# Размещение элементов QML

# 1. Позиционирование и размер

***Позиционирование и размеры*** *QML- элементов* можно выполнить вручную(*anchors*) или автоматически(*контейнеры*).

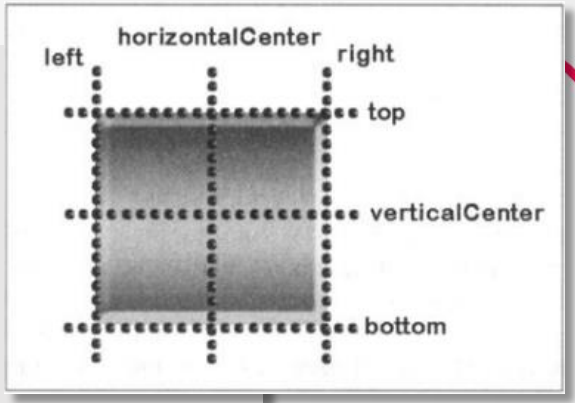
При ручном позиционировании мы явно указываем координаты (x, y) верхнего левого угла элемента, а так же возможно прикрепить один элемент к другому элементу. Для этого применяются **фиксаторы** (***anchors***)- группа свойств, которые есть у любого элемента управления. Они позволяют установить расположение относительно 7 условных линий:

```
Window {
      width: 200
       height: 200
       visible: true
       title: “PositionStudy”
        Rectangle {
              x: 20 ;  y: 10
              width: 150
              height: 100
              color: “green”
        }
}
```

Абсолютное расположение в области родительского элемента

Перечень основных св-в:

- **top**
- **bottom**
- **left**
- **right**
- **horizontalCenter**
- **verticalCenter**
- **baseline-**(воображаемая линия на которой находится текст)
- **centerIn** == (horizontalCenter, verticalCenter)
- **fill** == (top, bottom, left, right)

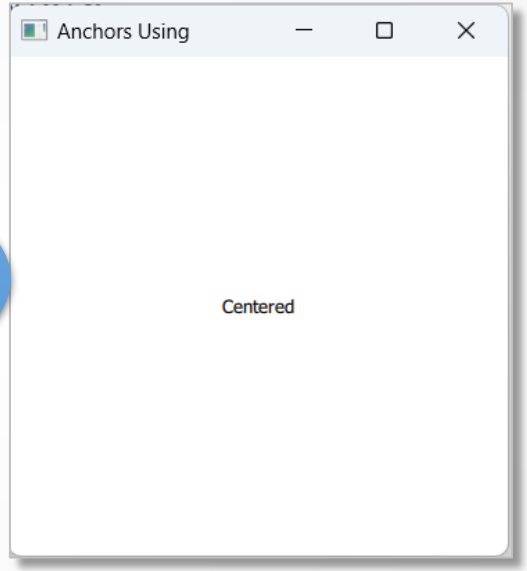


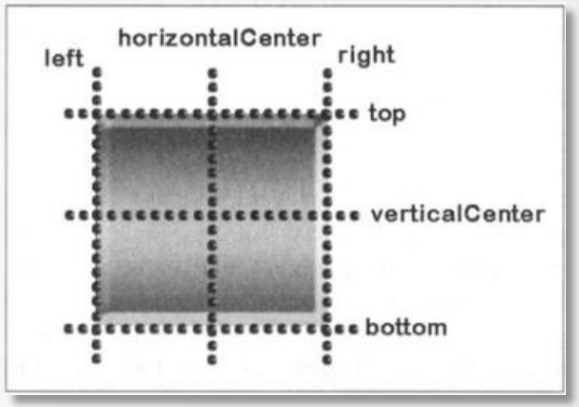
# 2. Фиксаторы

***Фиксатор*** (***anchor***) - задает позиции одного элемента относительно других. Принцип работы таков: вы сами определяете расположение элементов относительно фиксатора. Этот механизм позволяет располагать элементы более интуитивно и с учетом связей самих элементов.

```
import QtQuick 2.15
Import QtQuick.Window 2.15

Window {
    Rectangle {
        width: 360
        height: 360
        Text {
            text: "Centered"
            anchors.centerIn: parent
        }
    }
}
```

