполняться для элементов, являющихся частью данного позиционера в момент его создания. Переход выполняется при первом создании позиционера.

### Свойство **spacing** : **real**

Расстояние между элементами - это количество пикселей, оставляемых пустыми между соседними элементами. По умолчанию расстояние равно 0.

## СЛАЙД 24

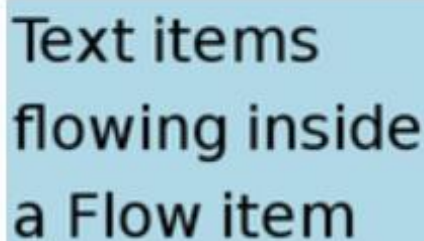
### 1.4.3. Элемент **Flow**

**Flow** - размещает свои дочерние элементы как слова на странице, обертывая их для создания строк или столбцов элементов.

Ниже приведен пример **Flow**, содержащий различные текстовые элементы:

```
Flow
{
  anchors.fill: parent
  anchors.margins: 4
  spacing: 10
  Text { text: "Text"; font.pixelSize: 40 }
  Text { text: "items"; font.pixelSize: 40 }
  Text { text: "flowing"; font.pixelSize: 40 }
  Text { text: "inside"; font.pixelSize: 40 }
  Text { text: "a"; font.pixelSize: 40 }
  Text { text: "Flow"; font.pixelSize: 40 }
  Text { text: "item"; font.pixelSize: 40 }
}
```

Flow автоматически размещает дочерние текстовые элементы рядом, перенося их по мере необходимости:



Text items  
flowing inside  
a Flow item

Если элемент в Flow не виден или если его ширина или высота равны 0, элемент не будет размещен и не будет виден в потоке. Кроме того, поскольку поток автоматически позиционирует свои дочерние элементы, дочерний элемент в потоке не должен устанавливать свои позиции x или y или привязывать себя к какому-либо из свойств привязки.

## СЛАЙД 25

Рассмотрим некоторые свойства элемента **Flow**.

Свойства:

```
→add : Transition;  
→bottomPadding : real;  
→effectiveLayoutDirection : enumeration;  
→flow : enumeration;  
→layoutDirection : enumeration;  
→leftPadding : real;  
→move : Transition;  
→padding : real;  
→populate : Transition;  
→rightPadding : real;  
→spacing : real;  
→topPadding : real.
```

Такие свойства, как:

```
→bottomPadding : real;  
→leftPadding : real;  
→padding : real;  
→rightPadding : real;  
→topPadding : real
```

удерживают отступы вокруг содержимого.

Рассмотрим остальные свойства.

## СЛАЙД 26

### Свойство **add : Transition**

Это свойство содержит переход, который должен выполняться для элементов, добавленных в этот позиционер. Для позиционера это относится к:

- элементы, которые создаются или повторно отображаются как дочерние элементы позиционера после того, как позиционер был создан;
- дочерние элементы, которые меняют свое свойство `Item::visible` с `false` на `true` и, таким образом, теперь видны.

### Свойство **effectiveLayoutDirection : enumeration**

Это свойство определяет эффективное направление компоновки потока.

При использовании прилагаемого свойства `LayoutMirroring::enabled` для локальных макетов направление визуальной компоновки позиционера сетки будет зеркально отображено. Однако свойство `layoutDirection` останется неизменным.

### Свойство **flow : enumeration**

Это свойство поддерживает поток макета.

Возможными значениями являются:

-**Flow.LeftToRight** (по умолчанию) - элементы располагаются рядом друг с другом в соответствии с **layoutDirection** до тех пор, пока не будет превышена ширина потока, затем переносятся на следующую строку.

-**Flow.TopToBottom** - элементы располагаются рядом друг с другом сверху вниз до тех пор, пока не будет превышена высота потока, затем переносятся в следующий столбец.

## СЛАЙД 27

### Свойство **layoutDirection** : enumeration

Это свойство определяет направление компоновки макета.

Возможными значениями являются:

-**Qt.LeftToRight** (по умолчанию) - элементы располагаются сверху вниз и слева направо. Направление потока зависит от свойства `Flow::flow`.

-**Qt.RightToLeft** - Элементы располагаются сверху вниз и справа налево. Направление потока зависит от свойства `Flow::flow`.

