

# Лекция 1 . Введение в Qt. Написание простого графического приложения. Понятия signal-slot

**Автор: Набиев Марат**

## Введение.

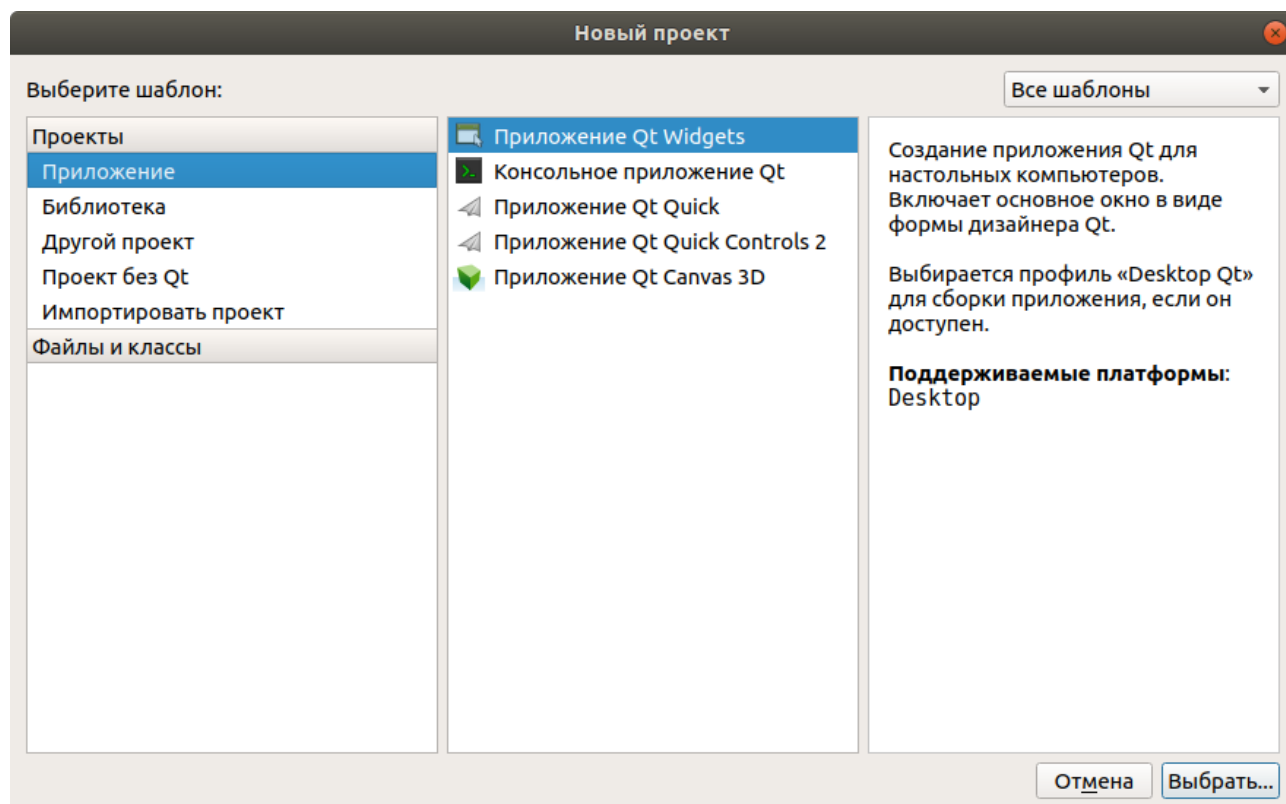
Qt — кроссплатформенный фреймворк для разработки ПО на C++. Также есть привязки к другим языкам: PyQt (Python), QtRuby (Ruby), Qt-Jambi (Java), PHP-Qt (PHP).

Так же на Qt написаны многие оболочки Linux. Qt появился в далеком 1996 году, изначально задумываясь как удобное средство для быстрой разработки графических интерфейсов на C++