Расположение относительно родительского элемента




=

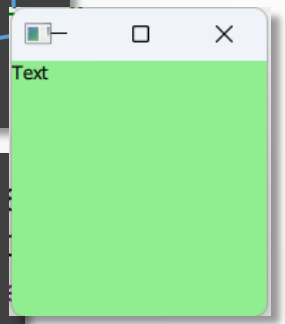
```
Text {
    text: "Centered"
    anchors.horizontalCenter: parent.horizontalCenter
    anchors.verticalCenter: parent.verticalCenter
}
```

# 3. Заполнение всей области элемента

Если нужно заполнить какую-нибудь область и тем самым изменить **не только позицию, но и размеры элемента**, можно связать свойства фиксатора со значениями свойств нужного элемента:

```
import QtQuick 2.15

   Rectangle {
      id: rect; width: 360 ;   height: 360
      Text {
            text: "Text"
            anchors.left: parent.left
            anchors.right: parent.right
            anchors.top: parent.top
            anchors.bottom: parent.bottom
      }
  }
```

Группированные свойства (общие в тематическом плане)

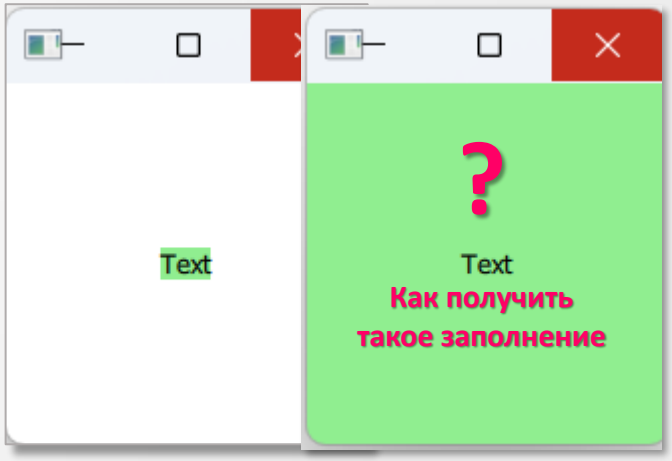



```
import QtQuick 2.15

   Rectangle {
      id: rect;  width: 360 ;  height: 360
      Text {
            text: "Text"
            anchors.fill: parent
      }
}
```

Или использовать свойство anchors.fill:

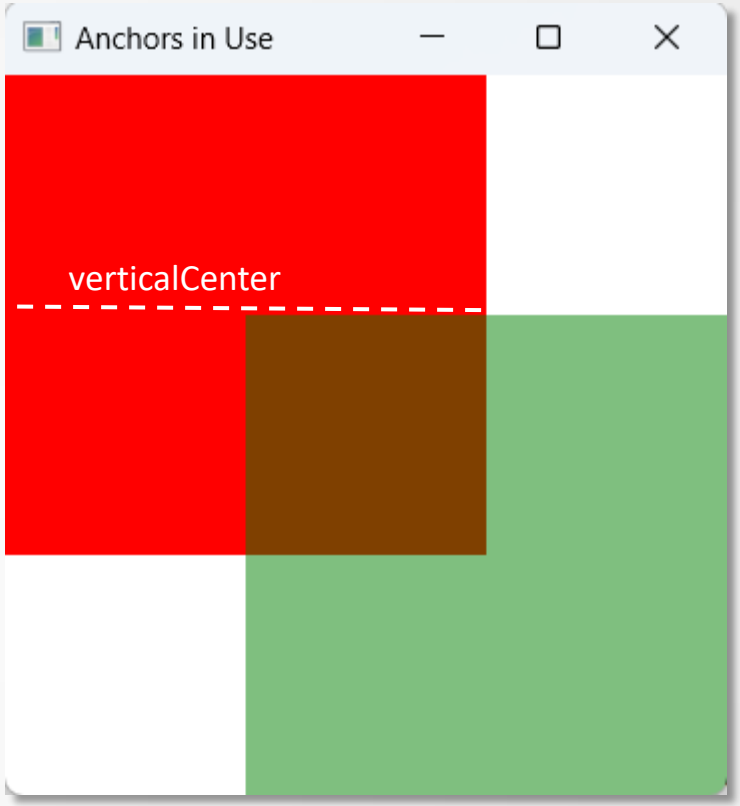
```
import QtQuick 2.15

   Rectangle {
      id: rect; width: 360;  height: 360
      Rectangle {
            color: "lightgreen"
            anchors.fill: text
      }
      Text{
            id: text
            text: "Text"
            anchors.fill: parent
//  anchors.centerIn: parent
      }
   }
```

```
import QtQuick.Window 2.15

Window {
   width: rect.width
   height: rect.height
   visible: true
   title: "Anchors  in Use“
. . .
}
```