### Свойство **move** : Transition

Это свойство позволяет выполнить переход для элементов, которые переместились внутри позиционера. Для позиционера это относится к:

-Дочерние элементы, которые перемещаются при их перемещении из-за добавления, удаления или перестановки других элементов в позиционере;

-Дочерние элементы, которые перемещаются из-за изменения размера других элементов в позиционере .

### Свойство **populate** : Transition

Это свойство содержит переход, который должен выполняться для элементов, являющихся частью этого позиционера на момент его создания. Переход выполняется при первом создании позиционера.

### Свойство **spacing** : real

Spacing - это величина в пикселях, оставленная пустой между каждым соседним элементом, и по умолчанию равна 0.

## СЛАЙД 28

### 1.4.4. Элемент Grid

**Grid** (сетка) - это тип, который позиционирует свои дочерние элементы при формировании сетки.

**Grid** создает сетку ячеек, достаточно большую, чтобы вместить все ее дочерние элементы, и размещает эти элементы в ячейках слева направо и сверху вниз. Каждый элемент расположен в верхнем левом углу своей ячейки с позицией (0, 0).

**Grid** по умолчанию состоит из четырех столбцов и создает столько строк, сколько необходимо для размещения всех ее дочерних элементов. Количество строк и столбцов можно ограничить, установив свойства строк и столбцов.

Например, ниже приведен `grid`, содержащий пять прямоугольников различных размеров:

```
import QtQuick 2.0
Grid
{
    columns: 3
```

```
spacing: 2
Rectangle { color: "red"; width: 50; height: 50 }
Rectangle { color: "green"; width: 20; height: 50 }
Rectangle { color: "blue"; width: 50; height: 20 }
Rectangle { color: "cyan"; width: 50; height: 50 }
Rectangle { color: "magenta"; width: 10; height: 10 }
}
```

Grid автоматически размещает дочерние элементы в виде сетки:



Если элемент в сетке не виден, или если его ширина или высота равны 0, элемент не будет размещен и не будет виден внутри столбца. Кроме того, поскольку сетка автоматически позиционирует свои дочерние элементы, дочерний элемент в сетке не должен устанавливать свои позиции по x или y или привязывать себя к какому-либо из свойств привязки.

## СЛАЙД 29

Рассмотрим некоторые свойства элемента **Grid**.

- add : Transition
- bottomPadding : real;
- columnSpacing : qreal;
- columns : int;
- effectiveHorizontalItemAlignment : enumeration;
- effectiveLayoutDirection : enumeration;
- flow : enumeration;
- horizontalItemAlignment : enumeration;
- layoutDirection : enumeration;
- leftPadding : real;
- move : Transition;
- padding : real;
- populate : Transition;
- rightPadding : real;
- rowSpacing : qreal;
- rows : int;
- spacing : qreal;
- topPadding : real;
- verticalItemAlignment : enumeration;

Такие свойства, как:

```
→bottomPadding : real;  
→leftPadding : real;  
→padding : real;  
→rightPadding : real;  
→topPadding : real;
```

удерживают отступы вокруг содержимого.

## СЛАЙД 30

Такие свойства, как:

```
→effectiveHorizontalItemAlignment : enumeration;  
→horizontalItemAlignment : enumeration;  
→verticalItemAlignment : enumeration;
```




задают выравнивание элементов в сетке по горизонтали и вертикали.

По умолчанию элементы выровнены по вертикали вверх. Горизонтальное выравнивание соответствует направлению расположения сетки, например, при использовании направления расположения слева направо элементы будут выровнены по левому краю.

Допустимыми значениями для **horizontalItemAlignment** являются **Grid.AlignLeft**, **Grid.AlignRight** и **Grid.AlignHCenter**.

Допустимыми значениями для **verticalItemAlignment** являются **Grid.AlignTop**, **Grid.AlignBottom** и **Grid.AlignVCenter**.

На приведенных ниже изображениях показаны три примера того, как выровнять элементы.

			
Horizontal alignment	AlignLeft	AlignHCenter	AlignHCenter
Vertical alignment	AlignTop	AlignTop	AlignVCenter

## СЛАЙД 31

### Свойство **add : Transition**

Это свойство содержит переход, который должен выполняться для элементов, добавленных в этот позиционер. Для позиционера это относится к:

- Элементы, которые создаются или повторно отображаются как дочерние элементы позиционера после того, как позиционер был создан;

- Дочерние элементы, которые меняют свое свойство `Item::visible` с `false` на `true` и, таким образом, теперь видны .

### Свойство **columnSpacing : qreal**

Это свойство определяет интервал в пикселях между столбцами.