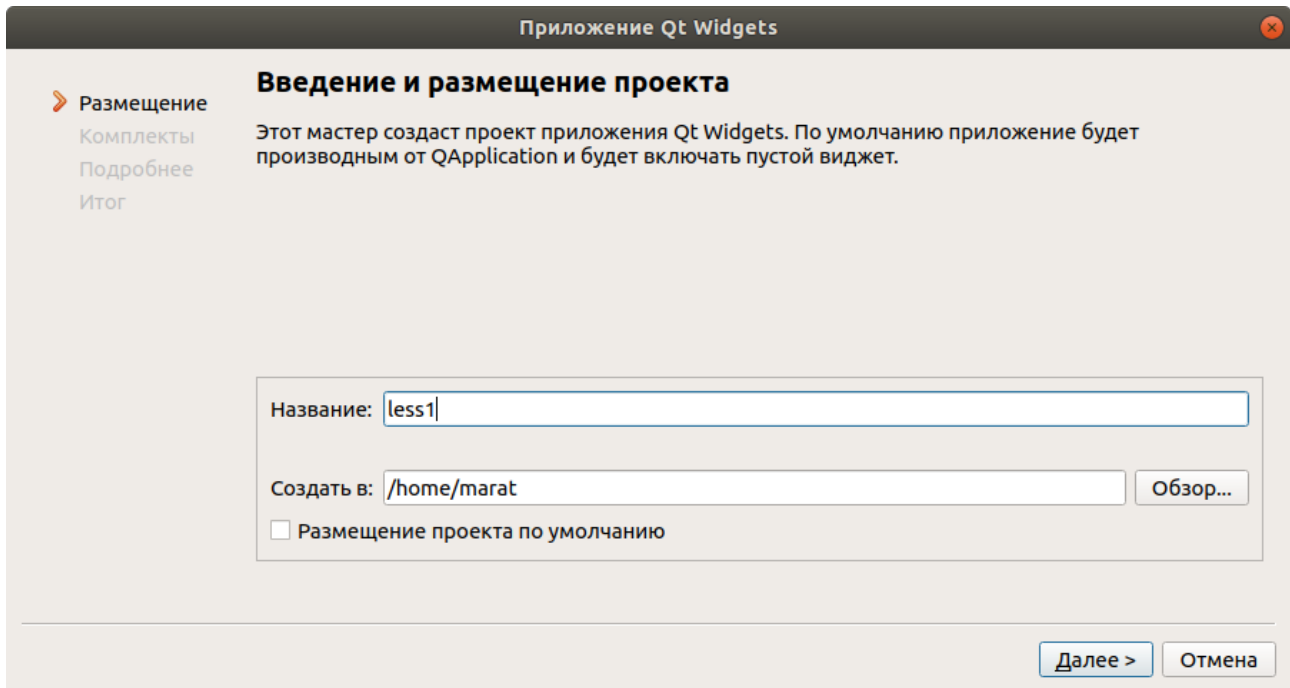
## Создание простейшего графического приложения.

Открываем Qt Creator — это удобная ide от Qt, нажимаем «новый проект».

Выбираем «приложение» → Приложение Qt Widgets → Выбрать

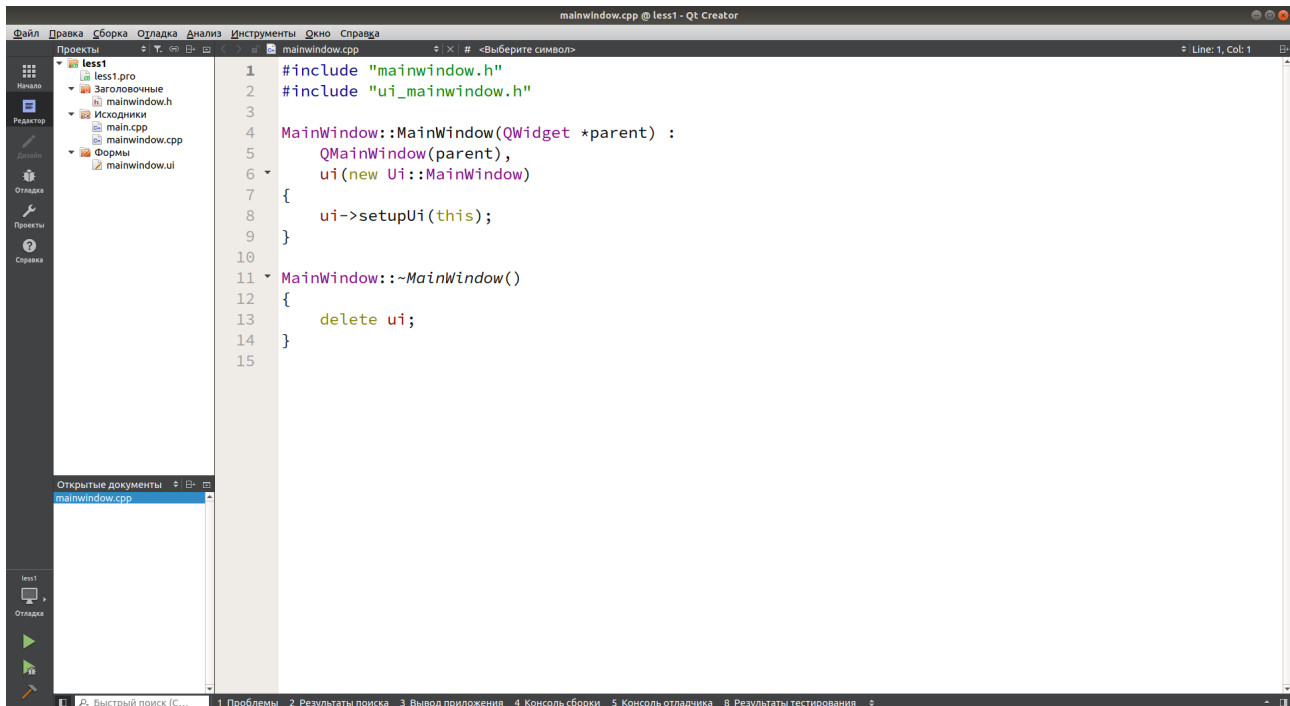


## Пишем название



потом далее, далее готово.

## Открывается следующее окошко



Если откроем `main.cpp`, то там будет следующий код

```
1  #include "mainwindow.h"
2  #include <QApplication>
3
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      MainWindow w;
8      w.show();
9
10     return a.exec();
11 }
12
```

Здесь `QApplication` — это организатор сбора сообщений всех графических элементов.

Если открыть `mainwindow.h`

```
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     explicit MainWindow(QWidget *parent = 0);
16     ~MainWindow();
17
18 private:
19     Ui::MainWindow *ui;
20 };
21
```

То мы здесь видим, что наше основное окно `MainWindow` является наследником `QMainWindow`, который является базовым классом для окон в Qt.

Также есть варианты `QWidget` , который предоставляет минимальный набор для работы. И еще есть `QDialog`, который является базовым классом для создания модальных окон.

Мы еще здесь видим макрос `Q_OBJECT` - при помощи которого можно генерировать и обрабатывать события, сильно заострять внимания пока на нем не стоит.

### **Добавление элементов на окно.**

Есть 2 способа добавления элементов на окно

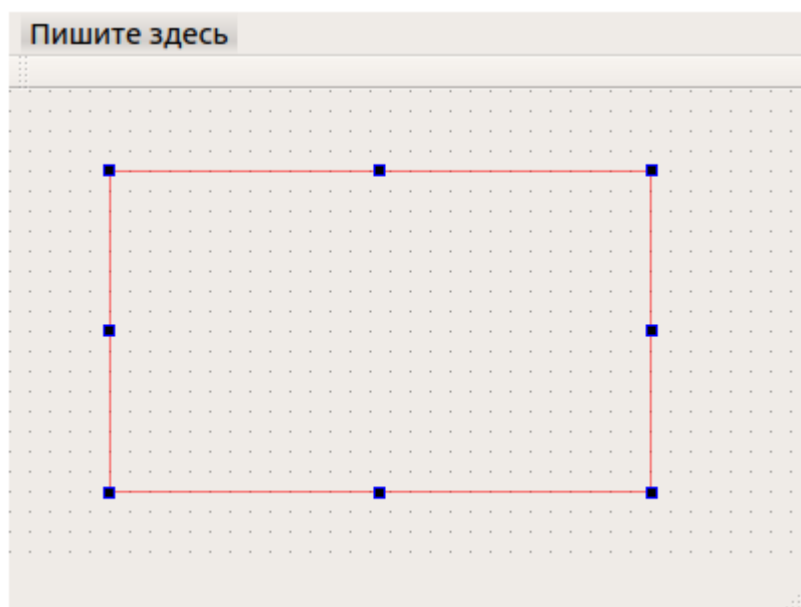
- 1) через Qt Designer, там можно перетаскивать как в шарповских формочках
- 2) прописывать все элементы прямо в коде.

Конечно, если у вас окно не меняется, логично создавать в конструкторе, а если меняется, то писать создание и удаление элементов в коде.

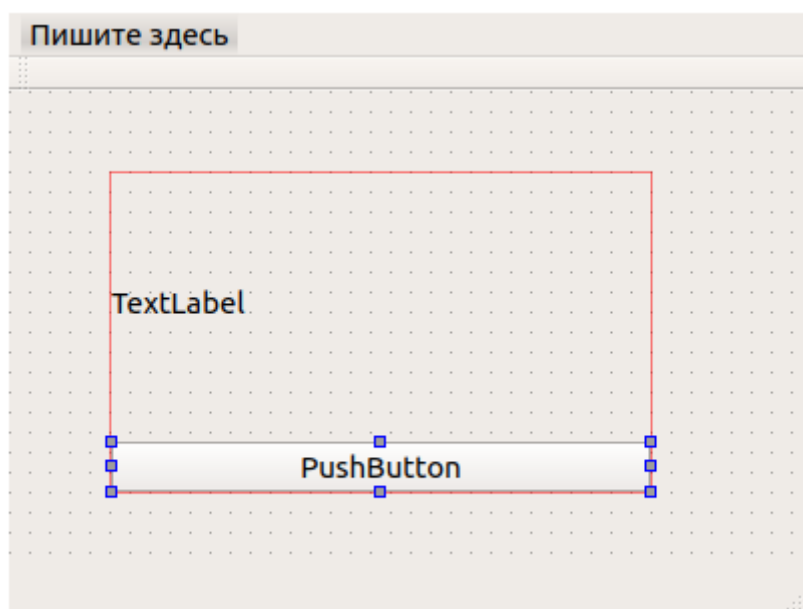
Для редактирования окна с помощью конструктора открываем `mainwindow.ui`

```
//окно бы я вставил, но что-то оно не вставляется, надеюсь, у вас открылось
```

Для того, чтобы у нас элементы были упорядочены по вертикали и мы не заботились об их размерах, добавим `Vertical Layout`.



Также добавим туда `Label` и `PushButton` (перетаскиваем прямо на `Layout`)



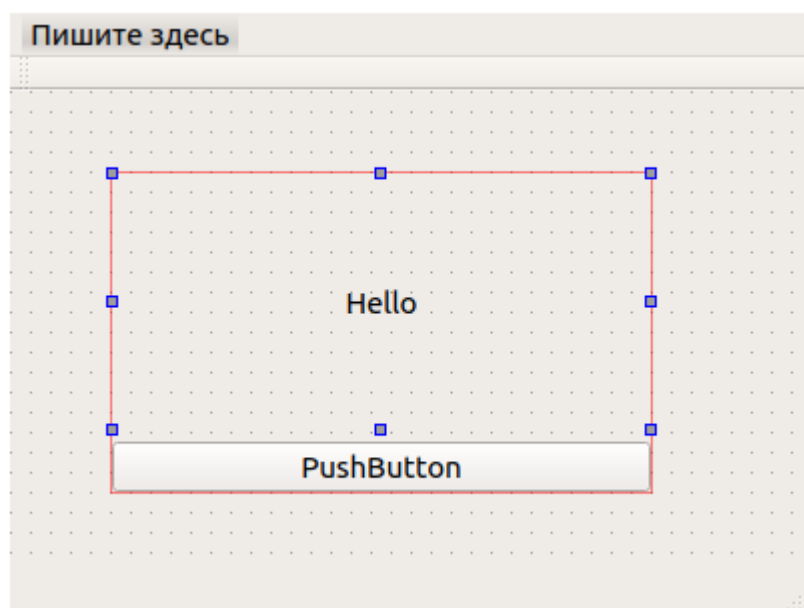
Сделаем текст на `Label` по-середине и изменим текст на нем, для этого нажимаем ЛКМ на наш `Label` и в панели справа меняем свойства

label : QLabel	
Свойство	Значение
autoFillBackgro...	<input type="checkbox"/>
styleSheet	
▶ locale	Russian, Russia
▶ inputMethodHi...	ImhNone
▼ QFrame	
frameShape	NoFrame
frameShadow	Plain
lineWidth	1
midLineWidth	0
▼ QLabel	
▶ text	Hello
textFormat	AutoText
pixmap	
scaledContents	<input type="checkbox"/>
▼ alignment	AlignLeft, AlignVCenter
Горизонтал...	AlignLeft
Вертикальное	AlignHCenter
wordWrap	AlignRight
margin	AlignJustify
indent	
openExternalLi...	<input type="checkbox"/>

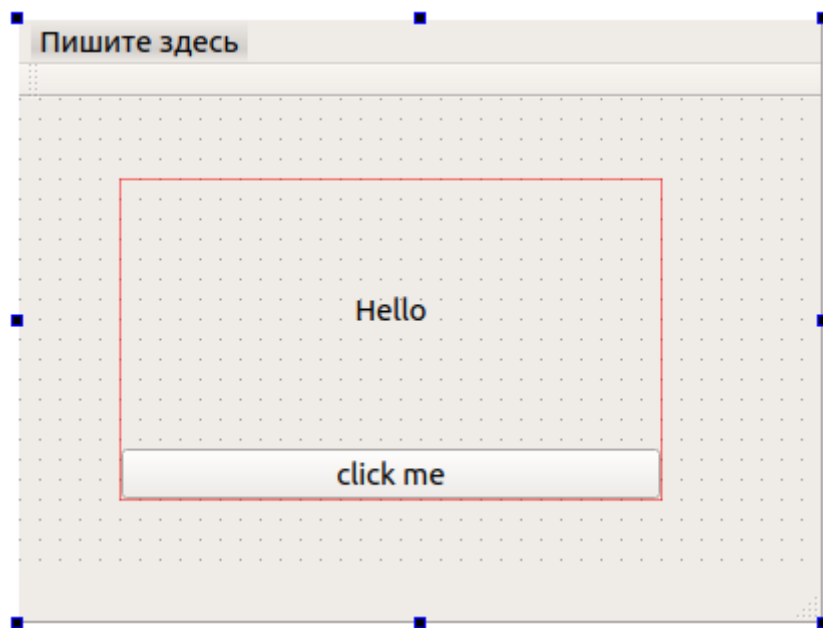
Делаем так:

Фильтр	
label : QLabel	
Свойство	Значение
autoFillBackgro...	<input type="checkbox"/>
styleSheet	
▸ locale	Russian, Russia
▸ inputMethodHi...	ImhNone
▼ QFrame	
frameShape	NoFrame
frameShadow	Plain
lineWidth	1
midLineWidth	0
▼ QLabel	
▸ text	Hello
textFormat	AutoText
pixmap	
scaledContents	<input type="checkbox"/>
▼ alignment	AlignHCenter, AlignVCenter
Горизонтал...	AlignHCenter
Вертикальное	AlignVCenter
wordWrap	<input type="checkbox"/>
margin	0
indent	-1
openExternalLi...	<input type="checkbox"/>

И у нас текст посередине



Аналогично можно изменить текст на кнопке и будет так:



Далее, мы хотим чтобы при нажатии на кнопку, текст на метке изменялся, а именно Hello → Привет → Сэлэм и так по кругу. Для этого в `MainWindow` заведем счетчик нажатий и в зависимости от него будем менять текст на метке. Далее мы переходим плавно к понятиям сигналы и слоты.

## **SIGNALS and SLOTS**

Это достаточно удобный механизм для событийного программирования. Попробую объяснить на примере:

При нажатии на кнопку, она издает сигнал `clicked` (т. е. она всем кричит: меня нажали! ), и мы хотим выполнять какие-то действия после ее нажатия. Мы пишем функцию, которая называется слот и коннектим слот к этому сигналу. Посмотрим, как это выглядит в коде.

Открываем `mainwindow.h` и там добавляем наш счетчик нажатий и добавляем слот для обработки сигнала кнопки

```

10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     explicit MainWindow(QWidget *parent = 0);
16     ~MainWindow();
17
18 private:
19     Ui::MainWindow *ui;
20     //наш счетчик нажатий
21     int clickCount;
22     //закрытый слот, т.е. из другого класса не могут его вызвать
23 private slots:
24     void buttonClick();
25 };
26

```

Далее нам надо заставить наш слот реагировать на нажатие кнопки. Для этого в конструкторе `MainWindow` пишем вот этот коннект: `connect(elem, SIGNAL(elemSignal()), elem2, SLOT(elem2Slot()))`

Это старый вид записи. У нас это будет выглядеть следующим образом:

```

4 MainWindow::MainWindow(QWidget *parent) :
5     QMainWindow(parent),
6     ui(new Ui::MainWindow)
7 {
8     ui->setupUi(this);
9     //для обращения к элементам, которые мы добавили в конструкторе пишем
10    //ui->element
11
12    connect(ui->pushButton, SIGNAL(clicked(bool)),
13           this, SLOT(buttonClick()));
14 }
15

```



Так же есть новый вид, который советуют использовать и он выглядит так:

```
4  MainWindow::MainWindow(QWidget *parent) :
5      QMainWindow(parent),
6      ui(new Ui::MainWindow)
7  {
8      ui->setupUi(this);
9      //для обращения к элементам, которые мы добавили в конструкторе пишем
10     //ui->element
11
12     // connect(ui->pushButton, SIGNAL(clicked(bool)),
13     //         this, SLOT(buttonClick()));
14     connect(ui->pushButton, &QPushButton::clicked,
15            this, &MainWindow::buttonClick);
16 }
17
```

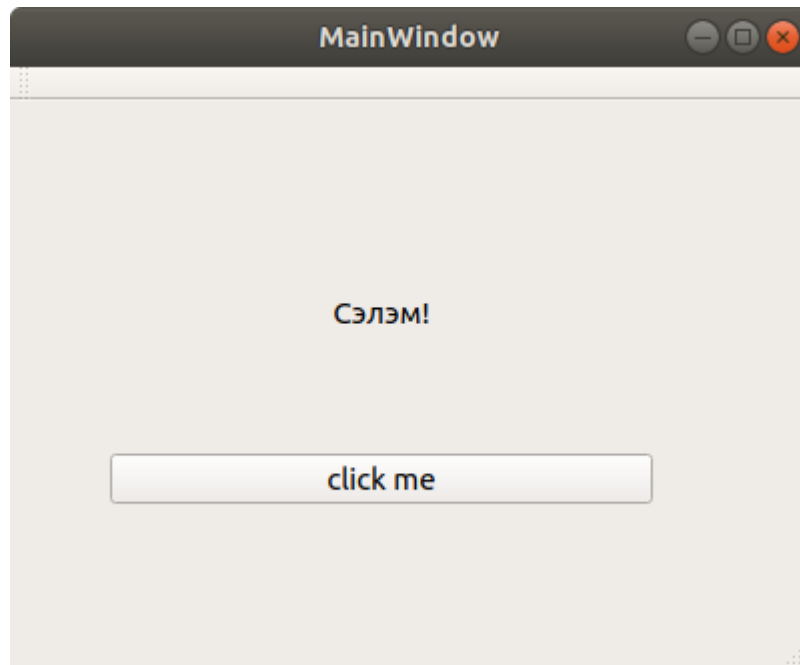
сами выбирайте какой хотите использовать.

Далее, для того, чтобы мы могли обрабатывать нажатие, надо реализовать этот buttonClick.

И надо не забыть в конструкторе обнулить счетчик нажатий. (напишите сами)

```
25 void MainWindow::buttonClick()
26 {
27     //чтобы не переполнялось
28     clickCount = (clickCount+1)%3;
29     switch (clickCount)
30     {
31         //меняем текст метки
32         case 0: ui->label->setText("Hello!"); break;
33         case 1: ui->label->setText("Привет!"); break;
34         case 2: ui->label->setText("Сэлэм!"); break;
35     }
36 }
```

и если все правильно, то при запуске будет появляться следующее окошко:



И при нажатии на кнопку будет меняться текст метки.

Далее пример сигнала. Сделаем так, чтобы когда у нас текст метки менялся на татарский, мы выводили сообщение при помощи `QMessageBox`. Для этого надо в `MainWindow` добавить сигнал, назовем его `makedTatarText()` и для его обработки напишем слот `tatarText()`

Добавляем сигнал и слот:

```
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     explicit MainWindow(QWidget *parent = 0);
16     ~MainWindow();
17
18 private:
19     Ui::MainWindow *ui;
20     //наш счетчик нажатий
21     int clickCount;
22     //закрытый слот, т.е. из другого класса не могут его вызвать
23 private slots:
24     void buttonClick();
25     void tatarText();
26     //объявили сигнал.
27 signals:
28     void makedTatarText();
29 };
```

Коннектим сигнал к слоту в конструкторе:

```
4 MainWindow::MainWindow(QWidget *parent) :
5     QMainWindow(parent),
6     ui(new Ui::MainWindow)
7 {
8     ui->setupUi(this);
9     //для обращения к элементам, которые мы добавили в конструкторе пишем
10    //ui->element
11
12    // connect(ui->pushButton, SIGNAL(clicked(bool)),
13    //         this, SLOT(buttonClick()));
14    connect(ui->pushButton, &QPushButton::clicked,
15           this, &MainWindow::buttonClick);
16    clickCount=0;
17    //коннектим наш сигнал к нашему слоту
18    connect(this, &MainWindow::makedTatarText,
19           this, &MainWindow::tatarText);
20
21 }
```

немного подкорректируем наш buttonClick:

```

28 void MainWindow::buttonClick()
29 {
30     //чтобы не переполнялось
31     clickCount = (clickCount+1)%3;
32     switch (clickCount)
33     {
34         //меняем текст метки
35         case 0: ui->label->setText("Hello!"); break;
36         case 1: ui->label->setText("Привет!"); break;
37         case 2:
38             ui->label->setText("Сэлэм!");
39             //кричим всем, что у нас текст на татарском стал
40             emit makedTatarText();
41             break;
42     }
43 }

```

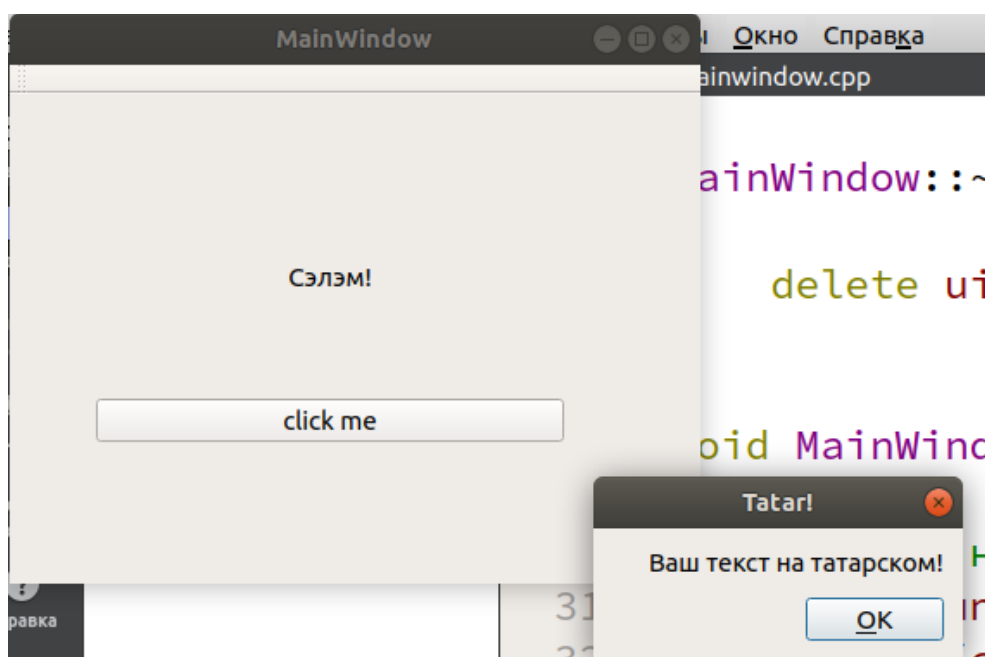
Пишем слот tatarText

```

44 void MainWindow::tatarText()
45 {
46     QMessageBox::about(NULL, "Tatar!", "Ваш текст на татарском!");
47 }

```

и если к этому моменту я ничего не забыл, и вы ничего не забыли, то когда у нас текст становится на татарском, что выходит еще одно окошко:



## MenuBar и Action-ы

Все мы знаем, что такое менюбары, так что не буду писать его важность. QAction — такой объект, который предоставляет абстрактное действие пользовательского интерфейса. Сделаем такой QAction, которой будет выводить окно перед закрытием. Для этого в MainWindow добавим closeAction (не забудьте подключить вначале написать #include<QAction>), добавим метод, в котором создадим closeAction, чтобы не загромождать конструктор и добавим слот, который будет выполняться при активации closeAction.

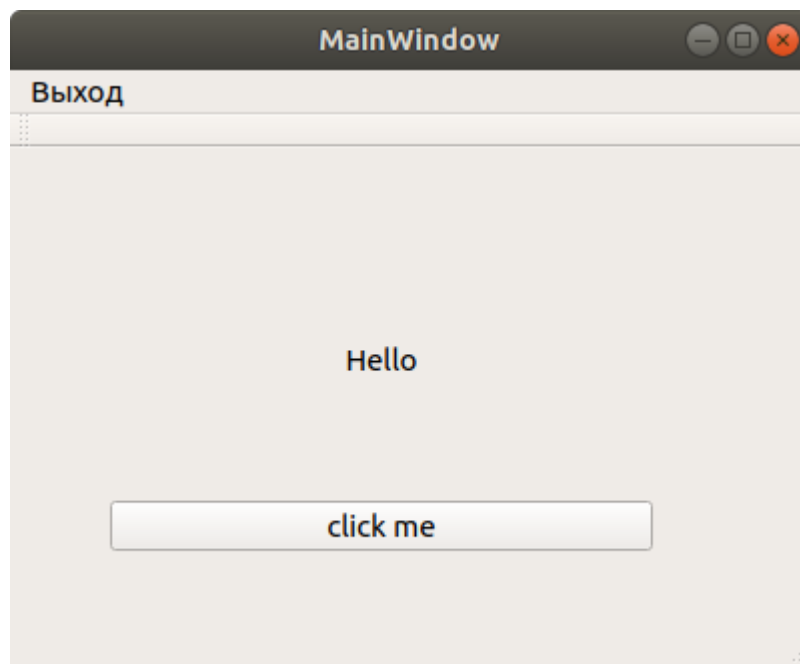
```
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     explicit MainWindow(QWidget *parent = 0);
16     ~MainWindow();
17
18 private:
19     Ui::MainWindow *ui;
20     //наш счетчик нажатий
21     int clickCount;
22     //наше закрывающее действие
23     QAction *closeAction;
24     //отдельный метод, в котором будем инициализировать action-ы
25     void initActions();
26     //закрытый слот, т.е. из другого класса не могут его вызвать
27 private slots:
28     void buttonClick();
29     void tatarText();
30     //слот для закрытия окна
31     void closeWindow();
32     //объявили сигнал.
33 signals:
34     void makedTatarText();
35 };
36
```

В `mainwindow.cpp` добавляем `initActions` и слот `closeWindow`, и не забудьте вызвать `initActions` в конструкторе

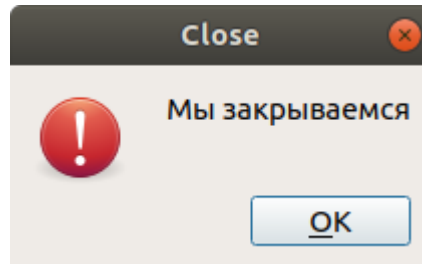
```
49 void MainWindow::initActions()
50 {
51     closeAction = new QAction("Выход");
52     //добавили горячую клавишу
53     closeAction->setShortcut(QKeySequence::fromString("Ctrl+X"));
54     //коннектим активацию к нашему слоту
55     connect(closeAction, &QAction::triggered,
56             this, &MainWindow::closeWindow);
57     //добавляем в меню-бар
58     ui->menuBar->addAction(closeAction);
59 }
60 //слот, обрабатывающий активацию действия
61 void MainWindow::closeWindow()
62 {
63     //выдаем предупреждение
64     QMessageBox::warning(NULL, "Close", "Мы закрываемся");
65     //закрываем
66     close();
67 }
```

т. к. мы создали `closeAction` при помощи `new`, надо в деструкторе его удалить. ( напишите сами это)

Если все хорошо, то при запуске будет следующее окошко:



При нажатии на выход, у нас появится окошко с предупреждением, и при нажатии на Ок все закроется.



Добавим в меню-бар выпадающее меню File, чтобы закрыть можно было и оттуда. Для этого в MainWindow добавим Menu \*fileMenu и немного подкорректируем initActions.

```
11 class MainWindow : public QMainWindow
12 {
13     Q_OBJECT
14 public:
15     explicit MainWindow(QWidget *parent = 0);
16     ~MainWindow();
17 private:
18     Ui::MainWindow *ui;
19     //наш счетчик нажатий
20     int clickCount;
21     //наше закрывающее действие
22     QAction *closeAction;
23     //отдельный метод, в котором будем инициализировать action-ы
24     void initActions();
25     //наша менюшка
26     QMenu *fileMenu;
27     //закрытый слот, т.е. из другого класса не могут его вызвать
28 private slots:
29     void buttonClick();
30     void tatarText();
31     //слот для закрытия окна
32     void closeWindow();
33     //объявили сигнал.
34 signals:
35     void makedTatarText();
```

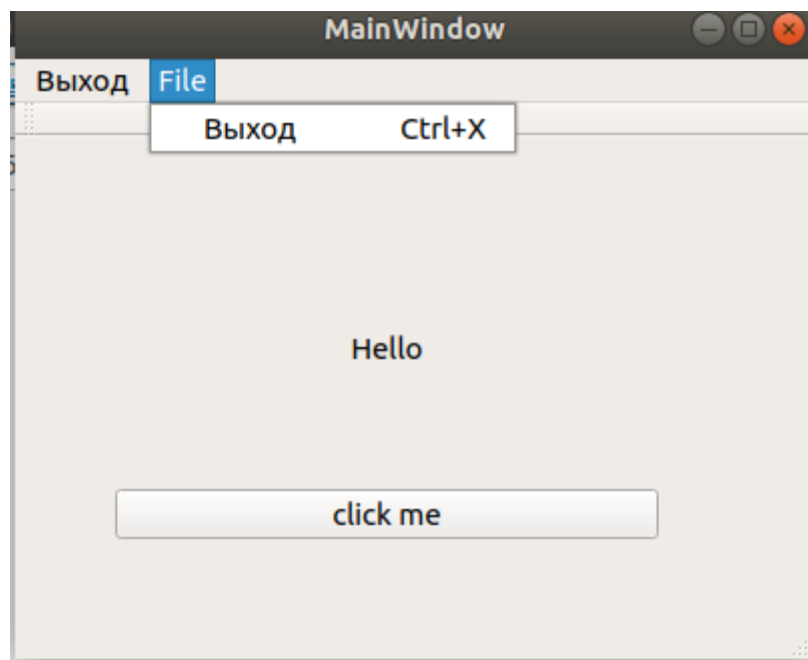
В initActions просим menuBar предоставить нам меню и добавляем туда action

```

50 void MainWindow::initActions()
51 {
52     closeAction = new QAction("Выход");
53     //добавили горячую клавишу
54     closeAction->setShortcut(QKeySequence::fromString("Ctrl+X"));
55     //коннектим активацию к нашему слоту
56     connect(closeAction, &QAction::triggered,
57             this, &MainWindow::closeWindow);
58     //добавляем в меню-бар
59     ui->menuBar->addAction(closeAction);
60
61     //просим создать menuBar менюшку для нас
62     fileMenu = ui->menuBar->addMenu("File");
63     //добавляем наш экшн в меню
64     fileMenu->addAction(closeAction);
65 }

```

и если запустим наше приложение, то у нас добавится в менюбаре меню File, и там будет наш action



На этом вроде все. Скоро выйдет и вторая лекция. Наверно, ближе к середине недели.

Ссылка на исходники <https://github.com/nma2207/qt-lessons/tree/master/less1>