# 4. Размещение с перекрытием

```
import QtQuick 2.15
Import QtQuick.Window 2.15

Window {
        width: 360 ;  height: 360
        visible: true
        Rectangle {
            id: redrect ;  color: "red"
             width: parent.width / 1.5
             height: parent.height / 1.5
             anchors.top: parent.top
             anchors.left: parent.left
        }
       Rectangle {
            opacity: 0.5 ;   color: "green"
            anchors.top: redrect.verticalCenter
            anchors.bottom: parent.bottom
            anchors.left: redrect.horizontalCenter
            anchors.right: parent.right
       }
   }
```



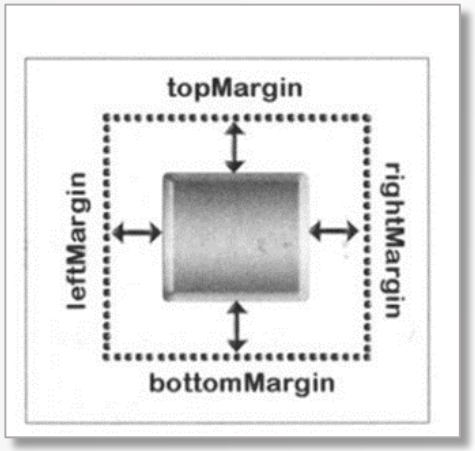
# 5. Контроль размеров среднего элемента

Когда задается вертикальные или горизонтальные расположения с помощью *фиксаторов*, то можно контролировать размеры элементов, которые находятся между другими элементами. На рис. показаны три элемента. У двух крайних ширина задана постоянной, ширина же среднего элемента высчитывается на основании левой и правой границ соседних элементов.


```
import QtQuick 2.15
Window {
    width: 360;  height: 60; visible: true
    Rectangle {
        id: redrect ;  color: "red"
        anchors.left: parent.left
        anchors.top: parent.top
        anchors.bottom: parent.bottom
        width: 60
    }
    Rectangle {
        color: "yellow"
        anchors.top: parent.top
        anchors.bottom: parent.bottom
        anchors.left: redrect.right
        anchors.right: greenrect.left
    }
    Rectangle {
        id: greenrect
        color: "green”
        anchors.right: parent.right
        anchors.top: parent.top
        anchors.bottom: parent.bottom
        width: 100
    }
}
```

# 6. Использование отступов

Отступы от краев элемента можно задать при помощи свойств ***topMargin, bottomMargin, leftMargin u rightMargin***, которые определены в свойстве *anchors*.