Если это свойство не задано, то для интервала между столбцами используется `spacing`.

По умолчанию это свойство не задано.

### **Свойство `columns` : `int`**

Это свойство содержит количество столбцов в сетке. Количество столбцов по умолчанию равно 4.

Если в сетке недостаточно элементов для заполнения указанного количества столбцов, некоторые столбцы будут иметь нулевую ширину.

### **Свойство `effectiveLayoutDirection` : `enumeration`**

Это свойство определяет эффективное направление расположения сетки.

При использовании прилагаемого свойства `LayoutMirroring::enabled` для локальных макетов направление визуальной компоновки позиционера сетки будет зеркально отображено. Однако свойство `layoutDirection` останется неизменным.

## **СЛАЙД 32**

### **Свойство `flow` : `enumeration`**

Это свойство поддерживает поток макета.

Возможными значениями являются:

-**`Grid.LeftToRight`** (по умолчанию) - элементы располагаются рядом друг с другом в `layoutDirection`, затем переносятся на следующую строку.

-**`Grid.TopToBottom`** - элементы располагаются рядом друг с другом сверху вниз, затем переносятся в следующий столбец.

### **Свойство `layoutDirection` : `enumeration`**

Это свойство определяет направление компоновки макета.

Возможными значениями являются:

-**`Qt.LeftToRight`** (по умолчанию) - элементы располагаются сверху вниз и слева направо. Направление потока зависит от свойства `Grid::flow`.

-**`Qt.RightToLeft`** - элементы располагаются сверху вниз и справа налево. Направление потока зависит от свойства `Grid::flow`.

## **СЛАЙД 33**

### **Свойство `move` : `Transition`**

Это свойство позволяет выполнить переход для элементов, которые переместились внутри позиционера. Для позиционера это относится к:

-дочерние элементы, которые перемещаются при их перемещении из-за добавления, удаления или перестановки других элементов в позиционере;

-дочерние элементы, которые перемещаются из-за изменения размера других элементов в позиционере.

### **Свойство `populate` : `Transition`**

Это свойство содержит переход, который должен выполняться для элементов, являющихся частью этого позиционера на момент его создания. Переход выполняется при первом создании позиционера.

## Свойство `rowSpacing` : `qreal`

Это свойство определяет интервал в пикселях между строками.

Если это свойство не задано, то для определения расстояния между строками используется `spacing`.

По умолчанию это свойство не задано.

### СЛАЙД 34

## Свойство `rows` : `int`

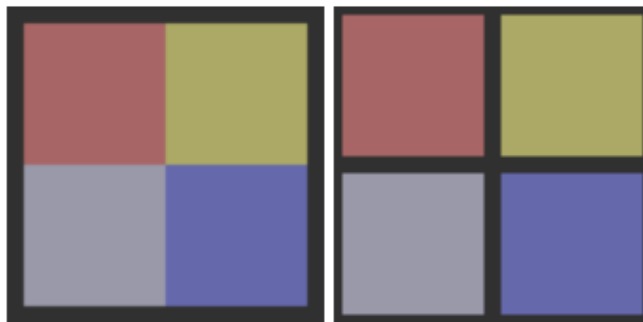
Это свойство содержит количество строк в сетке.

Если в сетке недостаточно элементов для заполнения указанного количества строк, некоторые строки будут иметь нулевую ширину.

## Свойство `spacing` : `qreal`

**Spacing** - это величина в пикселях, оставленная пустой между соседними элементами. Величина применяемого интервала будет одинаковой в горизонтальном и вертикальном направлениях. Интервал по умолчанию равен 0.

В приведенном ниже примере сетка, содержащая красный, синий и зеленый прямоугольники, размещается на сером фоне. Область, которую занимает позиционер сетки, окрашена в белый цвет. У позиционера слева интервал отсутствует (по умолчанию), а у позиционера справа интервал равен 6.



## ВТОРОЙ УЧЕБНЫЙ ВОПРОС. ОСНОВНЫЕ ВИЗУАЛЬНЫЕ ТИПЫ

### 2.1. Item как базовый тип

Item является базовым элементом для всех визуальных элементов, поскольку все остальные наследуются от Item. Он ничего не рисует сам по себе, но определяет все свойства, общие для всех визуальных элементов:

1. ГЕОМЕТРИЯ – `x` и `y` для определения положения сверху слева, `width` и `height` для размеров элемента и `z` для порядка наложения, чтобы поднимать или опускать элементы от их естественного порядка.

