Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
«Севастопольский государственный университет»

ИССЛЕДОВАНИЕ СПОСОБОВ ПОСТРОЕНИЯ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ С ПОМОЩЬЮ ЯЗЫКА РАЗМЕТКИ QML

Методические указания

к лабораторным работам по дисциплине

«Кроссплатформенное программирование»

для студентов, обучающихся по направлению

09.03.02 “Информационные системы и технологии”

очной и заочной форм обучения

Севастополь
2018

УДК 004.415.2
Исследование способов построения интерфейса пользователя с помощью языка разметки QML. Методические указания/Сост. Строганов В.А. – Севастополь: Изд-во СевГУ, 2018.–20 с.

Методические указания предназначены для оказания помощи студентам при выполнении лабораторных работ по дисциплине «Кроссплатформенное программирование».

Методические указания составлены в соответствии с требованиями программы дисциплины «Кроссплатформенное программирование» для студентов направления 09.03.02 и утверждены на заседании кафедры «Информационные системы»,
протокол № от « »_____ 2018 г.

Содержание

1. ЦЕЛЬ РАБОТЫ

Изучить основы языка разметки QML. Приобрести практические навыки создания графических интерфейсов Qt-приложений на основе разметки.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Традиционно все настольные приложения пишутся на императивных языках программирования. Этот подход прост и понятен: куда проще описать последовательность действий для решения той или иной задачи, нежели поставить задачу в понятной для машины форме, но когда речь заходит о проектировании внешнего вида и поведения, возникают сложности. Веб дизайнеры же привыкли описывать, как должно выглядеть веб приложение, то есть ставить задачу, а не по пунктам описывать её решение, такой подход называется декларативным, он удобен, но к сожалению в традиционных приложениях до сих пор господствует именно императивный подход. Есть конечно дизайнеры форм, но они лишь позволяют в общих чертах обрисовать внешний вид приложения и совершенно не способны описать его поведение. Для решения этой проблемы в Qt Software был предложен новый проект Declarative User Interface и в рамках него новый язык разметки: QML

QML – декларативный язык программирования, основанный на JavaScript, предназначенный для дизайна приложений, делающих основной упор на пользовательский интерфейс. Является частью Qt Quick, среды разработки пользовательского интерфейса, распространяемой вместе с Qt. В основном используется для создания приложений, ориентированных на мобильные устройства с сенсорным управлением.

QML-документ представляет собой дерево элементов. 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. QML элементы могут быть дополнены стандартными JavaScript вставками путем встраивания .js файлов. Также они могут быть расширены C++ компонентами через Qt Framework.

Основная идея использования QML – отделение логики приложения от его представления, что в итоге придает вашему приложению дополнительную гибкость, так как в итоге логика и представление слабо связаны. Эта слабая связь также необходима при покрытии кода unit-тестами, так как представление и модель могут быть легко заменены классами-макетами. Современные практики разработки ПО невозможно представить без подобного разделения.

2.1. Построение приложений на основе паттерна MVC

Одним из наиболее распространенных и эффективных приемов проектирования программ является использование шаблона программирования MVC (Model-View-Controller) — Модель-Представление-Контроллер. MVC позволяет разделить части программы, отвечающие за хранение и доступ к данным, отображение данных и за взаимодействие с пользователем на отдельные слабо связанные модули. Подобное разделение ответственности упрощает структуру программы и позволяет вносить изменения в одну из этих частей, не затрагивая остальные.

Разделение ответственности является одним из основополагающих принципов программирования. Суть его состоит в том, чтобы разделять программу на функциональные блоки, каждый из которых отвечает за свою часть работы. Этими блоками могут быть и функции, и объекты, и модули и т.д. Блоки между собой связываются через определенные интерфейсы. Это дает некоторую независимость блоков относительно друг друга и снижает сложность системы. Паттерн проектирования MVC позволяет снизить сложность и упростить архитектуру программ с графическим интерфейсом при помощи разделения ответственности. Программа при этом разделяется на три компонента:

1. Модель. Отвечает за данные и обеспечивает доступ к ним.
2. Представление. Отвечает за отображение данных, полученных из модели.
3. Контроллер. Отвечает за реализацию функциональности. Может изменять данные в модели.