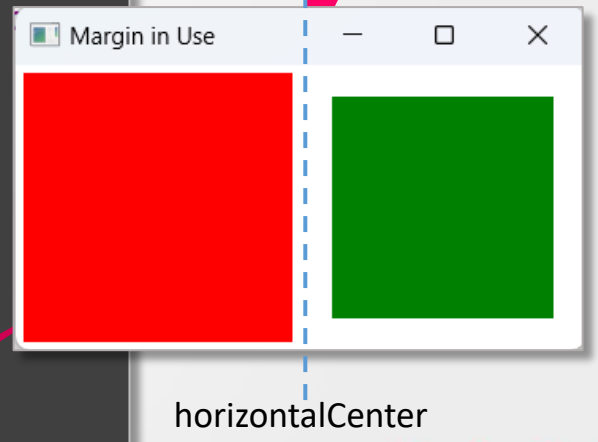



*Относительный отступ задается при помощи свойств, имена которых оканчиваются словом ***Offset***, - если добавить следующие инструкции в блок **anchors**, то будет осуществлен отступ элемента на десять пикселов вниз:

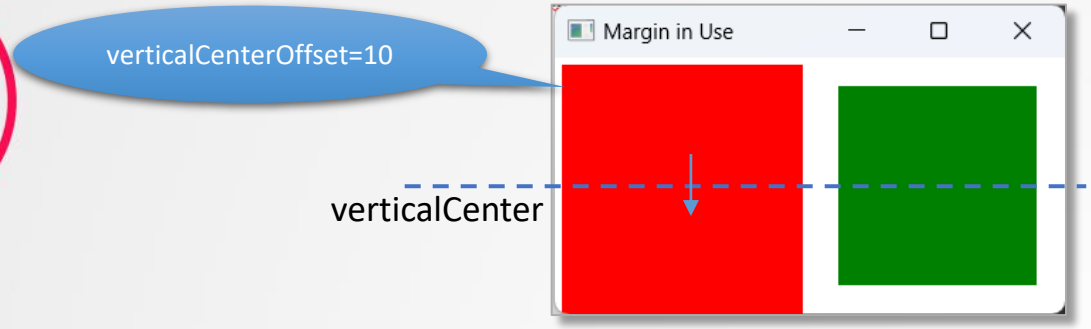
**verticalCenterOffset: 10**
**verticalCenter: parent.verticalCenter**


```
import QtQuick 2.15
    Window {
        width: 360;  height: 180
        visible: true

        Rectangle {
            color: "red"
            anchors {
                right: parent.horizontalCenter
                left: parent.left
                top: parent.top
                bottom: parent.bottom
                leftMargin: 5
                topMargin: 5
                rightMargin: 5
                bottomMargin: 5
            }
        }
    ..............
}
```

```
Rectangle {
            color: "green"
            anchors {
                left: parent.horizontalCenter
                right: parent.right
                top: parent.top
                bottom: parent.bottom
                leftMargin: 20
                topMargin: 20
                rightMargin: 20
                bottomMargin: 20
            }
        }
```



# 7. Margins, offsets, paddings, spacing




Перечень отступов:

- **margin**-размер пустого пространства(поля)
- **offset**-смещение основных линий элемента (*horizontalCenterOffset, verticalCenterOffset, baselineOffset*)

- **spacing-** определяет расстояние, разделяющее дочерние элементы в контейнере
- **padding-** определяет отступ от границ контейнера

# 8. «Традиционное» размещение

«Традиционные» методы размещения похожи на используемые в ***Qt***. Эти размещения являются тоже элементами («***контейнеры-позиционеры***»).

- ***Row, RowLayout*** - область для горизонтального размещения элементов, аналогом в *Qt* является класс *QHBoxLayout;*
- ***Column, ColumnLayout*** - область для вертикального размещения элементов, аналогом в *Qt* является класс *QVBoxLayout*;
- ***Grid, GridLayout*** - область для табличного размещения элементов, аналогом в *Qt* является класс *QGridLayout*;
- ***StackLayout*** - стековая область для размещения, в которой одновременно можно видеть только один элемент. Аналогом подобного размещения в *Qt* является класс *QStackedLayout*.
- ***Flow –*** расположение “Змейкой”