2. ОБРАБОТКА МАКЕТА – якоря `anchors` (слева, справа, сверху, снизу, по центру по вертикали и по горизонтали) для позиционирования элементов относительно других элементов с необязательными отступами `margins`.

3. ОБРАБОТКА КЛАВИАТУРЫ – прикрепленные свойства `Key` и `KeyNavigation` для управления обработкой нажатий клавиш и свойство `focus`, чтобы, в первую очередь, включить обработку клавиатуры.

4. ПРЕОБРАЗОВАНИЕ – преобразования масштабирования `scale` и поворота `rotate` и общее списочное свойство `transform` для преобразования `x`, `y`, `z`, а также точка `transformOrigin`.

5. ВИЗУАЛИЗАЦИЯ – `opacity` для управления прозрачностью, `visible` для отображения/скрытия элементов, `clip` для ограничения операций рисования по границе элемента и `smooth` для повышения качества рендеринга.

6. ОПРЕДЕЛЕНИЕ СОСТОЯНИЯ – списочное свойство `states` с поддерживаемым списком состояний, свойство текущего состояния `state` и списочное свойство переходов `transitions` для анимации изменений состояния.

Чтобы лучше понять различные свойства, мы попытаемся представить их при рассмотрении второго учебного вопроса в контексте представленных элементов.

Необходимо помнить, что эти основные свойства доступны для каждого визуального элемента и работают одинаково для всех этих элементов.

Элемент Item часто используется как контейнер для других элементов, подобно элементу `div` в HTML.

### 2.2. Элемент Rectangle

Rectangle расширяет Item и добавляет к нему цвет заливки. Кроме того, он поддерживает границы, определяемые свойствами `border.color` и `border.width`. Для создания прямоугольников со скругленными углами вы можете использовать свойство `radius`.

Свойства:

→ `border.color` : `color`

→ `border.width` : `int`

→ `color` : `color`

→**gradient** : **Gradient**

→**radius** : **real**

→**antialiasing**: **bool**

Каждый элемент **Rectangle** закрашивается либо сплошным цветом заливки, заданным с помощью свойства **color**, либо градиентом, заданным с помощью типа **Gradient** и установленным с помощью свойства **gradient**. Если указаны и цвет, и градиент, то используется градиент.

Вы можете добавить к прямоугольнику необязательную границу с собственным цветом и толщиной, установив свойства **border.color** и **border.width**. Установите цвет «**transparent**», чтобы нарисовать границу без цвета заливки.

Вы также можете создавать закругленные прямоугольники с помощью свойства **radius**. Поскольку при этом углы прямоугольника становятся изогнутыми, для улучшения его внешнего вида может оказаться целесообразным установить свойство **Item::antialiasing**.

Использование свойства **Item::antialiasing** улучшает внешний вид закругленного прямоугольника ценой снижения производительности рендеринга. Это свойство следует использовать, когда прямоугольник неподвижен.

По умолчанию используется значение **true** для прямоугольников с радиусом и **false** в противном случае.

## СЛАЙД 37

```
Rectangle
{
  id: rect1
  x: 12; y: 12
  width: 76; height: 96
  color: "lightsteelblue"
}
```

```
Rectangle
{
  id: rect2
  x: 112; y: 12
  width: 76; height: 96
  border.color: "lightsteelblue"
  border.width: 4
  radius: 8
}
```

```
Rectangle
{
  id: rect1
  x: 12; y: 12
  width: 176; height: 96
  gradient: Gradient
}
```

```

{
  GradientStop
  {
    position: 0.0;
    color: "lightsteelblue"
  }
  GradientStop
  {
    position: 1.0;
    color: "slategray"
  }
}
border.color: "slategray"
}

```

Градиент определяется последовательностью остановок градиента. Каждая остановка имеет позицию и цвет. Позиция отмечает положение по оси Y (0 = вверху, 1 = внизу). Также можно ставить дробные значение, например 0.33.

Цвет GradientStop отмечает цвет в этой позиции.

## СЛАЙД 38

### Свойства

**border.width : int**

**border.color : color**

Ширина и цвет, используемые для рисования границы прямоугольника.

Ширина 1 создает тонкую линию. Для отсутствия линии используйте ширину 0 или прозрачный цвет.

### Свойство **radius : real**

Это свойство содержит радиус угла, используемый для рисования закругленного прямоугольника.

