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

ИССЛЕДОВАНИЕ СПОСОБОВ ИНТЕГРАЦИИ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ НА ЯЗЫКЕ QML И ФУНКЦИОНАЛЬНОСТИ НА ЯЗЫКЕ C++

Методические указания

к лабораторной работе 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 интеграции интерфейса пользователя на языке QML и функциональности на языке C++. Методические указания/Сост. В.А.Строганов. – Севастополь: Изд-во СевГУ, 2018.–11 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 4
2. Основные теоретические положения 4
2.1. Пример использования функциональности C++ в QML-приложении 5
3. Порядок выполнения лабораторной работы 9
4. Содержание отчета 9
5. Контрольные вопросы 10
Библиографический список 10
Приложение 10

1. ЦЕЛЬ РАБОТЫ

Исследование способов взаимодействия языка C++ и языка разметки QML. Приобретение навыков разработки приложений на основе QML-интерфейса

2. ОСНОВНЫЕ ПОЛОЖЕНИЯ

После создания разметки приложения необходимо добавить определенную функциональность.

Qt Quick позволяет QML вызывать методы, написанные на C++, и дает возможность обрабатывать C++-сигналы при помощи выражений на JavaScript через контекст QML.

Для передачи данных из интерфейса приложения на серверную часть для последующей обработки необходимо обеспечить QML доступ к методам и свойствам серверных классов. Это достигается при помощи макросов Q_PROPERTY (для свойств) и Q_INVOKABLE (для методов).

Формат макроса Q_PROPERTY имеет следующий вид:

```
Q_PROPERTY(type name
    READ getFunction
    [WRITE setFunction]
    [RESET resetFunction]
    [NOTIFY notifySignal]
    [DESIGNABLE bool]
    [SCRIPTABLE bool]
    [STORED bool]
    [USER bool]
    [CONSTANT]
    [FINAL])
```

Определяется тип и имя свойства, а также опционально:

- функции для чтения записи и сброса значения,
- сигнал изменения свойства,
- видимость в редакторе,
- доступность в механизме сценариев,
- должно ли записываться значение свойства при сохранении состояния объекта,
- может ли свойство редактировать пользователем,
- константное ли свойство,
- может ли свойство быть перегружено в наследнике.

Макрос Q_INVOKABLE просто указывается перед типом возвращаемого значения.

2.1. Пример использования функциональности C++ в QML-приложении

Для калькулятора созданного в предыдущей лабораторной работе добавим кнопки M+, M-, MR, MC реализующие стандартный функционал (суммирование с сохраненным результатом, вычитание, вывод в поле результата и очистка результата). Сохраненный результат будем хранить в файле, чтобы была возможность получить его после случайного закрытия файла.

1. Создаем класс, реализующий необходимый функционал

```
#ifndef CALCULATOR_H
#define CALCULATOR_H

#include <QtGui/QGuiApplication>
#include <QFile>
#include <QTextStream>

class Calculator : public QObject
{
    Q_OBJECT

public:
    Calculator(QObject *parent = 0);

    int curResult;

    Q_PROPERTY(int curResult READ getResult)

    Q_INVOKABLE void add(QString result);
    Q_INVOKABLE void remove(int result);
    Q_INVOKABLE void clear();
    Q_INVOKABLE int getResult();

    ~Calculator();

private:
    QString FILENAME;

};

#endif // CALCULATOR_H

-------------------------------------------------

#include "calculator.h"

Calculator::Calculator(QObject *parent):
    QObject(parent)
{
    this->FILENAME = QString("results.txt");
```

```
}

Calculator::~Calculator()
{
}

void Calculator::add(QString result)
{
    QFile file(FILENAME);
    curResult = getResult();
    if (file.open(QFile::ReadWrite | QIODevice::Truncate)) {
        QTextStream inStream(&file);
        curResult += result.toInt();
        QTextStream outStream(&file);
        outStream << curResult;
    }
    file.close();
}

void Calculator::remove(int result)
{
    QFile file(FILENAME);
    curResult = getResult();
    if (file.open(QFile::ReadWrite | QIODevice::Truncate)) {
        QTextStream inStream(&file);
        curResult -= result;
        QTextStream outStream(&file);
        outStream << curResult;
    }
    file.close();
}

void Calculator::clear()
{
    QFile file(FILENAME);
    if (file.open(QFile::ReadWrite | QIODevice::Truncate)) {
        QTextStream outStream(&file);
        outStream << 0;
    }
    file.close();
}

int Calculator::getResult()
{
    QFile file(FILENAME);
    if (file.open(QFile::ReadWrite)) {
        QTextStream inStream(&file);
        curResult = inStream.readAll().toInt();
    }
    file.close();
    return curResult;
}
```

2. Редактируем main.cpp, добавляя объект реализованного класса в контекст Qt Quick

```
#include <QtGui/QGuiApplication>
#include <QQmlContext>
#include <QQmlEngine>
#include <QQuickView>
#include "qtquick2applicationviewer.h"
#include "calculator.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    QtQuick2ApplicationViewer viewer;


viewer.setMainQmlFile(QStringLiteral("qml/untitled/main.qml"));
    viewer.rootContext()->setContextProperty("calculator", new
Calculator());
    viewer.showExpanded();

    return app.exec();
}
```

3. Редактируем QML файл, добавляя кнопки и соответствующие вызовы функций

```
Button {
      id: mMinusButton
      x: 281
      y: 163
      width: 57
      height: 51
      buttonHeight: 99
      labelSize: 27
      label: "M-"
      buttonWidth: 100
      onButtonClick:
      {
          calculator.remove(outputEdit.text)
      }
  }

  Button {
      id: mrButton
      x: 281
      y: 301
      width: 57
      height: 51
      buttonHeight: 100
      labelSize: 25
```

```
        label: "MR"
        buttonWidth: 100
        onButtonClick:
        {
            outputEdit.text = calculator.getResult()
        }
    }
    Button {
        id: mPlusButton1
        x: 281
        y: 93
        width: 57
        height: 51
        buttonHeight: 99
        labelSize: 27
        buttonWidth: 100
        label: "M+"
        onButtonClick:
        {
            calculator.add(outputEdit.text)
        }
    }

    Button {
        id: mClearButton
        x: 281
        y: 232
        width: 57
        height: 51
        buttonHeight: 99
        labelSize: 27
        buttonWidth: 100
        label: "MC"
        onButtonClick:
        {
            calculator.clear()
        }
    }
```

4. Окно запущенного приложения

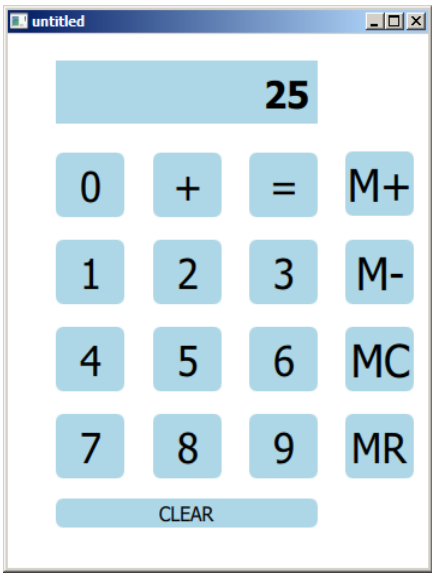

Рисунок 2.1 – Внешний вид приложения

3. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

3.1. Изучить способы организации взаимодействия QML и серверных класов на C++ (выполнется в ходе самостоятельной подготовки к лабораторной работе).

3.2. Разработать класс, реализующий функциональность по варианту задания, приведенному в Приложении.

3.3. Определить свойства и методы, необходимые для использования в QML разметке, с помощью соответствующих макросов.

3.4. Добавить класс в контекст Qt Quick приложения.

3.5. Дополнить разметку необходимыми элементами управления с вызовом соответствующих методов.

3.6. Исследовать эффективность работы полученного приложения, имитируя ошибки ввода/вывода.

3.7. Выполнить сравнительный анализ методов построения приложений в данной лабораторной работе и работе №3 по критерию трудоемкости проектирования и программирования.

4. СОДЕРЖАНИЕ ОТЧЕТА

4.1. Цель работы.

4.2. Постановка задачи.

4.3. Описание функциональности разработанного класса.

4.4. Текст программы.

4.5. Выводы.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1. Что понимается под макросом в C++?

5.2. Что необходимо сделать, чтобы обеспечить QML доступ к методам и свойствам серверных классов?

5.3. Какие макросы используются для предоставления QML доступа к свойствам серверного класса?

5.4. Какие макросы используются для предоставления QML доступа к методам серверного класса?

5.5.Назовите основные параметры макроса Q_PROPERTY. Для чего они используются?

5.6. Какие параметры имеет макрос Q_INVOKABLE?

5.7. Как добавить объект класса в контекст Qt Quick приложения?

5.8. Какие действия нужно выполнить для передачи данных из QML-интерфейса приложения на серверную часть?

5.9. Для чего необходимо взаимодействие QML с серверными классами, почему нельзя реализовать всю функциональность приложения в рамках QML?

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	Добавить кнопки, позволяющие сохранять и загружать текст из выбранного поля ввода.
2	Добавить две кнопки, по нажатию на первую текущее состояние игрового поля должно сохраняться в файл, по нажатию на вторую – загружаться из файла.

Номер варианта определить следующим образом:

Последняя цифра зачетной книжки (вариант, заданный преподавателем) % 2 + 1

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