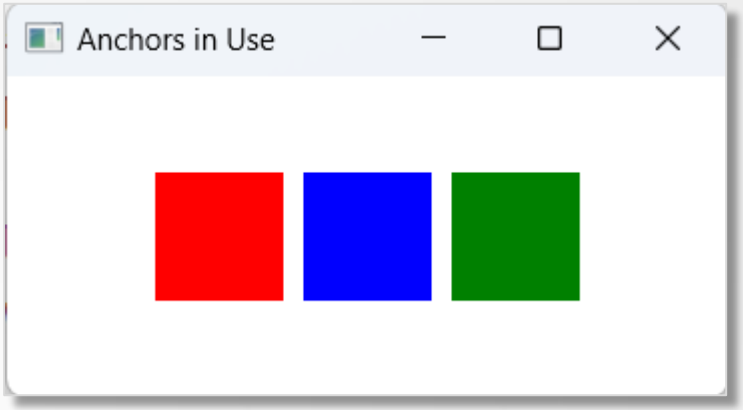
Элементы размещений с **коротким именем** обладают свойствами ***spacing*** и ***layoutDirection***. Эти свойства служат для установки промежутков между элементами и изменения направления размещения соответственно.

Элементы размещений, **оканчивающиеся словом *Layout***, содержатся в отдельном модуле ***QtQuick.Layouts*** и обладают, помимо указанных свойств, дополненным свойством ***Layout***. Это свойство дает возможность устанавливать и получать *min*, *max* и предпочтительную высоту и ширину:

- Layout .minimurnWidth- минимальная ширина;
- Layout .minimшnHeight - минимальная высота;
- Layout .maximurnWidth - максимальная ширина;
- Layout .maximшnHeight - максимальная высота;
- Layout. preferredWidth - предпочтительная ширина;
- Layout .preferredHeight - предпочтительная высота.
- Layout .fillWidth - заполнение по ширине элемента размещения;
- Layout. fillHeight - заполнение по высоте элемента размещения.

# 9. Пример горизонтального размещения

```
import QtQuick 2.15
    Window {
     width: 360 ;  height: 160; visible: true
     Row {
          anchors.centerIn: parent  // контейнер отцентрирован
                                    // по родительскому элементу
          spacing: 10      // расстояние между элементами в
                           // контейнере
          Rectangle {
              width: 64;  height: 64;  color: “red”
          }
          Rectangle {
               width: 64;   height: 64;  color: “blue”
          }
          Rectangle {
               width: 64;   height: 64;  color: “green“
          }
     }
}
```

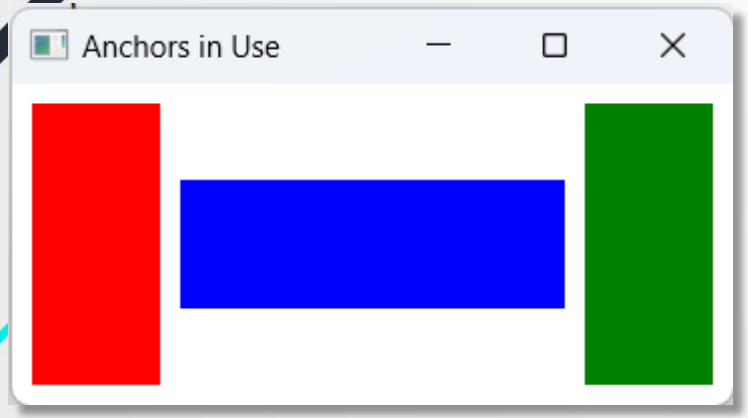



Горизонтальное размещение элементов с помощью элемента ***Row***

Размещение в вертикальном порядке работает аналогично, и чтобы в этом убедиться, в листинге просто замените имя элемента ***Row*** на ***Column.***

# 10. Пример горизонтального размещения

Включаем модуль ***QtQuick. Layouts*** для использования размещений. Заполняем всю область окна элемента размещения ***RowLayout*** присвоением *anchors.fill* элемента предка (parent). Устанавливаем рамку (бордюр), равную 10 (свойство *margins*), и фиксированное расстояние между элементами (свойство *spacing*), тоже равное 10. Встроенному свойству размещения *fillHeight* первого и последнего элементов ***Rectangle*** мы присваиваем значение true - что дает этим элементам возможность увеличиваться по высоте. В среднем элементе для возможности увеличения в ширину мы присваиваем свойству *fillWidth* значение *true*. Всем элементам ***Rectangle*** мы устанавливаем минимальные размеры 64x64 при помощи свойств *minimumWidth* и *minimumHeight*.