Если радиус не равен нулю, прямоугольник будет нарисован как закругленный прямоугольник, в противном случае он будет нарисован как обычный прямоугольник. Один и тот же радиус используется всеми четырьмя углами; в настоящее время нет возможности указать разные радиусы для разных углов.

## СЛАЙД 39

### 2.3. Элемент Text

Для отображения текста можно использовать элемент Text.

Наиболее заметным его свойством является свойство **text** типа **string**.

Элемент вычисляет свою начальную ширину и высоту на основе заданного текста и используемого шрифта.

На шрифт можно повлиять с помощью группы свойств шрифта (например, **font.family**, **font.pixelSize**, ...).

Чтобы изменить цвет текста, просто используйте свойство **color**.

**Text**

```
{
text: "The quick brown fox"
color: "#303030"
font.family: "Ubuntu"
font.pixelSize: 28
}
```

### 2.3.1. Свойства элемента Text

Текст можно выравнивать по каждой стороне и по центру с помощью свойств **horizontalAlignment** и **verticalAlignment**.

Для дальнейшего улучшения рендеринга текста вы можете использовать свойства **style** и **styleColor**, которые позволяют вам отображать текст в контурном, приподнятом и вдавленном режиме.

Для более длинного текста часто требуется определить позицию разрыва, например, **A very ... long text**, чего можно добиться с помощью свойства **elide**. Свойство **elide** позволяет установить положение многоточия слева, справа или в середине вашего текста.

Если вы не хотите, чтобы «...» режима **elide** появлялось, и всё же хотите видеть полный текст, вы также можете обернуть текст, используя свойство **wrapMode** (работает только тогда, когда ширина задана явно):

```
Text
{
width: 40; height: 120
text: 'A very long text'
// '...' должно появиться посередине
elide: Text.ElideMiddle 7

// стиль красного утопленного текста
style: Text.Sunken
styleColor: '#FF4444'
// выравниваем текст по верхнему краю
verticalAlignment: Text.AlignTop
// имеет смысл только тогда, когда нет режима elide
// wrapMode: Text.WordWrap
}
```

Элемент **Text** отображает только заданный текст, а оставшееся пространство, которое он занимает, прозрачно. Это означает, что он не отображает никакого фоновой оформления, и поэтому вы можете предоставить фон, если хотите.

## СЛАЙД 40

Итак, далее более подробно рассмотрим некоторые свойства элемента **Text**.

Свойства:

- **effectiveHorizontalAlignment** : enumeration;
- **horizontalAlignment** : enumeration;
- **verticalAlignment** : enumeration.

позволяют устанавливать горизонтальное и вертикальное выравнивание текста по ширине и высоте элементов текста.

По умолчанию текст выравнивается по вертикали к верху.

Горизонтальное выравнивание соответствует естественному выравниванию текста, например, текст, который читается слева направо, будет выровнен слева.

Допустимыми значениями для **horizontalAlignment** являются:

-*Text.AlignLeft*;

-*Text.AlignRight*;

-*Text.AlignHCenter*;

-*Text.AlignJustify*.

Для **verticalAlignment** допустимыми значениями являются:

-*Text.AlignTop*;

-*Text.AlignBottom*;

-*Text.AlignVCenter*.

Обратите внимание, что для одной строки текста размером является площадь текста. В этом общем случае все выравнивания эквивалентны. Если вы хотите, чтобы текст был, скажем, центрирован по своему родителю, то вам нужно либо изменить **Item: anchors**, либо установить **horizontalAlignment** в **Text.AlignHCenter** и привязать ширину к ширине родителя.

## СЛАЙД 41

### Свойства:

→**bottomPadding** : real;

→**leftPadding** : real;

→**padding** : real;

→**rightPadding** : real;

→**topPadding** : real.

В этих свойствах указывается пространство вокруг содержимого. Это пространство резервируется в дополнение к **contentWidth** и **contentHeight**.

### Свойство **advance : size**

Расстояние в пикселях от базовой линии начала первого символа элемента текста до базовой линии начала первого символа элемента текста, следующего непосредственно за этим символом в потоке текста.

Обратите внимание, что опережение может быть отрицательным, если текст идет справа налево.

### Свойство **antialiasing : bool**

Используется для определения того, следует ли тексту использовать сглаживание или нет. Отключить сглаживание может только текст с **renderType** **Text.NativeRendering**.

По умолчанию используется значение **true**.

## СЛАЙД 42

### Свойство **baseUrl : url**

Это свойство определяет базовый **URL**, который используется для разрешения относительных **URL** в тексте.

Ссылки разрешаются в том же каталоге, что и цель базового **URL**, то есть любая часть пути после последнего '/' будет проигнорирована.

### Свойство **clip** : **bool**

Это свойство определяет, обрезается ли текст.

Обратите внимание, что, если текст не помещается в ограничивающий прямоугольник, он будет резко обрезан.

Если вы хотите отобразить потенциально длинный текст в ограниченном пространстве, вам, вероятно, лучше использовать **elide**.

### Свойство **color** : **color**

Пример зеленого текста, заданного с помощью шестнадцатеричной системы счисления:

```
Text
{
  color: "#00FF00"
  text: "green text"
}
```

Пример текста стального синего цвета, заданного с помощью имени цвета SVG:

```
Text
{
  color: "steelblue"
  text: "green text"
}
```

## СЛАЙД 43

### Свойство **contentHeight** : **real**

Возвращает высоту текста, включая высоту за пределами границы, потому что текста больше, чем помещается в заданную высоту.

### Свойство **contentWidth** : **real**

Возвращает ширину текста, включая ширину за пределами границы, которая покрывается из-за недостаточной обертки, если установлен режим **WrapMode**.

### Свойство **elide** : **enumeration**

Установите это свойство для выравнивания частей текста в соответствии с шириной элемента **Text**. Текст будет выравниваться только в том случае, если была задана явная ширина.

Это свойство нельзя использовать с форматированным текстом.

Выравнивание может быть:

- Text.ElideNone** – по умолчанию;
- Text.ElideLeft** – влево;
- Text.ElideMiddle** – по середине;



-Text.ElideRight – вправо.

Если для этого свойства установлено значение **Text.ElideRight**, его можно использовать с обернутым текстом. Текст будет сдвигаться только в том случае, если задано значение **maximumLineCount** или **height**. Если заданы и **maximumLineCount**, и **height**, будет применяться **maximumLineCount**, если линии не помещаются в допустимую высоту.

Если текст представляет собой строку разной длины, а режим не **Text.ElideNone**, будет использована первая подходящая строка, в противном случае будет удалена последняя.

Длинные строки упорядочиваются от самой длинной к самой короткой, разделяясь символом Unicode "String Terminator" U009C (запишите это в QML как "\u009C" или "\x9C").

## СЛАЙД 44

### Свойство **font.bold** : **bool**

Устанавливает, является ли текст полужирным.

### Свойство **font.capitalization** : **enumeration**

-Font.MixedCase – это обычный вариант рендеринга текста, при котором изменение капитализации не применяется.

-Font.AllUppercase - изменяет текст для отображения в верхнем регистре.

-Font.AllLowercase – изменяет отображение текста в нижнем регистре.

-Font.SmallCaps – изменяет отображение текста в виде мелкого кегля.

-Font.Capitalize – изменяет отображение текста так, чтобы первый символ каждого слова был заглавным.

### Свойство **font.family** : **string**

Устанавливает стиль шрифта.

### Свойство **font.italic** : **bool**

Устанавливает, имеет ли шрифт курсивный стиль.

## СЛАЙД 45

### Свойство **font.letterSpacing** : **real**

Устанавливает расстояние между буквами для шрифта.

Расстояние между буквами изменяет расстояние между отдельными буквами в шрифте по умолчанию. Положительное значение увеличивает расстояние между буквами на соответствующие пиксели; отрицательное значение уменьшает расстояние.

### Свойство **font.pixelSize** : **int**

Устанавливает размер шрифта в пикселях.

Использование этой функции делает шрифт зависимым от устройства. Используйте **pointSize** для установки размера шрифта независимым от устройства способом.

### Свойство **font.strikeout** : **bool**

Устанавливает, имеет ли шрифт стиль зачеркивания.

### Свойство **font.underline** : **bool**

Устанавливает, будет ли текст подчеркнут.

## Свойство `font.wordSpacing` : `real`

Устанавливает интервал между словами для шрифта.

Интервал между словами изменяет интервал между отдельными словами по умолчанию. Положительное значение увеличивает интервал между словами на соответствующее количество пикселей, а отрицательное значение, соответственно, уменьшает межсловный интервал.

### СЛАЙД 46

## 2.4. Элемент `Image`

Элемент `Image` может отображать изображения в различных форматах (например, PNG, JPG, GIF, BMP, WEBP).

Полный список поддерживаемых форматов изображений смотрите в документации Qt.

Помимо свойства `source` для предоставления URL-адреса изображения, он содержит свойство `fillMode`, которое управляет поведением изменения размера.

```
Image
{
  x: 12; y: 12
  source: "assets/triangle_red.png"
}
```

```
Image
{
  x: 12+64+12; y: 12
  height: 72/2
  source: "assets/triangle_red.png"
  fillMode: Image.PreserveAspectCrop
  clip: true
}
```

### СЛАЙД 47

## 2.4.1. Свойства элемента `Image`

Свойства:

**`paintedHeight` : `real`**

**`paintedWidth` : `real`**

Эти свойства определяют размер изображения, которое фактически закрашивается.

В большинстве случаев он такой же, как ширина и высота, но при использовании `Image.PreserveAspectFit` или `Image.PreserveAspectCrop` `paintedWidth` или `paintedHeight` могут быть меньше или больше, чем ширина и высота элемента `Image`.

Свойства:

**`currentFrame` : `int`**

**`frameCount` : `int`**

**`currentFrame`** – это кадр, который виден в данный момент. По умолчанию 0. Вы можете установить его в число от 0 до `frameCount` - 1, чтобы отобразить другой кадр,

если изображение содержит несколько кадров.

**frameCount** – количество кадров в изображении. Большинство изображений имеют только один кадр.

## СЛАЙД 48

### Свойство **cache** : **bool**

Указывает, следует ли кэшировать изображение. По умолчанию используется значение true. Установка значения cache в false полезна при работе с большими изображениями, чтобы убедиться, что они не будут кэшироваться за счет маленьких изображений "ui element".

### Свойство **fillMode** : **enumeration**

Установите это свойство, чтобы определить, что происходит, когда исходное изображение имеет размер, отличный от размера элемента.

-Image.Stretch – изображение масштабируется до нужного размера

-Image.PreserveAspectFit – изображение равномерно масштабируется для подгонки без обрезки

-Image.PreserveAspectCrop – изображение равномерно масштабируется для заполнения, при необходимости обрезается.

-Image.Tile – изображение дублируется по горизонтали и вертикали

-Image.TileVertically – изображение растягивается по горизонтали и выкладывается вертикальной плиткой

-Image.TileHorizontally – изображение растягивается по вертикали и выкладывается горизонтальной плиткой

-Image.Pad – изображение не преобразуется

## СЛАЙД 49

### 2.5. Элемент Button

Кнопка представляет собой кнопочный элемент управления, который пользователь может нажать. Кнопки обычно используются для выполнения действия или ответа на вопрос.

Тип Button реализует оформленную в стиле ОС Аврора кнопку с текстовой меткой.

Ниже приведён пример кода, который реализует простую кнопку:

```
import QtQuick 2.2
import Sailfish.Silica 1.0

Button
{
    text: "Нажми меня"
    onClicked: console.log("clicked!")
}
```

Button наследуется от типа QtQuick MouseArea. Поэтому свойства, сигналы и методы MouseArea (например, onClicked в примере выше) также доступны для объектов Button. Полный список доступных свойств, сигналов и методов приведён в

документации `MouseArea`.

`MouseArea` является невидимым объектом, который обычно используется в сочетании с видимым объектом для обеспечения обработки мыши для того объекта.

При назначении ширины кнопки рекомендуется вместо свойства `width` использовать свойство `preferredWidth`. Это позволит автоматически увеличить размер кнопки для вмещения текста.

Если для объекта типа `Button` установлено значение высоты, то визуально кнопка сохраняет размер, увеличивается лишь область нажатия.

## СЛАЙД 50

### 2.5.1. Свойства элемента `Button`

#### Свойство `color` : `color`

С помощью свойства `color` задаётся цвет кнопки и размещённого на ней текста.

#### Свойство `down` : `bool`

Истинно, пока кнопка нажата, и положение касания остаётся в области этой кнопки.

#### Свойство `highlightBackgroundColor` : `color`

При истинном значении свойства `down` цвет кнопки задаётся с помощью свойства `highlightBackgroundColor`.

#### Свойство `highlightColor` : `color`

При истинном значении свойства `down` цвет текста кнопки задаётся с помощью свойства `highlightColor`.

#### Свойство `icon` : `Image`

Содержит отображаемое на кнопке-переключателе изображение. Если устанавливается только иконка, то `preferredWidth` кнопки изменится на `Theme.buttonWidthTiny`. Не рекомендуется использовать кнопки только с иконками, кнопки только с текстом, и кнопки с текстом и иконкой в одном и том же макете.

## СЛАЙД 51

#### Свойство `layoutDirection` : `enumeration`

Этот параметр управляет положением текста и значка внутри кнопки относительно друг друга.

-`Qt.LeftToRight` (значением по умолчанию) - сначала отображается иконка. Текст располагается справа от иконки.

-`Qt.RightToLeft` - зеркальное отображение элементов относительно значения по умолчанию. Иконка располагается справа от текста.

Если кнопка с текстом и значком расположена внутри макета, который по своей структуре является столбцом, то содержимое внутри кнопки выравнивается по левому краю самого большого по ширине элемента. В противном случае используется выравнивание по центру.

#### Свойство `preferredWidth` : `real`

Содержит размер кнопки. Данное свойство может принимать следующие значения:

-`Theme.buttonWidthTiny` — размер по умолчанию для квадратных кнопок, состоящих только из иконок.

-Theme.buttonWidthExtraSmall — значение ширины самой маленькой кнопки в интерфейсе ОС Аврора, которое используется только в случае плотной компоновки кнопок. На экране с портретной ориентацией могут быть размещены три такие кнопки. В большинстве приложений следует избегать использования данного варианта.

-Theme.buttonWidthSmall — значение ширины маленькой кнопки, которое используется по умолчанию. Данное значение подобрано так, что на экране с портретной ориентацией могут быть размещены две такие кнопки. На больших экранах или на экранах с альбомной ориентацией может быть размещено больше двух маленьких кнопок.

-Theme.buttonWidthMedium — значение ширины средней кнопки. На экране (в зависимости от его размера и ориентации), как правило, может быть размещена только одна средняя кнопка.

-Theme.buttonWidthLarge — значение ширины большой кнопки. На экране (в зависимости от его размера и ориентации), как правило, может быть размещена только одна большая кнопка.

В случае, если заданный текст не вмещается в указанный размер кнопки, ее ширина будет автоматически увеличена, чтобы вместить текст.

### Свойство **text** : **string**

Содержит отображаемый на кнопке текст.

## СЛАЙД 52

### 2.6. Элемент **Component**

Компоненты – это повторно используемые инкапсулированные типы QML с четко определенными интерфейсами.

Компоненты часто определяются файлами компонентов, то есть файлами .qml. Тип компонента, по сути, позволяет определять компоненты QML встроено, внутри документа QML, а не как отдельный файл QML. Это может быть полезно для повторного использования небольшого компонента в файле QML или для определения компонента, который логически связан с другими компонентами QML в файле.

Например, вот компонент, который используется несколькими объектами Loader. Он содержит один элемент, прямоугольник:

```
import QtQuick 2.0
Item {
    width: 100; height: 100
    Component {
        id: redSquare
        Rectangle {
            color: "red"
            width: 10
            height: 10
        }
    }
    Loader { sourceComponent: redSquare }
```

```
Loader { sourceComponent: redSquare; x: 20 }  
}
```

Обратите внимание, что, хотя прямоугольник сам по себе будет автоматически отрисован и отображен на экране, это не относится к вышеупомянутому прямоугольнику, поскольку он определен внутри компонента. Компонент инкапсулирует типы QML внутри, как если бы они были определены в отдельном файле QML, и не загружается до тех пор, пока не будет запрошен (в данном случае двумя объектами Loader). Поскольку компонент не является производным от элемента, вы не можете привязать к нему что-либо.

Определение компонента аналогично определению документа QML. Документ QML содержит единственный элемент верхнего уровня, который определяет поведение и свойства этого компонента, и не может определять свойства или поведение вне этого элемента верхнего уровня. Аналогичным образом, определение компонента содержит единственный элемент верхнего уровня (который в приведенном выше примере является прямоугольником) и не может определять какие-либо данные за пределами этого элемента, за исключением идентификатора (которым в приведенном выше примере является redSquare).

Тип компонента обычно используется для предоставления графических компонентов для представлений. Например, свойство ListView::delegate требует, чтобы компонент указывал, как должен отображаться каждый элемент списка.

**ТРЕТИЙ УЧЕБНЫЙ ВОПРОС.  
ДОРОГА К ЗАЧЁТУ**