Между этими компонентами есть определенные интерфейсы, позволяющие развязать их между собой. В идеале эти модули вообще должны независимы друг от друга и позволять вносить изменения или даже полностью заменить какой-либо модуль без переделки остальных.

Поскольку одним из основных предназначений Qt является разработка графических интерфейсов пользователя, он активно применяет MVC. Qt использует свою вариацию MVC — Model-View. Основная идея применения данного шаблона в Qt — разделение данных и их отображения.

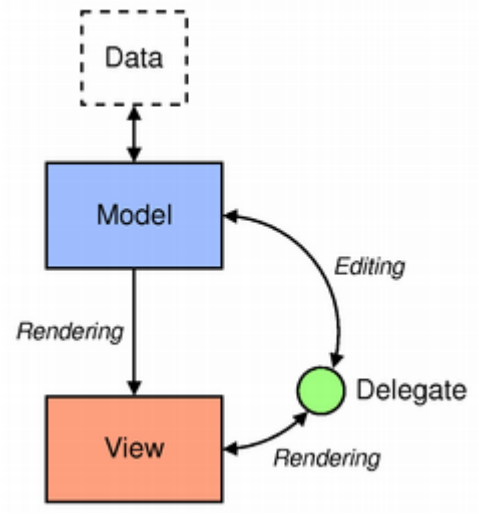

Рисунок 2.1 – Схема MV

Здесь модель отвечает за данные и доступ к ним. Поскольку в графическом интерфейсе за отображение элементов и получение ввода от пользователя нередко отвечают одни и те же элементы, то вполне логична идея объединить представление и контроллер. Именно так и сделано в Qt: представление не только отображает данные, но и выполняет функции контроля и отвечает за взаимодействие с пользователем. Но для того, чтобы из-за такого объединения не терять гибкость, было введено понятие делегата. Делегат позволяет определять, как эти данные будут отображаться и как пользователь может их изменять. Представление же, по сути, теперь является контейнером для экземпляров делегата.

В QML этот принцип тоже применяется, даже в гораздо большей степени. Model-View является одной из основополагающих концепций QML. Одно из основных предназначений QML — это выделение интерфейса программы в отдельную часть, которая может быть очень гибкой и легко адаптироваться для разных нужд. Например, для десктопного приложения может быть один вариант интерфейса, а для мобильного – другой. При этом ядро программы (модель) может быть написано на C++ и оставаться неизменным.

2.2. Пример создания интерфейса пользователя на основе QML

В качестве примера рассмотрим создание приложения «Калькулятор» на QML. Для простоты ограничимся одной лишь операцией суммирования.

2.1.1. Создадим новый проект Qt Quick 2.

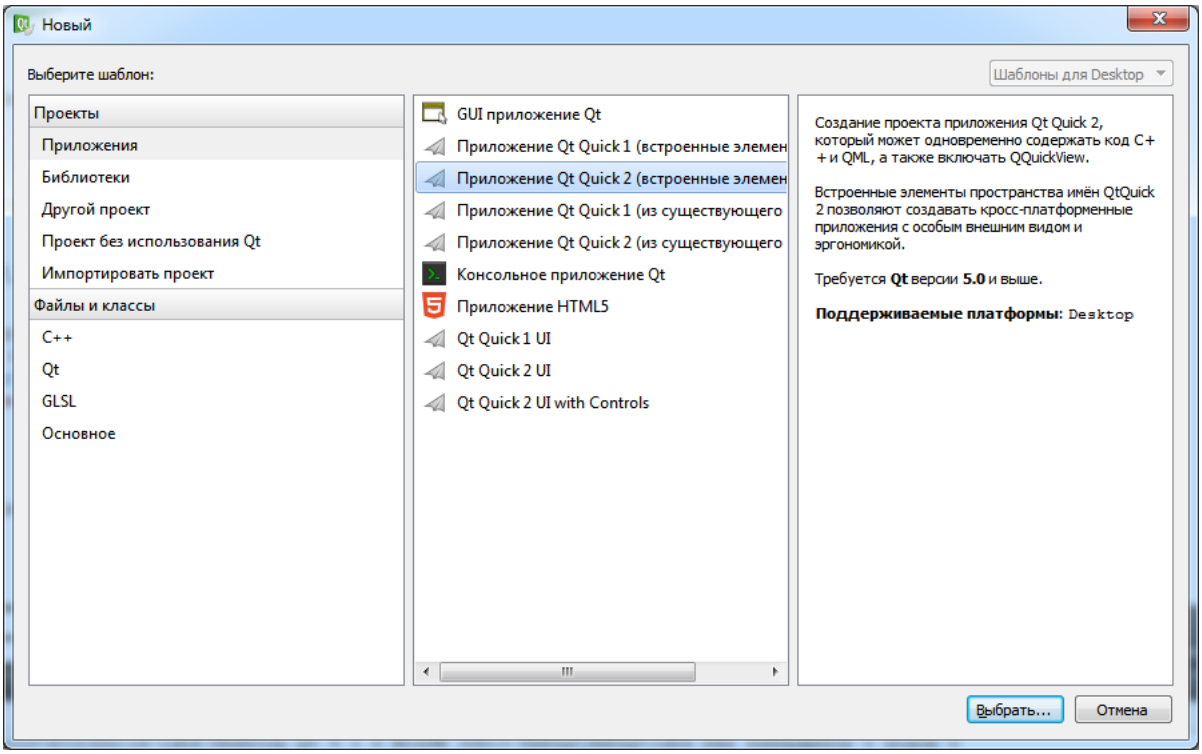

Рисунок 2.2 – Создание нового проекта Qt Quick 2

2.1.2. Определим собственный QML-элемент – кнопку. Нажимаем правой кнопкой на проект и выбираем «Добавить новый…». Далее выбираем «Файл QML (Qt Quick 2)».

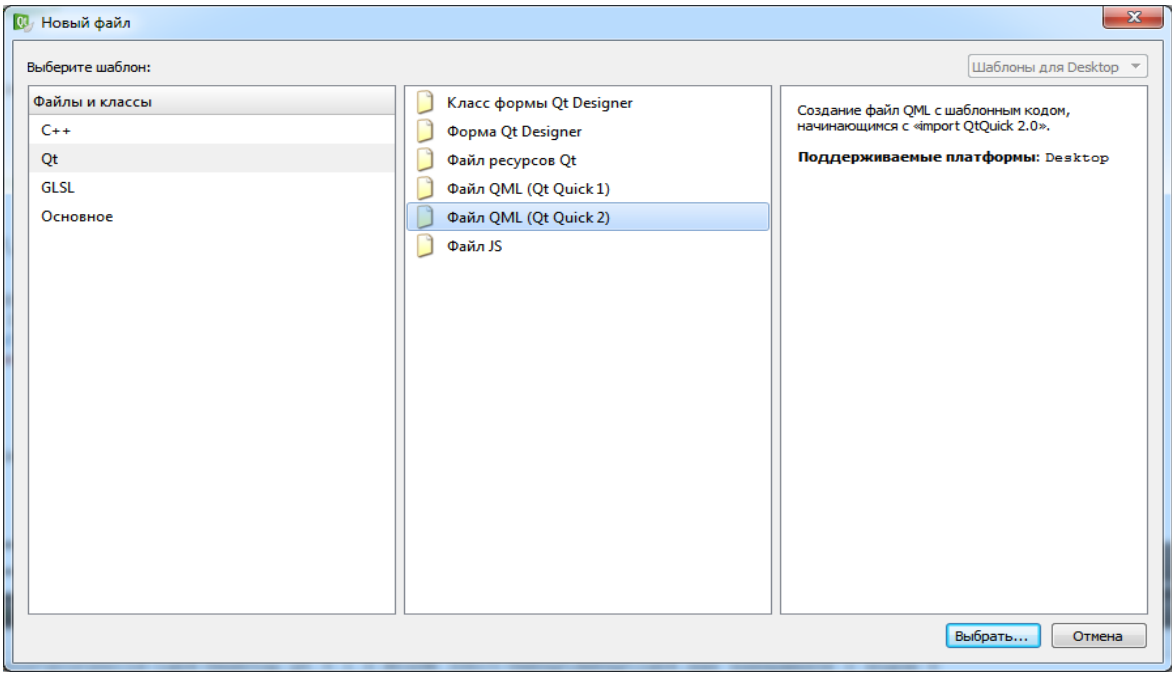

Рисунок 2.3 – Создание QML-элемента

Указываем имя Button. Сохраняем в отдельном новом подкаталоге «core».

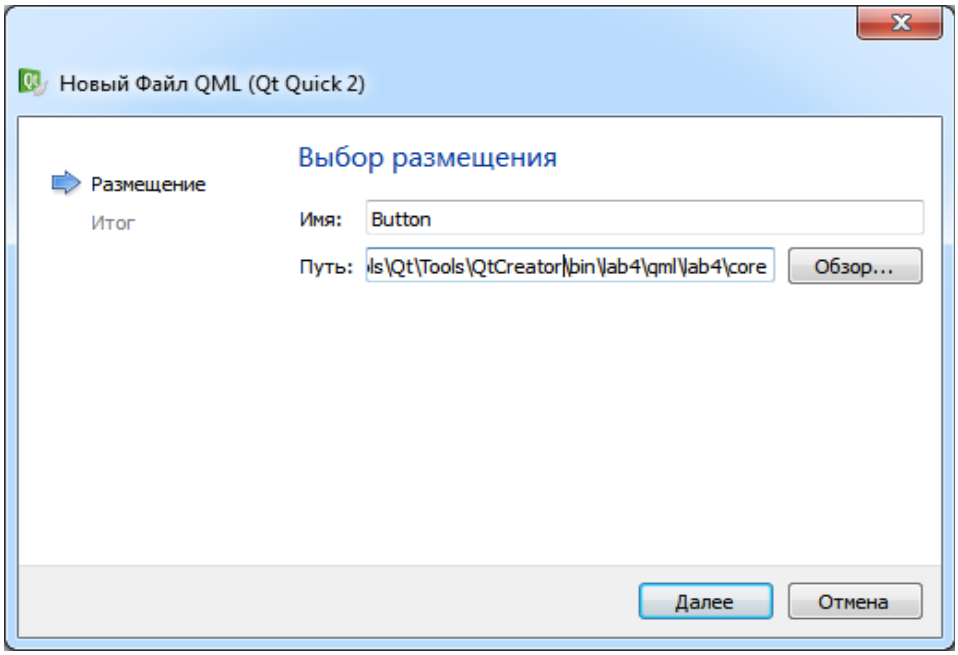

Рисунок 2.4 – Создание QML-элемента

Элемент “Кнопка” будет представлять собой следующее (пример взят из официальной документации. Пояснения приведены в комментариях):

```
/****************************************************************************
    **
    ** Copyright (C) 2012 Digia Plc and/or its subsidiary(-ies).
    ** Contact: http://www.qt-project.org/legal
    **
    ** This file is part of the QtQml module of the Qt Toolkit.
    **
    ** $QT_BEGIN_LICENSE:BSD$
    ** You may use this file under the terms of the BSD license as
follows:
    **
    ** "Redistribution and use in source and binary forms, with or
without
    ** modification, are permitted provided that the following
conditions are
    ** met:
    **   * Redistributions of source code must retain the above
copyright
    **     notice, this list of conditions and the following
disclaimer.
    **   * Redistributions in binary form must reproduce the above
copyright
    **     notice, this list of conditions and the following
disclaimer in
    **     the documentation and/or other materials provided with
the
```

```
    **     distribution.
    **   * Neither the name of Digia Plc and its Subsidiary(-ies)
nor the names
    **     of its contributors may be used to endorse or promote
products derived
    **     from this software without specific prior written
permission.
    **
    **
    ** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS
    ** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT
    ** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR
    ** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT
    ** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL,
    ** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT
    ** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE,
    ** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY
    ** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT
    ** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
    ** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE."
    **
    ** $QT_END_LICENSE$
    **

****************************************************************
**********/
    import QtQuick 2.0

    Rectangle {
        //идентификатор элемента
        id: button

        //эти свойства используются как константы, доступны также
извне
        property int buttonHeight: 75
        property int buttonWidth: 150

        //свойство для хранения текста на кнопке
        property string label
        property color textColor: buttonLabel.color

        //цвет при наведении курсора
```

```
        property color onHoverColor: "lightsteelblue"
        property color borderColor: "transparent"

        //цвет кнопки
        property color buttonColor: "lightblue"

        property real labelSize: 14

        //свойства отображения
        radius: 6
        antialiasing: true
        border { width: 2; color: borderColor }
        width: buttonWidth; height: buttonHeight

        Text {
            id: buttonLabel
            anchors.centerIn: parent
            text: label      //привязываем текст к тексту
"родителя"
            color: "#000000"
            font.pointSize: labelSize
        }

        //сигнал, который будет вызываться при нажатии
        signal buttonClick()

        //определяем "кликабельную" зону равную поверхности всей
кнопки
        MouseArea {
            id: buttonMouseArea
            anchors.fill: parent    //размер равен размеру
"родителя"
            onClicked: buttonClick()

            //если равно true, то будет вызываться onEntered и
onExited при наведении/удалении курсора
            //false - необходимо кликнуть чтобы отработал mouse
hover
            hoverEnabled: true

            //отобразить рамку, если навели курсор
            onEntered: parent.border.color = onHoverColor
            //удалить рамку при удалении курсора с кнопки
            onExited:  parent.border.color = borderColor
        }

        //изменить цвет кнопки при нажатии
        color: buttonMouseArea.pressed ? Qt.darker(buttonColor,
1.5) : buttonColor
        //анимация смены цвета
       Behavior on color { ColorAnimation{ duration: 55 } }

        //увеличить кнопку при нажатии
```

```
        scale: buttonMouseArea.pressed ? 1.1 : 1.00
        //анимация при увеличении кнопки
        Behavior on scale { NumberAnimation{ duration: 55 }}
}
```

2.1.3. Добавляем строчку import "core" в main.qml. Это сделает доступным наш тип кнопки.

2.1.4. Открываем дизайнер для файла main.qml. Как мы видим, наш элемент-кнопка стал доступен в дизайнере.

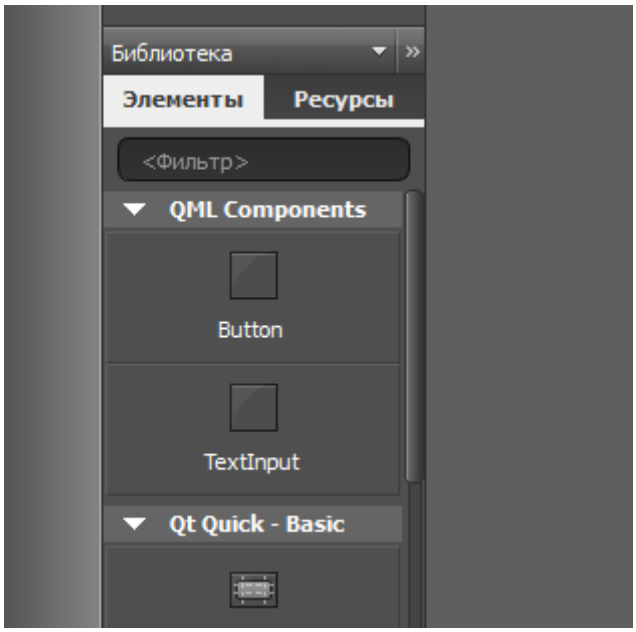

Рисунок 2.5 – Элемент “Button”

2.1.5. Располагаем элементы интерфейса.

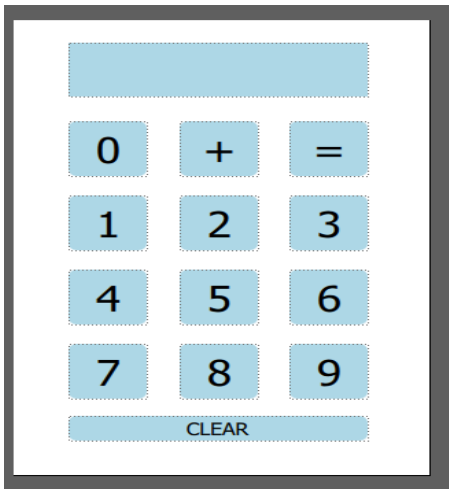

Рисунок 2.6 – Элементы графического интерфейса

2.1.6. Открываем редактор. В файле main.qml реализуем логику

приложения:

```
import QtQuick 2.0
import "core"

Rectangle {
    width: 303
    height: 423
    //флаг, определяющий была ли нажата клавиша "+"
    property bool wasPlusPressed : false
    //флаг, сигнализирующий о том, что при след. вводе необходимо
очистить поле ввода
    property bool shouldClear: false
    //текущая сумма
    property int sum : 0
    //максимальная длина результата/вводимого числа
    property int maxLen: 9

    //функция, обрабатывающее нажатие клавиши-цифры
    function numClicked(value)
    {
        if (shouldClear)
        {
            outputEdit.text = "";
            shouldClear = false;
        }

        if(outputEdit.text.length < maxLen)
        {
            outputEdit.text += value;
        }
    }

        Button {
            id: zeroButton
            x: 40
            y: 94
            width: 57
            height: 51
            textColor: "#000000"
            labelSize: 25
            label: "0"
            buttonWidth: 100
            buttonHeight: 100
            onButtonClick:
            {
                numClicked('0');
            }
        }

        Button {
            id: plusButton
            x: 121
```

```
    y: 94
    width: 57
    height: 51
    labelSize: 25
    label: "+"
    buttonWidth: 100
    buttonHeight: 100
    onButtonClick:
    {
        sum = outputEdit.text * 1;
        wasPlusPressed = true;
        shouldClear = true;
    }
}

Button {
    id: eqButton
    x: 201
    y: 94
    width: 57
    height: 51
    labelSize: 25
    label: "="
    buttonWidth: 100
    buttonHeight: 100
    onButtonClick:
    {
        if (wasPlusPressed)
        {
            if (outputEdit.text != "")
            {
                sum = outputEdit.text * 1 + sum;
                if ((sum + '').length <= maxLen)
                {
                    outputEdit.text = sum;
                }
                else
                {
                    outputEdit.text = "too much :(";
                }
            }
        }
        wasPlusPressed = false;
    }
}

Button {
    id: oneButton
    x: 40
    y: 163
    width: 57
    height: 51
    label: "1"
```

```
        labelSize: 25
        buttonWidth: 100
        buttonHeight: 100
        onButtonClick:
        {
            numClicked('1');
        }
    }

    Button {
        id: twoButton
        x: 121
        y: 163
        width: 57
        height: 51
        label: "2"
        labelSize: 25
        buttonWidth: 100
        buttonHeight: 100
        onButtonClick:
        {
            numClicked('2');
        }
    }

    Button {
        id: threeButton
        x: 201
        y: 163
        width: 57
        height: 51
        label: "3"
        labelSize: 25
        buttonWidth: 100
        buttonHeight: 100
        onButtonClick:
        {
            numClicked('3');
        }
    }

    Button {
        id: fourButton
        x: 40
        y: 232
        width: 57
        height: 51
        label: "4"
        labelSize: 25
        buttonWidth: 100
        buttonHeight: 100
        onButtonClick:
        {
```

```
            numClicked('4');
        }
    }

    Button {
        id: fiveButton
        x: 121
        y: 232
        width: 57
        height: 51
        label: "5"
        labelSize: 25
        buttonWidth: 100
        buttonHeight: 100
        onButtonClick:
        {
            numClicked('5');
        }
    }

    Button {
        id: sixButton
        x: 201
        y: 232
        width: 57
        height: 51
        label: "6"
        labelSize: 25
        buttonWidth: 100
        buttonHeight: 100
        onButtonClick:
        {
            numClicked('6');
        }
    }

    Button {
        id: sevenButton
        x: 40
        y: 301
        width: 57
        height: 51
        label: "7"
        labelSize: 25
        buttonWidth: 100
        buttonHeight: 100
        onButtonClick:
        {
            numClicked('7');
        }
    }

    Button {
```

```
        id: eightButton
        x: 121
        y: 301
        width: 57
        height: 51
        label: "8"
        labelSize: 25
        buttonWidth: 100
        buttonHeight: 100
        onButtonClick:
        {
            numClicked('8');
        }
    }

    Button {
        id: nineButton
        x: 201
        y: 301
        width: 57
        height: 51
        label: "9"
        labelSize: 25
        buttonWidth: 100
        buttonHeight: 100
        onButtonClick:
        {
            numClicked('9');
        }
    }

    Button {
        id: clearButton
        x: 40
        y: 368
        width: 218
        height: 23
        labelSize: 12
        buttonWidth: 100
        label: "CLEAR"
        buttonHeight: 100
        onButtonClick:
        {
            outputEdit.text = "";
            sum = 0;
            wasPlusPressed = false;
        }
    }

    Rectangle {
        id: rectangle1
        x: 40
        y: 21
```

```
            width: 218
            height: 50
            color: "lightblue"
        }

        TextEdit {
            id: outputEdit
            x: 46
            y: 29
            width: 205
            height: 34
            color: "#000000"
            text: qsTr("")
            selectedTextColor: "#000000"
            selectionColor: "#000000"
            horizontalAlignment: TextEdit.AlignRight
            readOnly: true
            cursorVisible: false
            font.bold: true
            font.pixelSize: 30
        }
}
```

2.1.7. Компилируем и запускаем приложение. После запуска окно приложения будет иметь следующий вид:

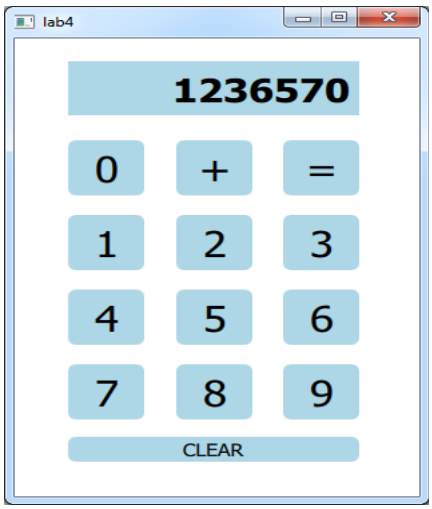

Рисунок 2.7 – Внешний вид окна приложения

3. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

3.1. Изучить принципы построения приложений на основе шаблона MVC (выполняется в ходе самостоятельной подготовки к лабораторной работе).

3.2. Выполнить проектирование графического интерфейса приложения, согласно варианту задания (выполняется в ходе самостоятельной подготовки к

лабораторной работе).

3.3. Создать проект Qt Quick 2 приложение.

3.4. Добавить собственный класс Button в проект (см. пример в разделе 2.1). Изменить его поведение под свой вариант, если необходимо.

3.5. Создать в дизайнере интерфейс, согласно варианту задания.

3.6. Реализовать логику приложения средствами QML по варианту.

3.7. Выполнить сравнительное исследование методов построения интерфейса пользователя, рассмотренных в лабораторной работе №3, и на основе QML. Сравнение провести по критериям: 1) трудоемкости реализации; 2) гибкости получаемого программного решения.

4. СОДЕРЖАНИЕ ОТЧЕТА

4.1. Цель работы.

4.2. Постановка задачи.

4.3. Описание взаимодействия элементов интерфейса. Скриншот окна

4.4. приложения.

4.5. Текст программы.

4.6. Выводы.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1. Что такое QML?

5.2. В чем состоит различие декларативного и императивного подхода?

5.3. Какие плюсы в разделении частей логики и представления в приложении?

5.4. Какие минусы в разделении частей логики и представления в приложении?

5.5. На каком языке пишется интерактивная часть приложений QML?

5.6. Какой паттерн проектирования рекомендуется к использованию с QML?

5.7. На основе какого формата построен язык разметки QML?

5.8. Каково назначение контроллеров в шаблоне MVC?

5.9. Каково назначение моделей в шаблоне MVC?

5.10. Каково назначение представлений в шаблоне MVC?

5.11. Для чего применяются делегаты в MVC-приложениях?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ж. Бланшет. Qt 3: программирование GUI на C++ / Бланшет Ж., Саммерфилд М. — М. : КУДИЦ — ОБРАЗ, 2005. - 448 с.

2. Е.Р. Алексеев. Программирование на языке C++ в среде Qt Creator /Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк, О. В. Чеснокова, А. С. Чмыхало.—

М.:Альт Линукс, 2015. — 448 с.

3. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++/Г. Буч. — М. : БИНОМ ; СПб. : Невский диалект, 2001. - 560 с.

4. Шилдт, Г. C++: базовый курс, 3-е издание /Г. Шилдт. — М.: «Вильямс», 2012. — 624 с.

5. Шилдт, Г. Полный справочник по C++, 4-е издание /Г. Шилдт. — М.: «Вильямс», 2011. — 800 с.

ПРИЛОЖЕНИЕ – Варианты заданий

Таблица 1 — Варианты заданий

Вариант	Описание
1	Внешний вид приложения показан на рисунке 1. При нажатии на кнопку «Сравнить», осуществить сравнение текста в первом поле ввода с текстом во втором. Результат сравнения вывести соответствующим сообщением в текстовую метку («Текст совпадает» или «Текст не совпадает»)
2	Внешний вид приложения показан на рисунке 2. Изначально все кнопки, кроме первой, должны быть синего цвета. Первая – красного. При нажатии на кнопку красного цвета, она должна изменить свой цвет на синий. При этом любая другая случайно выбранная кнопка должна изменить свой цвет на красный.

Номер варианта определить следующим образом:

Последняя цифра зачетной книжка (вариант выданный преподавателем) % 2 + 1.

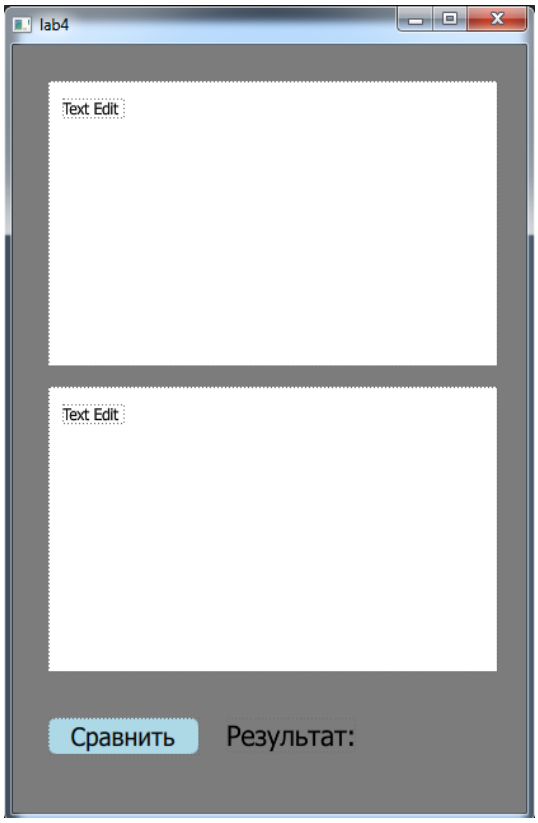

Рисунок 1 – Внешний вид графического интерфейса для варианта 1

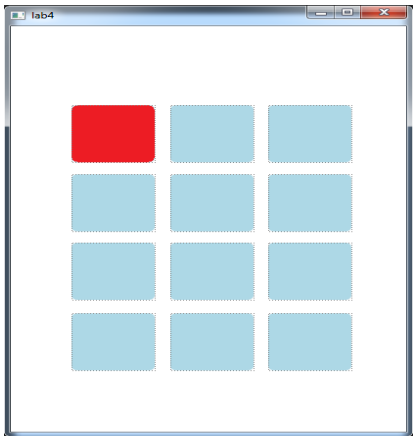

Рисунок 2 – Внешний вид графического интерфейса для варианта 2

Заказ № _____ от «_____» _____ 2018 г. Тираж экз.