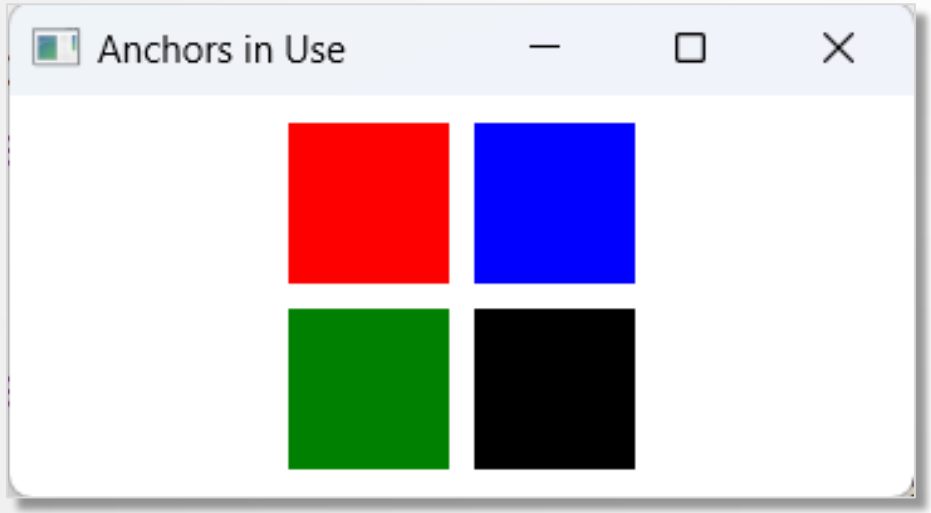

Горизонтальное размещение элементов с помощью элемента ***RowLayout***

Размещение в вертикальном порядке работает аналогично. Просто замените имя элемента ***Row*** на ***Column***.

```
import QtQuick 2.15
import QtQuick.Layouts 1.3
Window {
    width: 360 ;  height: 160; visible: true
    RowLayout {
        anchors.fill: parent
        anchors.margins: 10
        spacing: 10
        Rectangle {
            Layout.fillHeight: true
            Layout.minimumWidth: 64;
            Layout.minimumHeight: 64;
            color: "red"
        }
        Rectangle {
            Layout.fillWidth: true
            Layout.minimumWidth: 64;
            Layout.minimumHeight: 64;
            color: “blue"
        }
        Rectangle {
            Layout.fillHeight: true
            Layout.minimumWidth: 64;
            Layout.minimumHeight: 64;
            color: "green“
        }
    }
}
```

# 11. Размещение в виде таблицы

В табличном размещении (***Grid***) есть дополнительные свойства: ***rows*** и ***columns***, которые задают количество строк и столбцов таблицы.




Размещение элементов в виде таблицы

```
import QtQuick 2.15
  Window {
     width: 360 ;  height: 160; visible: true
      Grid {
           rows: 2;  columns: 2
           anchors.centerIn: parent
           spacing: 10
           Rectangle {
               width: 64; height: 64; color: "red“
           }
           Rectangle {
              width: 64; height: 64; color: “blue"
          }
          Rectangle {
              width: 64; height: 64; color: "green“
          }
          Rectangle {
              width: 64; height: 64; color: “black“
        }
    }
}
```

# 12. Размещение в виде потока

Размещение в виде потока (***Flow***). Оно упорядочивает элементы «Змейкой», которая пытается разместить как можно большее количество элементов в заданной области окна.

