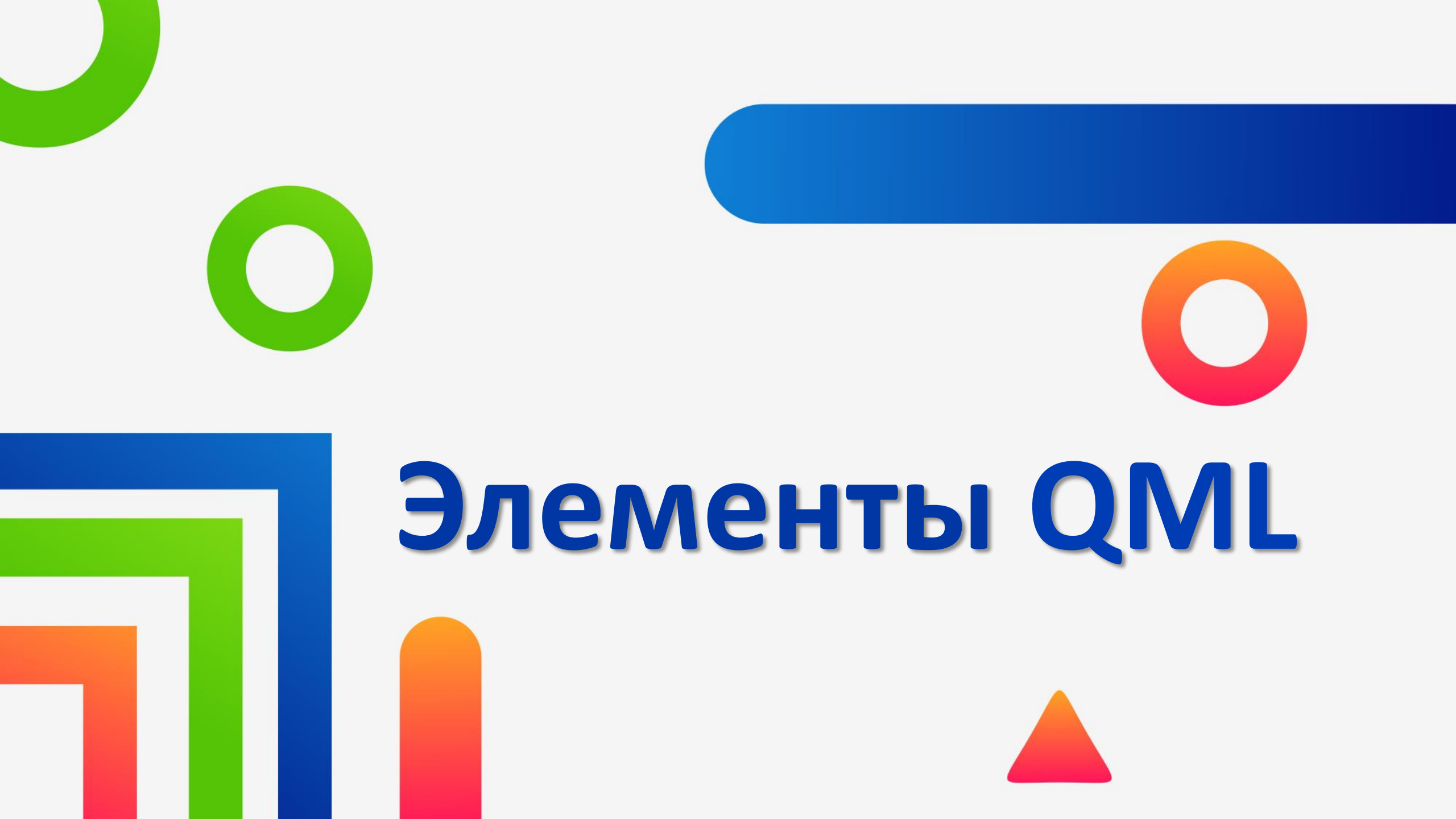

Элементы QML

Общие понятия

QML-элементы – «строительный материал» GUI приложений на QML.

Визуальные элементы

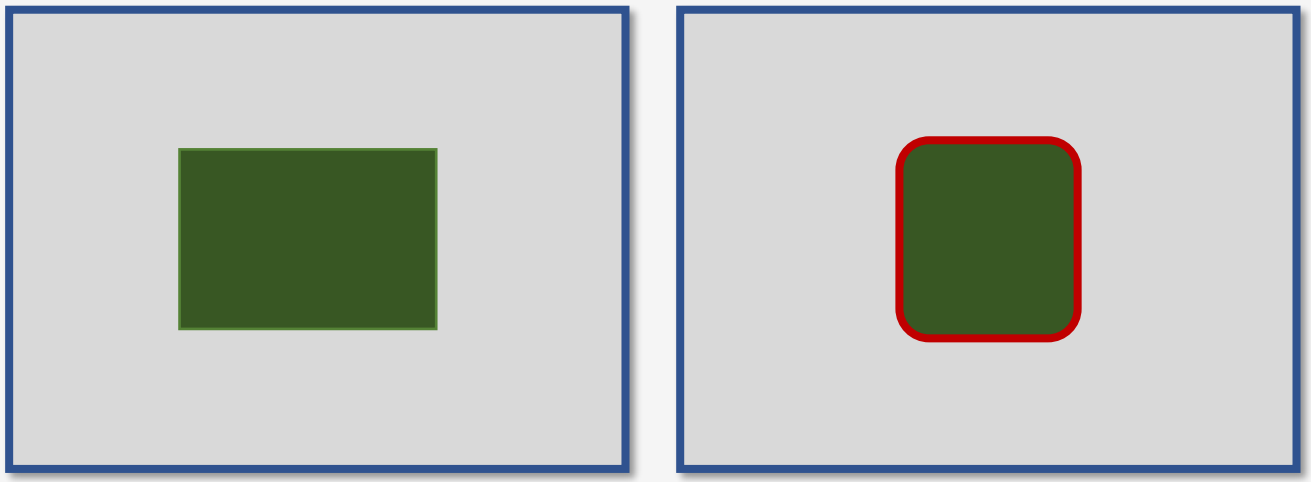
Элемент	Описание
Item	Базовый тип для всех элементов. Аналог *QWidget* в *Qt*. Элемент – невидимый, но имеет позицию, ширину, высоту. Служит для объединения в группу видимых элементов. Используется для создания слоев.
Rectangle	Заполненная прямоугольная область с необязательным контуром.
Image	Растровое изображение.
BorderImage	Растровое изображение с контуром.
Text	Форматированный текст.
TextEdit, TextInput	Ввод и отображение текста.
Window	Элемент главного окна приложения.
WebView	Для отображения веб-содержания.
ListView	Представление в виде списка.
GridView	Представление в виде таблицы.
PathView	Представление, позволяющее располагать элементы вдоль определенного пути и контролировать из масштаб, прозрачность и другие атрибуты

Практический пример 1

```
import QtQuick 2 .15
import QtQuick.Window 2.15

Window {
   width: 360;  height: 360; visible: true
   Rectangle {
      color: "darkkhaki"
       x: 100
       y: 50
       width: 170
       height: 200
       border. color: “red”
       border. width: 10
       radius: 40
       smooth: true
   }
}
```

“first_rec.qml”

Имена элементов всегда начинаются с *заглавных букв* . В *QML* так же, как и в *Qt*, существует механизм объектной иерархии, и элементы тоже могут иметь *предков и потомков.*



*Элемент ***Window*** является элементом верхнего уровня и выполняет роль основного окна, начальные размеры которого 360 x 360 задаются при помощи свойств *width* и *height*. Элемент ***Rectangle*** является потомком ***Window*** и отображается относительно и внутри предка на позиции *x*: 100 и *y*: 50 (свойства *x* и *y*) с размерами 170 x 200 (свойства *width* и *height*). Свойством *color* ему присваивается цвет заполнения - темное хаки : "**darkkhaki**" .

Атрибуты QML-элементов

Каждый тип QML-элементов (визуальный, не визуальный) имеет определенный набор атрибутов. Каждый экземпляр создается с набором атрибутов, определенных для этого типа QML-элемента. Существует несколько различных *типов атрибутов*, которые можно указать.

Набор атрибутов QML-элемента выглядит следующим образом:

- атрибут **id** - экземпляру элемента может быть присвоено значение (имя), позволяющее идентифицировать этот экземпляр. Через атрибут *id* один экземпляр может ссылаться на другие экземпляры. Атрибут *id* должен начинаться с буквы нижнего регистра или символа подчеркивания.
- атрибуты свойств **(property)** - им может быть присвоено статическое значение или значение, привязанное к динамическому выражению. Значение «свойства» может быть прочитано, изменено(если это явно не запрещено) другими экземплярами.
- атрибуты сигналов **(signal)** - это уведомление от экземпляра о том, что произошло какое-то событие: например, изменилось свойство, запустилась или остановилась анимация, загрузилась картинка.
- атрибуты обработчиков сигналов, обработчик изменений свойств **(On<Signal>, On<Property>Changed)**– экземпляр может быть уведомлен через обработчик всякий раз, когда испускается конкретный сигнал. Он должен быть объявлен в определении экземпляра, испускающего сигнал и должен содержать блок JS, выполняющегося при вызове.
- атрибуты методов **(function <nameMeth>(<params,>){ <body>})**- это функции, которые можно вызвать для выполнения каких-либо обработок или инициирования дальнейших событий. Методы могут быть связаны с сигналами, чтобы вызываться всякий раз, когда сигнал испускается.
- прикрепленные свойства и прикрепленные атрибуты обработчиков сигналов - механизмы, которые позволяют расширять экземпляры дополнительными свойствами или обработчиками сигналов, которые иначе недоступны для экземпляра. В частности, они позволяют объектам получать доступ к свойствам или сигналам, которые имеют непосредственное отношение к отдельному другому типу экземпляра.
- атрибуты перечислений **(enum)**- предоставляют фиксированный набор именованных вариантов.

Задание атрибутов

Задание «свойств»

```
Item {
    property int someNumber
    property string someString
    property url someUrl
}
```

«Обработчик изменений свойств»

```
TextInput {
    text: "Change this!"
    onTextChanged: console.log("Text has changed to:", text)
}
```

«Задание метода» (вызывается при инициализации высоты элемента)

```
Rectangle {
    id: rect
    function calculateHeight() {
        return rect.width / 2;
    }
    width: 100
    height: calculateHeight()
}
```

«Обработчик сигналов»

```
Rectangle {
    id: root
    signal activated(real xPosition, real yPosition)
    signal deactivated
    property int side: 100
    width: side; height: side
    MouseArea {
        anchors.fill: parent
        onReleased: root.deactivated()
        onPressed: (mouse)=> root.activated(mouse.x, mouse.y)
        onDeactivated: console.log("Deactivated!")
        onActivated: (xPosition, yPosition)=> console.log("Activated at " + xPosition + "," + yPosition)
    }
}
```

Задание «id»

```
Column {
    width: 200; height: 200
    TextInput {
        id: myTextInput;
        text: "Hello World"
    }
    Text {
        text: myTextInput.text
    }
}
```

Задание «enum»

```
Text {
    enum TextType {
        Normal,
        Heading
    }
    property int textType: MyText.TextType.Normal
```

Свойства элементов

Первое свойство идентификации - ***id*** - задает элементу имя, с помощью которого можно будет ссылаться на этот элемент.
Второе свойство ***- parent*** - позволяет сослаться на элемент предка.

Свойства служат для изменения поведения и внешнего вида элементов.

Имена для идентификаторов (для свойств ***id***) должны начинаться либо с маленькой буквы, со знака подчеркивания и могут содержать только буквы, числа и знаки подчеркивания.

К стандартным свойствам относятся:

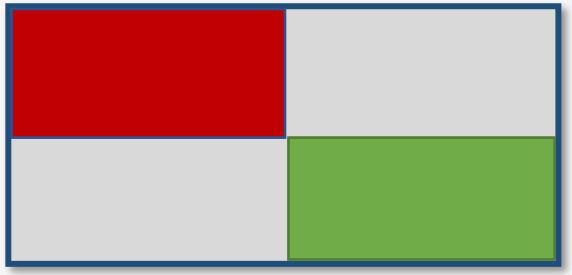
- св-ва для позиционирования: ***x, y, z, position***;
- св-ва для задания и получения размеров: ***width, height, implicitWidth, implicitHeight***;
- св-ва для графических операций и преобразований: ***rotation, scale, clip, transform, transformOrigin, antialiasing, smooth;***
- серия св-в фиксации: ***anchors ;***
- св-ва для ссылок на элемент: ***id, parent***;
- св-ва доступности/недоступности: ***enabled;***
- св-ва установки фокуса: ***focus;***
- св-ва управления видимостью: ***visible, opacity, visibleChildren***;
- св-ва управления состояниями и переходами: ***states, state, transitions*** ;
- серия свойств для работы со слоями: ***layer***;
- св-ва для работы с дочерними элементами: ***children, childrenRect***.

*Все элементы содержат стандартные свойства и могут также быть расширены *дополнительными собственными свойствами*.

Практический пример 2

```
import QtQuick 2.8
import QtQuick.Window
Window {
    width: 360
    height: 360
    Rectangle {
        id: redrect;
        color: "red“
        x: 0;  y: 0
        width: parent.width / 2
        height: parent.height / 2
    }
    Rectangle {
            color: "green";
            x: redrect.width
            y: redrect.height
            width: redrect.width
            height: redrect.height
            }
}
```

При изменении размеров родительского элемента, QML «видит», что значения свойств одного объекта изменяются, и после этого осуществляет связывание со свойствами, использующими это значение.



```
import QtQuick 2.8
import QtQuick.Window
Window {
    width: 200
    height: 200
    Rectangle {
        width: parent.width
        height: parent.height
        onWidthChanged: {
            console.log("width changed:" + width)
        }
        onHeightChanged: {
            console.log("height changed:" + height)
        }
    }
}
```

Свойства для отображения текущих размеров

Собственные свойства

Можно не только использовать уже существующие свойства, но и добавлять свои собственные.

Этой цели служит ключевое слово ***property***, которое имеет следующий синтаксис:

[default] [required] [readonly] **property <тип> <имя> [: <значение>/<выражение>]**

Свойства строго типизированы и свойствам одного типа не могут быть присвоены значения других типов.

Тип QML	Описание
bool	Логический тип, принимает значения false, true
double	Число двойной точности (double precision)
enumerator	Тип перечисления
int	Целочисленный тип (например, 400)
list	Список объектов
real	Числа с плавающей запятой (например, 1. 7)
string	Строка (например «Hello QML»)
time	Время в формате HH:MM:SS (например, 09:23:01)
url	Строка URL (Uniform Resource Locator)
var	эквивалентный *QVariant*, используется для переменных любого типа

Тип Qt Quick 2.8	Описание
color	Тип для обозначения цвета
date	Дата в формате YYYY-MM-DD
font	Тип шрифта
matrix4x4	Матрица из четырех строк и четырех столбцов
point	Тип точка. Состоит из пары значений: x и y
quaternion	Состоит из четырех атрибутов: scalar, x, y и z
rect	Тип прямоугольник. Состоит из четырех значений: x, y, width (ширина) и height (высота)
size	Тип размер. Состоит из значений: width и height
Vector2d	Состоит из x-, y-координат
Vector3d	Состоит из x-, y- и z-координат
Vector4d	Состоит из x-, y- z- и w-координат

Собственные свойства

Объект может иметь одно свойство **по умолчанию**. Свойство по умолчанию – это свойство, которому присваивается значение, если объект объявлен в определении другого объекта без объявления его как значения для определенного свойства. При его объявлении используется ключ. слово ***default.***

```
// MyLabel.qml
import QtQuick 2.0

Text {
    default property var someText
    text: "Hello, " + someText.text
}
```

```
MyLabel {
    Text { text: "world!" }
}
```

В объявлении объекта можно определять обязательное свойство, используя ключевое слово ***requiared***.
Обязательные свойства должны быть снабжены значениями при создании экземпляра объекта. Нарушение этого правила приведет возникновению ошибки.

Дополнение элемента

```
import QtQuick 2.15
Window {
    width: 200 ; height: 100
    Item {
        id : myelement
        property string name: “My Element “
        property int ver: 1
        property real pi: 3.14159
        property bool condition: true
        property var variant: 53 . 1
        property url  link: “http: //www. bhv. com/ “
    }
   Text {
       x: 0;  y: 0
       text : myelement .name + “<br>”
              + myelement.ver + “<br>”
              + myelement.pi + “<br>”
              + myelement .condition + “<br>”
              + mye lement .variant + “<br>”
              + myelement.link  }
}
```

Дополнение свойствами QML-элемента

Обращение к доп. свойствам QML-элемента

```
Rectangle {
 . . .
    QtObject  {
        id: priv
        readonly property int nX: 23
        readonly property int nX: 50
    }
    x: priv.nX
    y: priv.nY
 . . .
}
```

Если требуется инкапсуляция

```
onConditionChanged: {
//do something
}
```

onX - свойство

Каждое из новых свойств после определения тоже будет иметь соответствующее свойство типа *onX*, которое станет реагировать на каждое изменение свойства.
Все новые свойства вместе с их ***onX***-свойствами будут также доступны в *Qt Creator* для автоматического дополнения.

Создание элементов

Для каждого нового *QML-элемента* нужен свой отдельный файл. Элементы, расположенные в одном и том же каталоге, автоматически доступны друг для друга.

Синоним (***alias***) - это механизм для опубликования свойств под другим именем на более высоком уровне иерархии.

Для создания нового элемента:

1. Генерируем новый проект *Qt Quick* (или используем старый).
2. Создадим и добавим новый файл «*TextField.qml*» в тот же каталог где находится “*main.qml*”.
3. Возможность изменять текст со стороны дает определение ***синонима(alias)*** *к свойству* в элементе *Rectangle*(являющегося базовым) для нашего элемента.

В синонимах нуждаются только свойства вложенных элементов — если их необходимо сделать доступными для изменения извне.

Свойство-синоним ***text,*** *с идентификатором txt, используется для связи его со свойством text элемента Text.*

```
// TextField.qml
import QtQuick 2.15
Rectangle {
    property alias text: txt.text
    property string name: "TextField"
    width: txt.width
    height: txt.height
    Text {
        id: txt
        x: 0
        y: 0
    }
}
```

Создание элемента

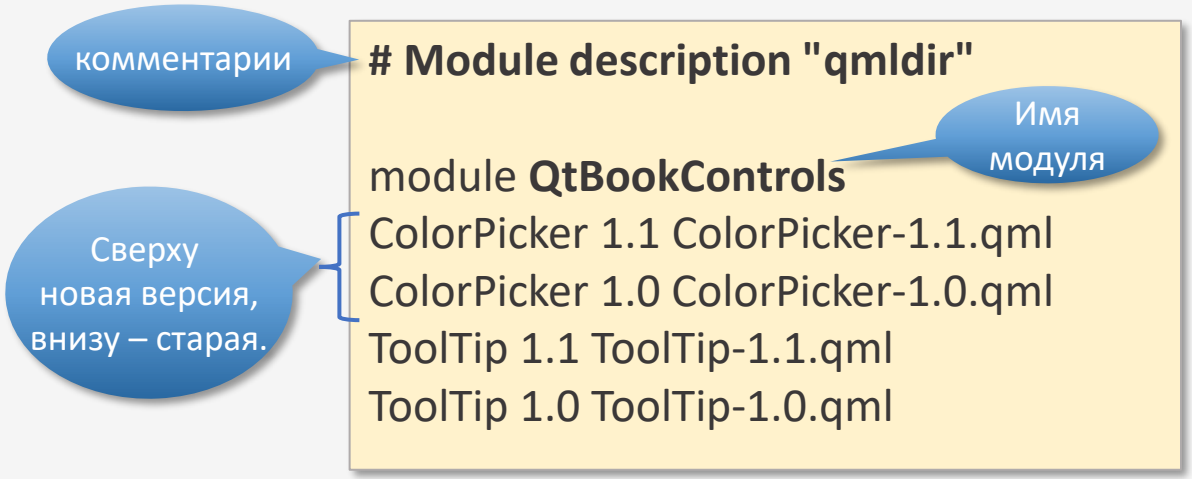
```
// main.qml
import QtQuick 2.15
Window {
      width: 150
      height: 100
      visible: true
      title: qsTr("Hello World")

     TextField {  // новый элемент из TextField.qml
              x: 10
              y: 20
             color: "yellow"
             text: "Text Text Text<br>Text Text Text"
             border.width: 1
     }
}
```

Пример использования нового *QML-элемент* «*TextField*». Поскольку содержащий его файл *TextField.qml* находится в том же каталоге ресурса, что и файл *main.qml*, он автоматически доступен.
Текст устанавливаем при помощи введенного нового свойства ***text*** (только мы знаем, что это новое свойство).
Устанавливаем толщину рамки, равную 1 px (свойство *border.width*).

Создание модулей

Для повторного использования коллекции QML-элементов их можно объединять в отдельные модули. Эти модули можно импортировать для использования в *QML*-приложениях с помощью директивы ***import***.
Средства QML позволяют назначать модулям номера версий.

Чтобы создать модуль, нужно создать каталог и разместить в нем *QML-файлы* элементов и файл описания, который должен называться ***qmldir***. В этом файле должны быть указанные все входящие в модуль *QML*-файлы с указанием номеров их версий.



Использовать каталог с компонентами модуля в программе QML можно так:

import "QtBookControls"

Если каталог расположен **уровнем выше**, то так:

import " .. /QtBookControls"

Можно также задать и **идентификатор** для модуля:

import "QtBookControls" as QBC.

Этот подход можно также использовать, если вы намерены поместить свой модуль с компонентами на удаленном сервере и получать доступ к нему по сети, например через Интернет.

Динамическое создание элементов

В *QML* имеется удобный механизм, который позволяет избежать повторного копирования кода элементов. Этот *QML-элемент* называется *повторителем* (***Repeater***). Он позволяет производить элементы, используя любой тип **модели данных(*model*)**, которая может быть указана статически в виде:

- *числа*,
- *массива*,
- *набора элементов JSON-типа*.

Модель данных(*modelData*) - является инструкцией к созданию элементов, а содержащийся внутри *повторителя* элемент представляет собой *шаблон*, по образцу которого будут создаваться элементы.

```
Repeater {
    model: 10
    Rectangle { }
}
```

Создание 10 элементов ***Rectangle***

```
Repeater {
    model: [“one”,  “two”, “three”, “four”, “five”']
    Text { text: modelData}
}
```

Создание 5 элементов ***Text*** из массива строк. Каждый элемент будет отображать отдельный элемент массива

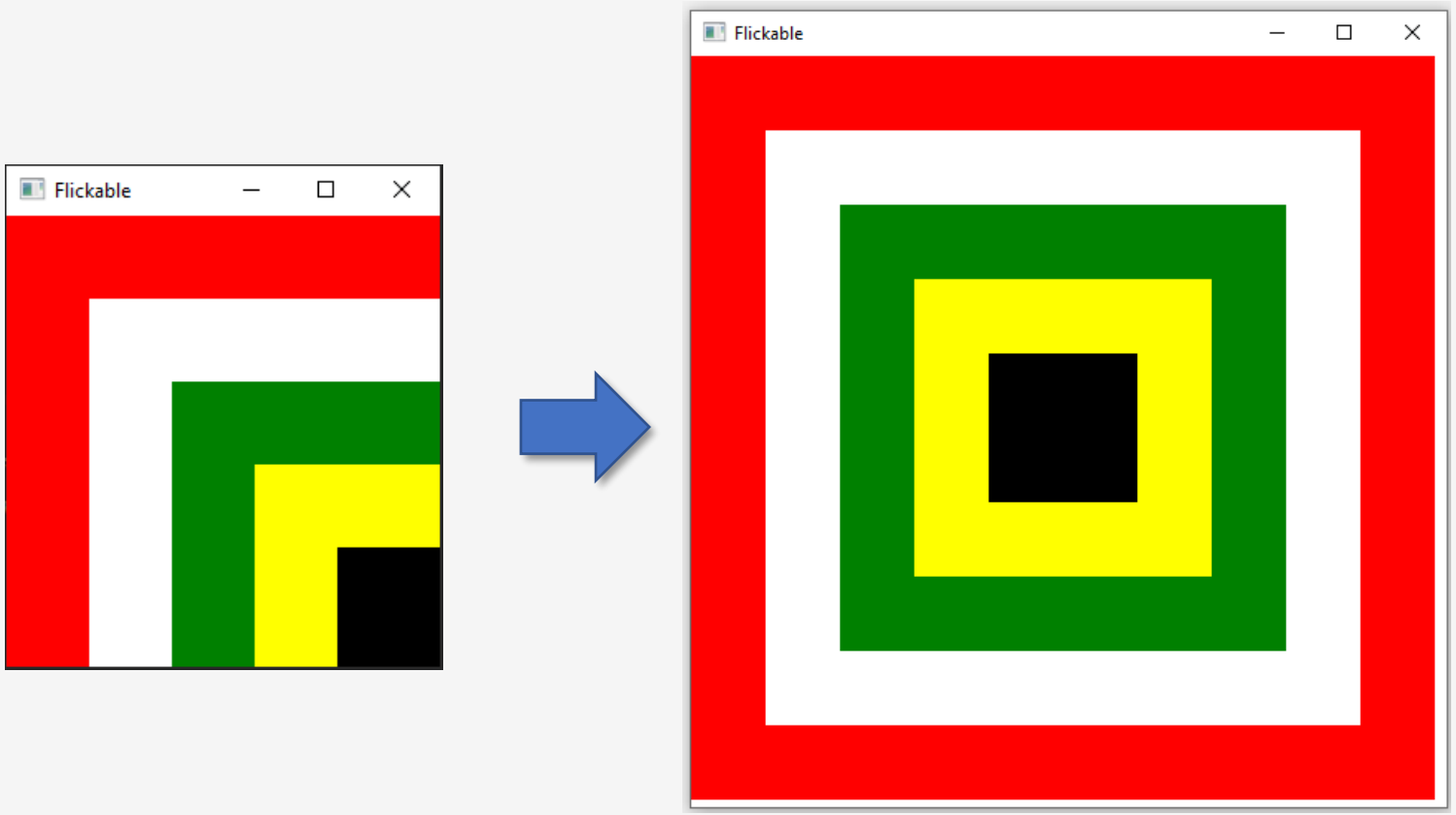
Элемент Flickable

QML-элемент ***Flickable*** напоминает класс из библиотеки *Qt* - ***QScrollArea***. Он так же предназначен для показа элементов или их частей, в том случае, если размеры превышают размеры области показа. Перемещать части элемента для показа можно при помощи пальца на сенсорном экране или мыши на компьютере. Само перемещение сопровождается также эффектом анимации.

Пример использования ***Flickable***, с элементом ***Repeater:*** Создаем с помощью повторителя 5 элементов ***Rectangle*** разных цветов и размещаем их внутри элемента ***Flickable***. Размеры элементов ***Rectangle*** превышают размеры самого элемента ***Flickable***. Чтобы увидеть другую его часть изображения, необходимо переместить область просмотра в нужное место.

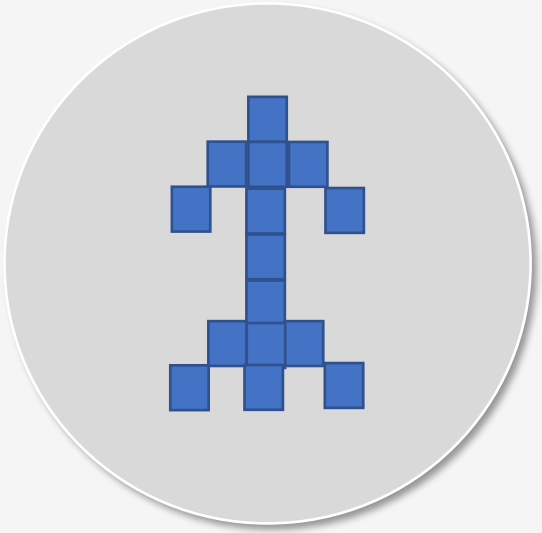
```
import QtQuick 2.15
Flickable {
    id: view
    width: 250
    height: 250
    contentWidth: 500
    contentHeight: 500

    Repeater {
        model: ["red", "white", "green", «yellow", "blue"]
        Rectangle {
            color: modelData
            width: view.contentWidth - index * 100
            height: view.contentHeight - index * 100
            x: view.contentWidth / 2 - width / 2
            y: view.contentHeight / 2 - height / 2
        }
    }
}
```

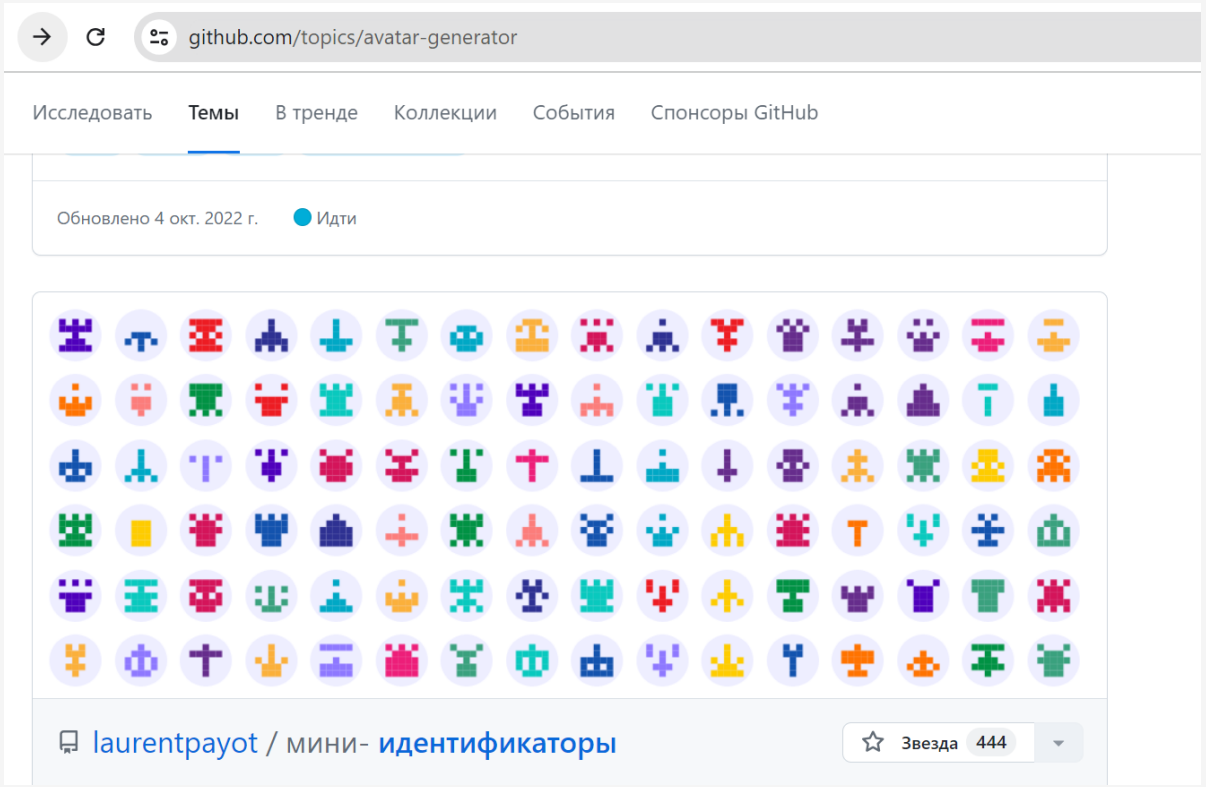
Элемент Flickable



Домашка # 6

1. Создаем аватарки в стиле github





2. Создаем модуль «GeometryFigures» на основе файлов *.qml геометрических фигур: квадрат, круг, треугольник, прямоугольник.