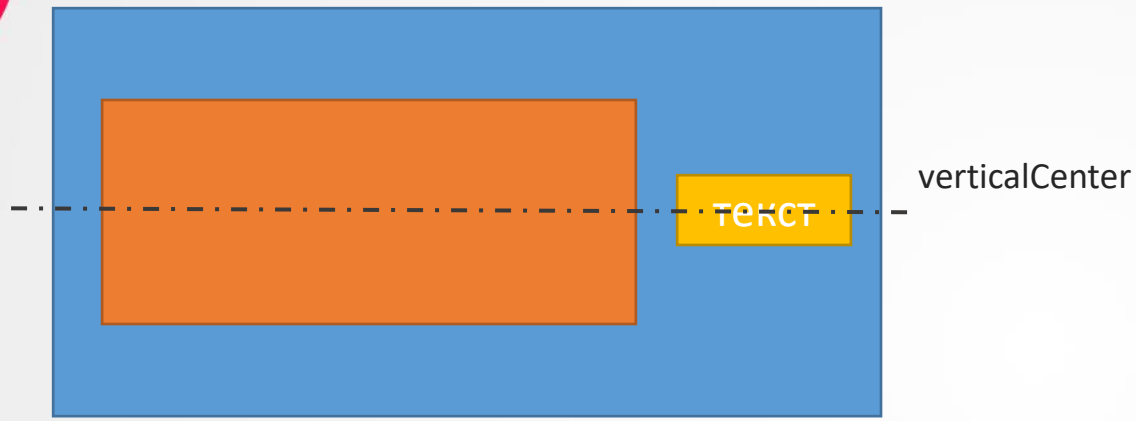

```
import QtQuick 2 . 15
    Window {
        width: 480;  height: 200;  visible: true
        Flow {
            anchors.fill: parent
            anchors.margins: 20 // отступы от всех сторон гл. окна
            spacing: 20     // промежутки между элементами
            Repeater {
                model: {  // создание модели для повторителя
                    var v = new Array (10) ;
                    for (var i = 0; i  <  v.length; ++i ) {
                        v[i] = i % 2 ? "green" : "red"
                    }
                    return v;
                }
                Rectangle {
                    width: 64;  height: 64; radius: 32; color: modelData
                    Text {
                        color: "white“;  font.pixelSize: 48
                        font. family: "Courier"
                        anchors.centerIn: parent; text: index
                    }
                }
            }
        }
}
```

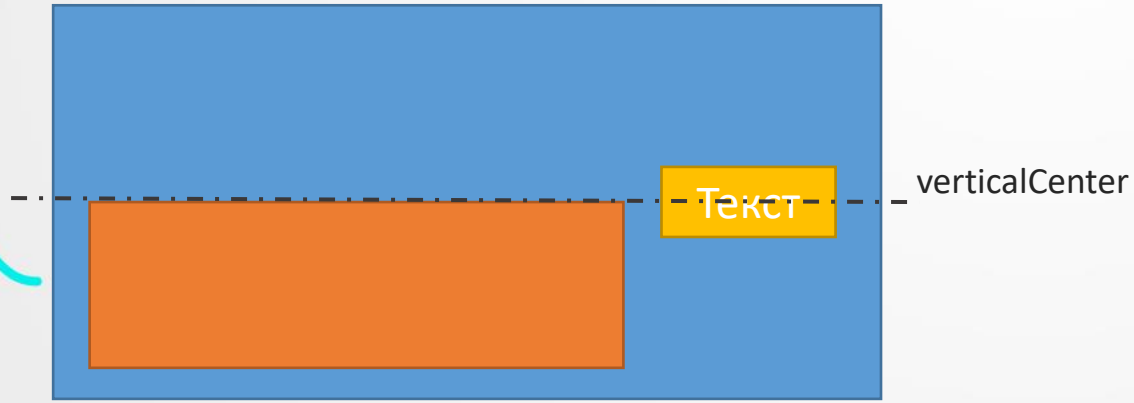
# 13. Размещение в виде потока

Элемент размещения ***Flow*** внутри главного элемента ***Window***.
Основной цвет элемента задается в прямоугольнике посредством текущего значения модели данных (*modelData*). Цифровое значение отображается при помощи дочернего элемента ***Text***, в котором само значение является текущим индексом элемента (*index*) встроенной модели.

# 14. Домашка # 8

Выполнить размещение элементов при помощь фиксаторов:




Выполнить размещение элементов «традиционным» способом: