

**Федеральное агентство связи
Федеральное государственное образовательное бюджетное учреждение высшего
профессионального образования
«Сибирский государственный университет телекоммуникаций и информатики»
(ФГОБУ ВПО «СибГУТИ»)**

Е. Ю. Мерзлякова

ЧЕЛОВЕКО-МАШИННОЕ ВЗАИМОДЕЙСТВИЕ

Методические указания к практическим занятиям

Новосибирск 2019

УДК

ктн Е. Ю. Мерзлякова

Человеко-машинное взаимодействие: Методические указания к практическим занятиям / Сиб. гос. ун-т телекоммуникаций и информатики. – Новосибирск, 2019. – с.

Методические указания предназначены для студентов технических специальностей, изучающих дисциплину «Человеко-машинное взаимодействие» и содержит описание практических работ.

Рисунков —, таблиц —. Список лит. – назв.

Кафедра прикладной математики и кибернетики.

Рецензент: дтн С. Н. Мамоиленко

Утверждено редакционно-издательским советом СибГУТИ
в качестве методических указаний.

© Сибирский государственный университет
телекоммуникаций и информатики, 2019 г.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №1

Цель: Знакомство со средой QtCreator, изучение механизма сигналов и слотов. Использование QAction.

Требование: Каждая подгруппа из 2-х человек выполняет общее задание.

На компьютерах установлена версия программы, расположенная по ссылке:
<http://download.qt.io/archive/qt/5.10/5.10.0/>

Задание:

1. Запустить QtCreator. Создать приложение Qt Widgets.

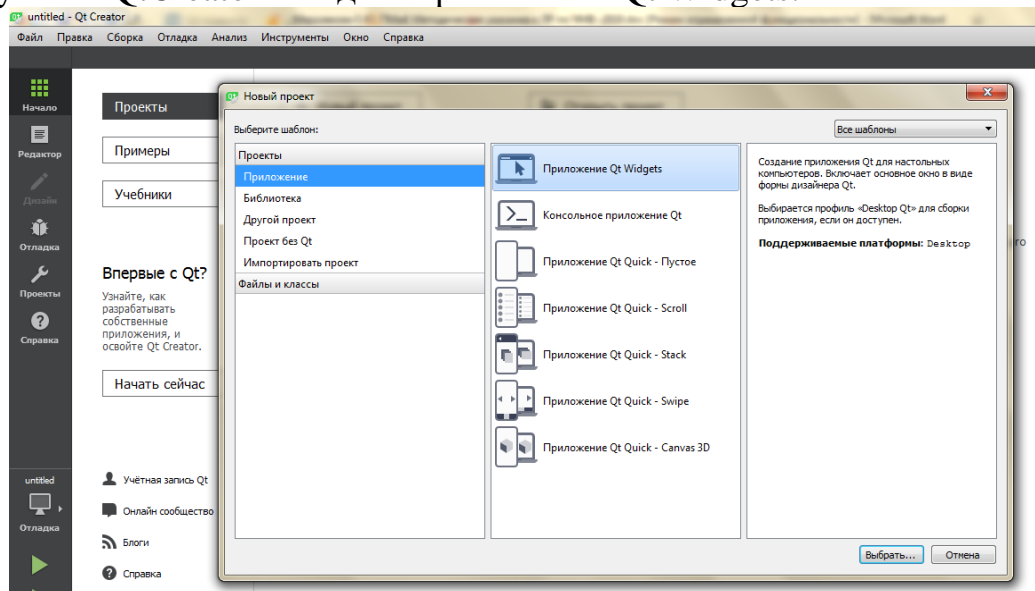


Рисунок 1. Создание приложения

2. Открыть форму MainWindow и создать 2 пункта главного меню: «Авторы» и «Выход».

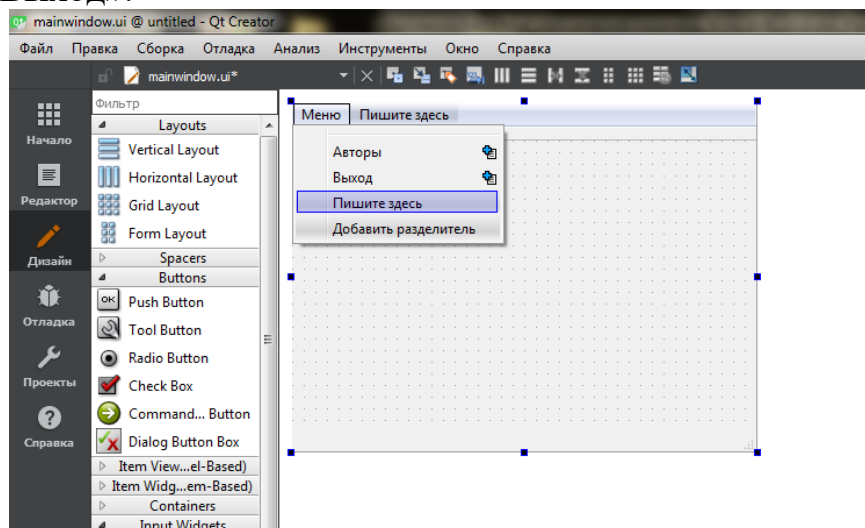


Рисунок 2. Создание меню на главной форме

- В свойстве формы «window title» задать заголовок окна «Лабораторная работа №1»:

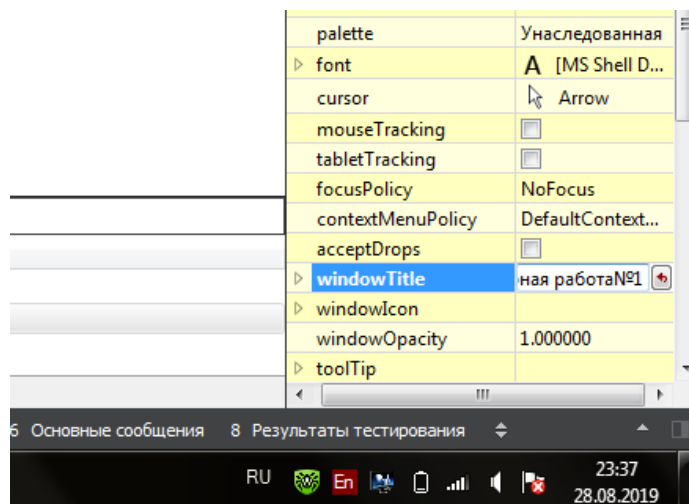


Рисунок 3. Создание заголовка главной формы

- В режиме редактирования сигналов и слотов установить соответствующий слот для меню «Выход». Для этого нужно выбрать режим «Изменение сигналов и слотов»:

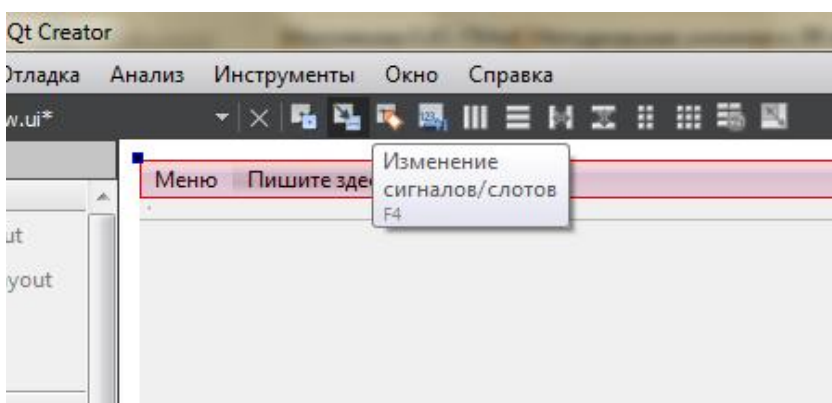


Рисунок 4. Режим изменения сигналов и слотов

Затем в нижнем окне выбрать «редактор сигналов и слотов», нажать на зеленый «плюс», создать связь между действием в меню «Выход» (action3) и слотом close() для окна приложения:

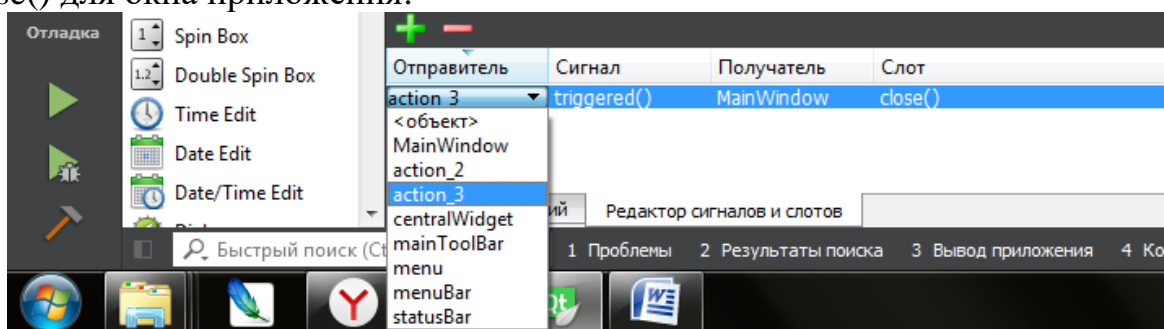


Рисунок 5. Изменение сигналов и слотов

5. Запустите приложение и проверьте его работу.

6. Теперь нужно создать вторую форму, на которой будет отображаться информация об авторах. Данная форма должна открываться при нажатии на пункт меню «Авторы».

Для начала нужно создать новый класс формы Qt Designer . Для этого в списке файлов проекта находим «формы», правой кнопкой мыши выбираем «Добавить новый», затем выбираем «Класс формы Qt Designer». Назовем, например, **auth** (об авторах):

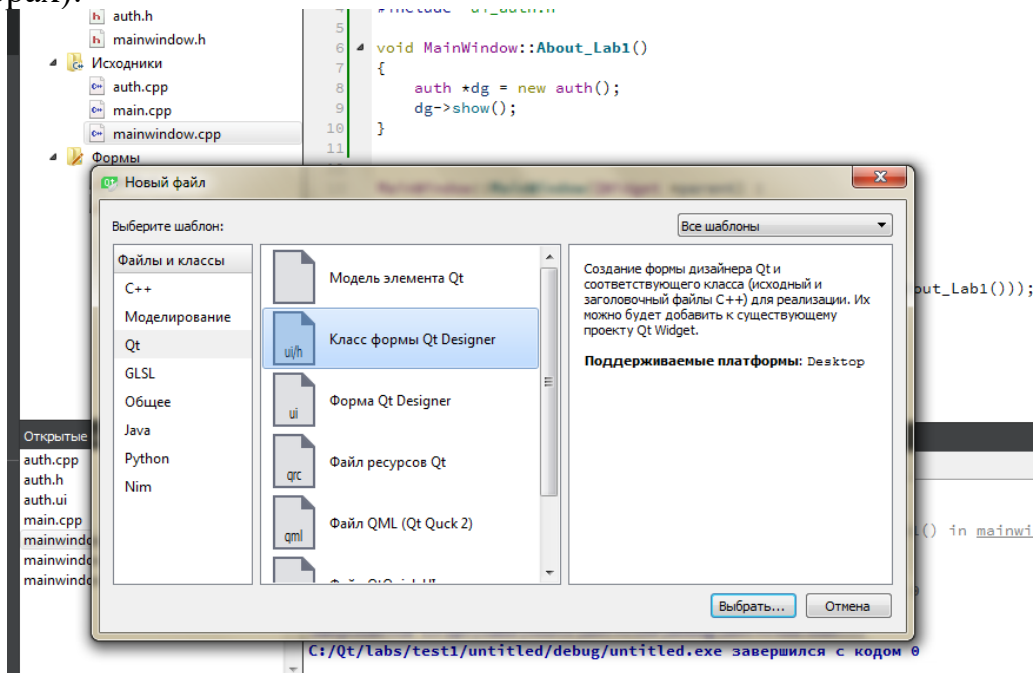


Рисунок 6. Создание новой формы

Затем в файле `mainwindow.cpp` нужно прописать процедуру открытия формы об авторах. Далее, пользуясь функцией `connect`, установить соединение со слотом `About_Lab1()` при выборе пункта меню «Авторы».



Рисунок 7. Слот About_Lab1()

Указать данный слот в заголовочном файле mainwindow.h:

```
8 }
9
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 private slots:
22     void About_Lab1();
23
24 };
25
26 #endif // MAINWINDOW_H
27
```

Рисунок 8. Файл mainwindow.h

7. Запустите приложение и проверьте его работу.

8. Установите заголовок окна второй формы «Информация об авторах». Расположите рисунок на форме. Для этого поместите на форму виджет Label и в его свойстве pixmap укажите файл с картинкой:

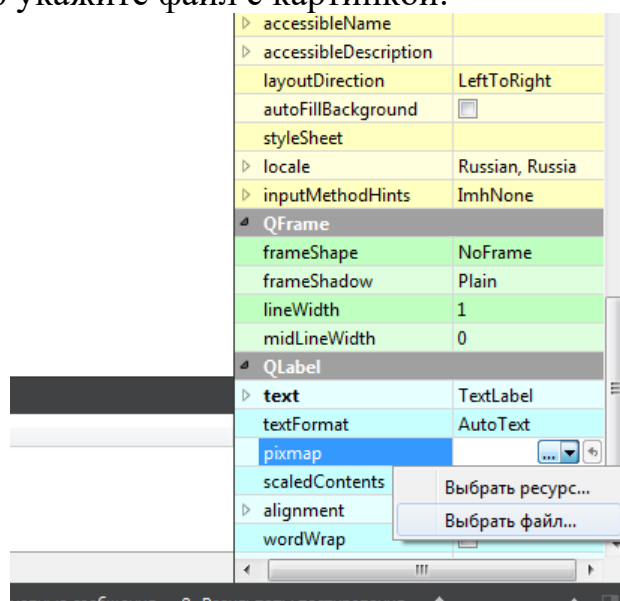


Рисунок 9. Выбор файла в Label

9. Разместите данные об авторах с помощью виджета TextEdit (двойным щелчком в нем открывается редактор текста):

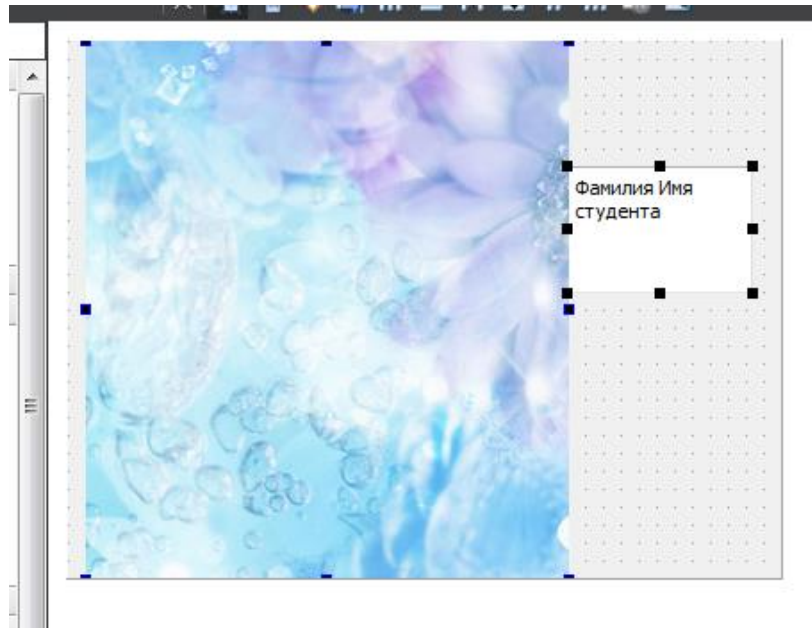


Рисунок 10. Информация об авторах

10. Запустите приложение и проверьте его работу.

11. Создадим еще один пункт меню главного окна, но другим способом. Пишем в файле `mainwindow.cpp`. Будем использовать класс действий `QAction`. Этот класс объединяет следующие элементы интерфейса пользователя:

- текст для всплывающего меню;
- текст для всплывающей подсказки;
- текст подсказки «Что это?»;
- «горячие клавиши»;
- ассоциированные значки;
- шрифт;
- текст строки состояния.

Для установки каждого из перечисленных элементов в объекте `QAction` существует свой метод. Метод `addAction()` позволяет внести объект действия в нужный виджет. В данном случае это виджет всплывающего меню `QMenu` (указатель `pmnuFile`) :

```
QAction* pactOpen = new QAction("file open action", 0);
pactOpen->setText("&Открыть");
pactOpen->setShortcut(QKeySequence("CTRL+S"));
pactOpen->setToolTip("Открытие документа");
pactOpen->setStatusTip("Открыть файл");
pactOpen->setWhatsThis("Открыть файл");
pactOpen->setIcon(QPixmap("1.png"));
connect(pactOpen, SIGNAL(triggered()), SLOT(slotOpen()));
QMenu* pmnuFile=new QMenu("&Файл");
pmnuFile->addAction(pactOpen);
menuBar()->addMenu(pmnuFile);
```

12. Затем нужно определить slotOpen. В этом слоте должно открываться диалоговое окно открытия файла. Содержимое файла загрузится в TextEdit (поместите его на главную форму). Обратите внимание на подключение библиотеки QFileDialog.

```
5  #include "QFileDialog"
6
7  void MainWindow::About_Lab1()
8  {
9      auth *dg = new auth();
10     dg->show();
11 }
12
13 void MainWindow::slotOpen()
14 {
15     QString filename = QFileDialog::getOpenFileName(0, "Открыть файл", QDir::currentPath(), "*.cpp *.txt");
16     QFile file(filename);
17     if (file.open(QIODevice::ReadOnly | QIODevice::Text))
18         ui->textEdit->setPlainText(file.readAll());
19 }
```

13. Запустите программу и проверьте ее работу.

14. Аналогично создайте действие в меню для сохранения файла. Слот сохранения текста из TextEdit выглядит так:

```
void MainWindow::slotSave()
{
    QString filename = QFileDialog::getSaveFileName(0, "Сохранить файл", QDir::currentPath(), "*.cpp *.txt");
    QTextDocumentWriter writer;
    writer.setFileName(filename);
    writer.write(ui->textEdit->document());
}
```

Здесь был создан объект поддержки записи (writer). Затем, установлено имя файла, взятое из диалогового окна открытия файлов. Вызов метода write() выполняет запись в файл, этот метод принимает в качестве параметра указатель на объект класса QTextDocument.

15. Создайте еще одно действие меню – «Очистить». Для очистки TextEdit воспользуйтесь методом clear().

16. Добавьте созданные действия в главную панель инструментов:

```
ui->mainToolBar->addAction(pactOpen);
ui->mainToolBar->addAction(pactSave);
ui->mainToolBar->addAction(pactClear);
```

17. Запустите приложение и проверьте его работу.

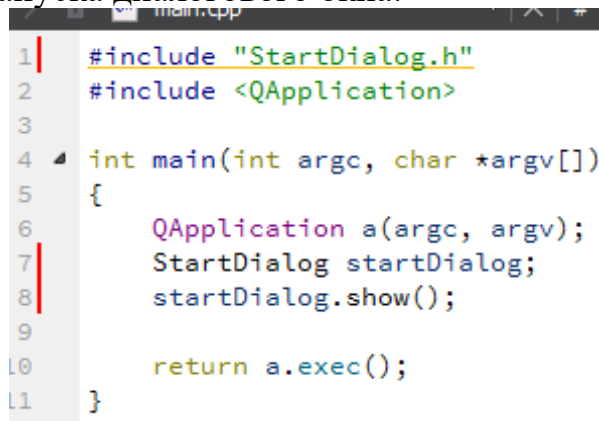
ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №2

Цель: Создание собственного диалогового окна, класс QDialog.

Требование: Каждая подгруппа из 2-х человек выполняет общее задание. В названиях создаваемых классов добавлять свои фамилии.

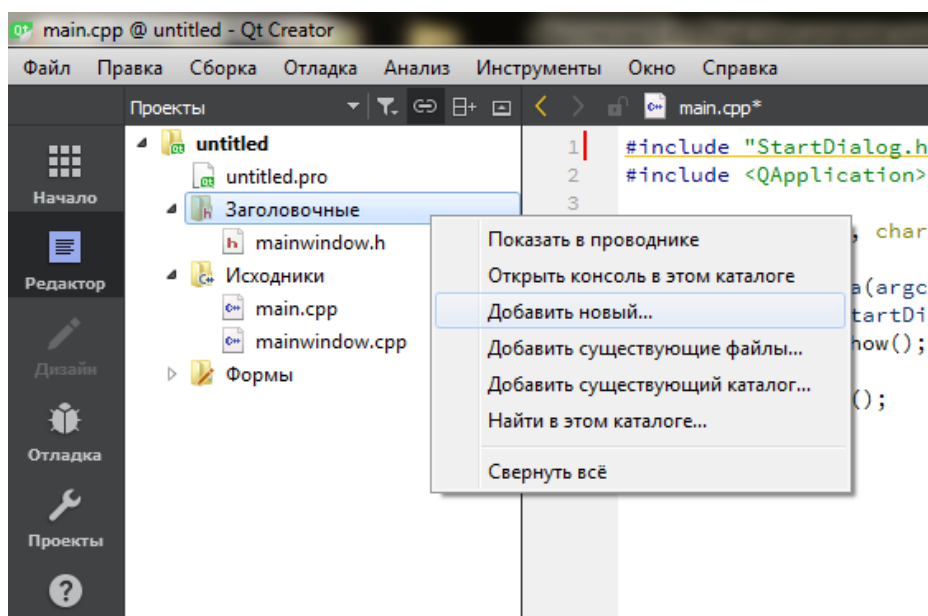
Задание:

1. Запустить QtCreator и создать приложение Qt Widgets. Приложение будет представлять из себя кнопку, по нажатию на которую отобразится диалоговое окно ввода имени и фамилии.
2. Создать виджет класса StartDialog (к названию добавить свои фамилии), предназначенный для запуска диалогового окна:



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

3. Класс StartDialog, приведенный в данном листинге, будет унаследован от класса кнопки QPushButton. Сначала создадим класс StartDialog:



В названии класса должны быть фамилии как в п.2

Определить класс

Имя класса: StartDialog_Ivanov_Petrov

Базовый класс: <Особый>

☐ Подключить QObject
☐ Подключить QWidget
☐ Подключить QMainWindow
☐ Подключить QDeclarativeItem - Qt Quick 1
☐ Подключить QQuickItem - Qt Quick 2
☐ Подключить QSharedData

Заголовочный файл: startdialog_ivanov_petrov.h

Файл исходных текстов: startdialog_ivanov_petrov.cpp

Путь: C:\Qt\abs\test2\untitled Обзор...

Далее Отмена

4. Класс StartDialog унаследован от класса кнопки QPushButton. Сигнал clicked() методом connect() соединяется со слотом slotButtonClicked():

```

StartDialog.h
1  #ifndef STARTDIALOG_H
2  #define STARTDIALOG_H
3
4  #include <QWidget>
5  #include <QPushButton>
6  #include <QMessageBox>
7  #include "InputDialog.h"
8
9  class StartDialog: public QPushButton {
10     Q_OBJECT
11
12 public:
13     StartDialog(QWidget* pwgt = 0) : QPushButton("Нажми", pwgt)
14     {
15         connect(this, SIGNAL(clicked()), SLOT(slotButtonClicked()));
16     }
17     public slots:
  
```

5. В слоте slotButtonClicked() создается объект диалогового окна InputDialog, который не имеет предка. Диалоговые окна, не имеющие предка, будут центрироваться на экране.

В условии оператора if выполняется запуск диалогового окна. После же его закрытия управление передается основной программе, и метод exes() возвращает значение нажатой пользователем кнопки.

В том случае, если пользователем была нажата кнопка Ok, будет отображено информационное окно с введенными в диалоговом окне данными:

```

StartDialog.h
14 {
15     connect(this, SIGNAL(clicked()), SLOT(slotButtonClicked())
16 }
17 public slots:
18     void slotButtonClicked()
19     {
20         InputDialog* pInputDialog = new InputDialog;
21         if (pInputDialog->exec() == QDialog::Accepted){
22             QMessageBox::information(0,
23                                     "Ваша информация: ",
24                                     "Имя: "
25                                     + pInputDialog->firstName()
26                                     + "\nФамилия: "
27                                     + pInputDialog->lastName()
28                                     );
29     }
30 }

```

6. По завершению метода диалоговое окно нужно удалить самому, так как у него нет предка, который позаботится об этом:

```

27         + pInputDialog->last
28         );
29     }
30     delete pInputDialog;
31 }
32 };
33
34 #endif // STARTDIALOG_H
35
36

```

7. Для создания своего собственного диалогового окна нужно унаследовать класс QDialog. Класс InputDialog (создайте его со своими фамилиями в названии) содержит два атрибута: указатели `m_ptxtFirstName`, `m_ptxtLastName` на виджеты однострочных текстовых полей и два метода, возвращающие содержимое этих полей: `firstName()` и `lastName()`:

```

InputDialog.h
1 #ifndef INPUTDIALOG_H
2 #define INPUTDIALOG_H
3
4 #include <QDialog>
5 #include <QLineEdit>
6
7 class QLineEdit;
8
9 class InputDialog: public QDialog
10 {
11     Q_OBJECT
12 private:
13     QLineEdit * m_ptxtFirstName;
14     QLineEdit * m_ptxtLastName;
15 public:
16     InputDialog(QWidget* pwgt = 0);
17
18     QString firstName() const;
19     QString lastName() const;
20 };
21
22 #endif // INPUTDIALOG_H
23

```

8. Модальное диалоговое окно всегда должно содержать кнопку Cancel. Сигналы clicked() кнопок Ok и Cancel соединяются со слотами accept() и rejected() соответственно. Это делается для того, чтобы метод exec() возвращал при нажатии кнопки Ok значение QDialog::Accepted, а при нажатии на кнопку Cancel – значение QDialog::Rejected

```
> InputDialog.cpp # InputDialog::InputDialog(QWidget *)
1  #include "InputDialog.h"
2  #include <QLabel>
3  #include <QLayout>
4  #include <QString>
5  #include <QPushButton>
6
7  InputDialog::InputDialog(QWidget* pwgt): QDialog(pwgt)
8  {
9      m_ptxtFirstName = new QLineEdit;
10     m_ptxtLastName = new QLineEdit;
11
12     QLabel* plblFirstName = new QLabel("&Имя");
13     QLabel* plblLastName = new QLabel("&Фамилия");
14
15     plblFirstName->setBuddy(m_ptxtFirstName);
16     plblLastName->setBuddy(m_ptxtLastName);
17
18     QPushButton* pcmdOk = new QPushButton("&Ok");
19     QPushButton* pcmdCancel = new QPushButton("&Cancel");
20
21     connect(pcmdOk, SIGNAL(clicked()), SLOT(accept()));
22     connect(pcmdCancel, SIGNAL(clicked()), SLOT(reject()));
```

9. Добавим менеджер компоновки:

```
22     connect(pcmdCancel, SIGNAL(clicked()), SLOT(reject()));
23
24     QGridLayout* ptopLayout = new QGridLayout;
25     ptopLayout->addWidget(plblFirstName,0,0);
26     ptopLayout->addWidget(plblLastName,1,0);
27     ptopLayout->addWidget(m_ptxtFirstName,0,1);
28     ptopLayout->addWidget(m_ptxtLastName,1,1);
29     ptopLayout->addWidget(pcmdOk,2,0);
30     ptopLayout->addWidget(pcmdCancel,2,1);
31     setLayout(ptopLayout);
32
33 }
```

10. Метод firstName() возвращает введенное пользователем имя:

```
33 }
34
35 InputDialog::firstName() const
36 {
37     return m_ptxtFirstName->text();
38 }
```

11. Добавьте аналогично метод для фамилии и запустите приложение.

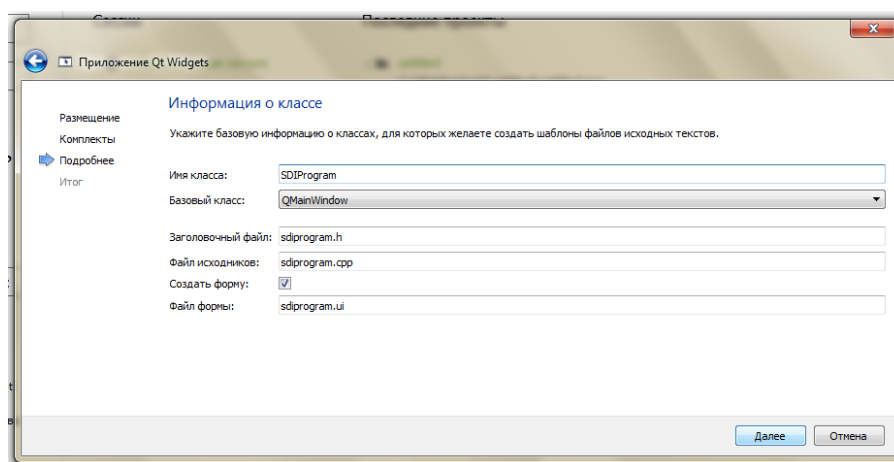
ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №3

Цель: Создание SDI- приложения (Single Document Interface, однодокументный интерфейс). Простой текстовый редактор. Окно заставки.

Требование: Каждая подгруппа из 2-х человек выполняет общее задание. В названиях создаваемых классов добавлять свои фамилии.

Задание:

1. В SDI-приложениях рабочая область одновременно является окном приложения, а это значит, что в одном и том же таком приложении невозможно открыть сразу два документа. Типичным примером SDI-приложения является программа Блокнот (Notepad) из состава ОС Windows. Реализуем упрощенный вариант этой программы – простой текстовый редактор. Запустите QtCreator и создайте приложение Qt Widgets. При этом, смените имя класса на **SDIProgram** (к названию добавить свои фамилии):



2. Создайте класс DocWindow (к названию добавить свои фамилии), унаследованный от класса QTextEdit. Он представляет собой окно для редактирования. В его определении содержится атрибут m_strFileName, в котором хранится имя изменяемого файла. Сигнал changeWindowTitle() предназначен для информирования о том, что текстовая область заголовка должна быть изменена. Файл DocWindow.h

```

1
2 #ifndef DOCWINDOW_H
3 #define DOCWINDOW_H
4
5 #include<QTextEdit>
6
7 class DocWindow: public QTextEdit {
8     Q_OBJECT
9 private:
10     QString m_strFileName;
11
12 public:
13     DocWindow(QWidget* pwgt = 0);
14 signals:
15     void changeWindowTitle(const QString&);
16
17 };
18
19 #endif // DOCWINDOW_H
20

```

3. Слоты slotLoad(), slotSave() и slotSaveAs() необходимы для проведения операций чтения и записи файлов:

```

15 void changewindowt
16
17
18 public slots:
19 void slotLoad();
20 void slotSave();
21 void slotSaveAs();
22
23 };
24

```

4. В конструктор класса (DocWindow.cpp) передается указатель на виджет предка:

```

1 #include "docwindow.h"
2
3 DocWindow::DocWindow(QWidget* pwgt): QTextEdit(pwgt)
4 {
5
6 }
7

```

5. Метод slotLoad() отображает диалоговое окно открытия файла вызовом статического метода QFileDialog::getOpenFileName(), с помощью которого пользователь выбирает файл для чтения. В том случае, если пользователь отменит выбор, нажав на кнопку Cancel, этот метод вернет пустую строку. В данном случае это проверяется с помощью метода QString::isEmpty():

```

7      }
8
9      void DocWindow::slotLoad()
10     {
11         QString str = QFileDialog::getOpenFileName();
12         if (str.isEmpty()){
13             return;
14         }
15     }

```

6. Если метод `getOpenFileName()` возвращает непустую строку, то будет создан объект класса `QFile`, проинициализированный этой строкой. Передача `QIODevice::ReadOnly` в метод `QFile::open()` говорит о том, что файл открывается только для чтения. В случае успешного открытия файла создается объект потока `stream`, который здесь используется для чтения текста из файла. Чтение всего содержимого файла выполняется при помощи метода `QTextStream::readAll()`, который возвращает его в объекте строкового типа `QString`. Текст устанавливается в виджете методом `setPlainText()`. После этого файл закрывается методом `close()`:

```

15
16     QFile file(str);
17     if(file.open(QIODevice::ReadOnly)){
18         QTextStream stream(&file);
19         setPlainText(stream.readAll());
20         file.close();
21     }
22

```

7. Об изменении местонахождения и имени файла оповещается отправкой сигнала `changeWindowTitle()`, для того чтобы использующее виджет `DocWindow` приложение могла отобразить эту информацию, изменив заголовок окна:

```

3
4     m_strFileName=str;
5     emit changeWindowTitle(m_strFileName);
6 }
7 }
8 }
9

```

8. Следующий слот `slotSaveAs()` отображает диалоговое окно сохранения файла с помощью статического метода `QFileDialog::getSaveFileName()`. Если пользователь не нажал в этом окне кнопку `Cancel`, и метод вернул непустую строку, то в атрибут `m_strFileName` записывается имя файла, указанное пользователем в диалоговом окне, и вызывается слот `slotSave()`:

```

25
26 void DocWindow::slotSaveAs()
27 {
28     QString str = QFileDialog::getSaveFileName(0, m_strFileName);
29     if (!str.isEmpty()){
30         m_strFileName=str;
31         slotSave();
32     }
33
34 }
35

```

9. Запись в файл представляет собой более серьезный процесс, чем считывание, так как она связана с рядом обстоятельств, которые могут сделать ее невозможной. Например, на диске не хватит места, или он будет не доступен для записи. Сначала проверим, не пустая ли строка с именем файла. В этом случае нужно вызвать другой слот:

```

35 void DocWindow::slotSave()
36 {
37     if(m_strFileName.isEmpty()){
38         slotSaveAs();
39         return;
40     }
41

```

10. Затем для записи в файл нужно создать объект класса QFile и передать в него строку с именем файла.

```

41
42     QFile file(m_strFileName);
43

```

11. Затем надо вызвать метод open(), передав в него значение QIODevice::WriteOnly (флаг, говорящий о том, что будет выполняться запись в файл). В том случае, если файла с таким именем на диске не существует, он будет создан. Если существует – он будет открыт для записи. Если файл открыт успешно, то создается промежуточный объект потока, в который при помощи оператора << передается текст виджета, возвращаемый методом toPlainText().

```

44
45     if(file.open(QIODevice::WriteOnly)){
46         QTextStream(&file)<<toPlainText();
47
48

```


12. После этого файл закрывается методом `QFile::close()`, и отсылается сигнал с новым именем и местонахождением файла. Это делается для того, чтобы эту информацию могло отобразить приложение, использующее наш виджет `DocWindow`.

```
48  
49  
50     file.close();  
51     emit changeWindowTitle(m_strFileName);  
52 }
```

13. Добавьте окно информационного сообщения после успешного сохранения файла.
14. Класс `SDIPProgram` унаследован от класса `QMainWindow`. В его конструкторе создаются три виджета: всплывающие меню `File` – указатель `pmnuFile`, `Help` – указатель `pmnuHelp` и виджет созданного вами окна редактирования – указатель `pdoc`.

```
> sdiprogram.h* SDIPProgram(QWidget * = 0)  
1  #ifndef SDIPROGRAM_H  
2  #define SDIPROGRAM_H  
3  
4  #include <QMainWindow>  
5  #include <QtWidgets>  
6  #include "docwindow.h"  
7  #include "sdiprogram.h"  
8  
9  class SDIPProgram : public QMainWindow  
10 {  
11     Q_OBJECT  
12  
13 public:  
14     SDIPProgram(QWidget *pwgt = 0): QMainWindow(pwgt)  
15     {  
16         QMenu* pmnuFile = new QMenu("&File");  
17         QMenu* pmnuHelp = new QMenu("&Help");  
18         DocWindow* pdoc = new DocWindow;  
19     }
```

15. Затем несколькими вызовами метода `addAction()` неявно создаются объекты действий и добавляются в качестве команд меню. Третьим параметром указывается слот, с которым должна быть соединена команда, во втором параметре указан сам объект, который содержит этот слот. Таким образом, команда `Open...` соединяется со слотом `slotLoad()`:

```

19
20 pmnuFile->addAction("&Open...",
21                      pdoc,
22                      SLOT(slotLoad()),
23                      QKeySequence("CTRL+O")
24                      );
25 pmnuFile->addAction("&Save",

```

16. Добавьте аналогично остальные четыре действия: команда Save – со слотом slotSave, а команда Save As... - со слотом slotSaveAs(). Все эти слоты реализованы в классе DocWindow (обратите внимание на второй параметр, pdoc). Затем добавьте разделитель в меню. Далее, команда Quit соединяется со слотом quit() (второй параметр уже будет QApplication).

Команда About должна находиться в Help (вместо pmnuFile пишем pmnuHelp), она соединяется со слотом slotAbout(), предоставляемым классом SDIProgram (поэтому второй параметр this). Быстрый доступ для About необходимо задать клавишей F1, указав в последнем параметре Qt::Key_F1.

17. Затем menuBar() возвращает указатель на виджет меню верхнего уровня, а вызов методов addMenu() добавляет созданные всплывающие меню File и Help:

```

48
49
50 menuBar()->addMenu(pmnuFile);
51 menuBar()->addMenu(pmnuHelp);
52

```

18. Вызов метода setCentralWidget() делает окно редактирования центральным виджетом, то есть рабочей областью вашей программы.

```

53
54 setCentralWidget(pdock);
55

```

19. Для изменения текстового заголовка программы после загрузки файла или сохранения его под новым именем сигнал changeWindowTitle(), отправляемый виджетом окна редактирования, соединяется со слотом slotChangeWindowTitle().

56
57
58
59
60
61
62

```
connect(pdoc,  
        SIGNAL(changeWindowTitle(const QString&)),  
        SLOT(slotChangeWindowTitle(const QString&))  
        );
```

20. Метод `showMessage()`, вызываемый из виджета строки состояния, отображает надпись `Ready` на время, установленное во втором параметре (это 2 сек.)

60
61
62
63
64

```
),  
  
    statusBar()->showMessage("Ready",2000);  
}
```

21. Осталось добавить слоты:

64
65
66
67
68
69
70
71
72
73
74
75
76
77
78

```
public slots:  
    void slotAbout()  
    {  
        QMessageBox::about(this, "Application", "SDI Example");  
    }  
  
    void slotChangeWindowTitle(const QString& str)  
    {  
        setWindowTitle(str);  
    }  
  
};
```

22. Теперь можно запустить программу и проверить ее работу.

23. Замените надпись в окне `About` на свои фамилии и группу.

24. Добавьте аналогичным образом еще один пункт в меню `File -> Color`. При выборе данного пункта должно открыться диалоговое окно выбора цвета. Затем выбранный цвет должен быть установлен для текста методом `setTextColor()`. После чего попробуйте напечатать текст.

25. Добавим окно заставки. В библиотеке `Qt` такое окно реализовано в классе `QSplashScreen`. Объект этого класса создается в функции `main()` до вызова метода `exec()` объекта приложения. В конструктор передадим растровое

изображение, которое будет отображаться после вызова метода show(). Самим приложением, которое должно быть запущено является w:

```
25 int main(int argc, char *argv[])
26 {
27     QApplication a(argc, argv);
28     QSplashScreen splash(QPixmap("s.png"));
29     splash.show();
30     SDIProgram w;
31     loadModules(&splash);
32     splash.finish(&w);
33     w.show();
34
35     return a.exec();
36 }
37
```

26. Функция loadModules() является эмуляцией загрузки модулей программы, в нее передается адрес объекта окна заставки, чтобы функция могла отображать информацию о процессе загрузки. Объект класса QTimer служит для того, чтобы значение переменной i увеличивалось только по истечении 40 мсек. Отображение информации выполняется при помощи метода showMessage(), в который первым параметром передается текст, вторым - расположение текста, а третьим – цвет текста. Вызов метода finish() закрывает окно заставки. В этот метод передается указатель на главное окно приложения, и появление этого окна приводит к закрытию окна заставки:

```

1  #include "sdiprogram.h"
2  #include <QApplication>
3  #include <QtWidgets>
4
5  void loadModules(QSplashScreen* psplash)
6  {
7      QTime time;
8      time.start();
9
10     for (int i=0; i<100; ){
11         if(time.elapsed()>40){
12             time.start();
13             ++i;
14         }
15
16         psplash->showMessage("Loading modules: "
17                             + QString::number(i)+"%",
18                             Qt::AlignCenter|Qt::AlignCenter,
19                             Qt::black
20                             );
21         qApp->processEvents();
22     }
23 }
24

```

27. Запустите приложение, проверьте его работу.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №4

Цель: Научиться проводить SWT-анализ

Требование: Выбрать программу, успешно выполняющую минимум две задачи. Программа должна иметь недостатки интерфейса, быть «не идеальной».

Задание: Провести SWT анализ двух задач выбранной программы. Отчет оформить на бумаге.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №5

Цель: Научиться проводить GOMS-анализ

Требование: Выбрать программу, успешно выполняющую минимум две задачи. Программа должна иметь недостатки интерфейса, быть «не идеальной».

Задание: Провести GOMS анализ двух задач выбранной программы. Отчет оформить на бумаге.

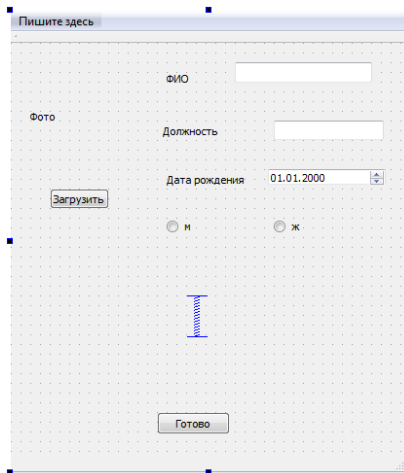
ЛАБОРАТОРНАЯ РАБОТА 6

Цель: Научиться обмену данными между формами. Использовать стили.

Требование: Каждая подгруппа из 2-х человек выполняет общее задание.

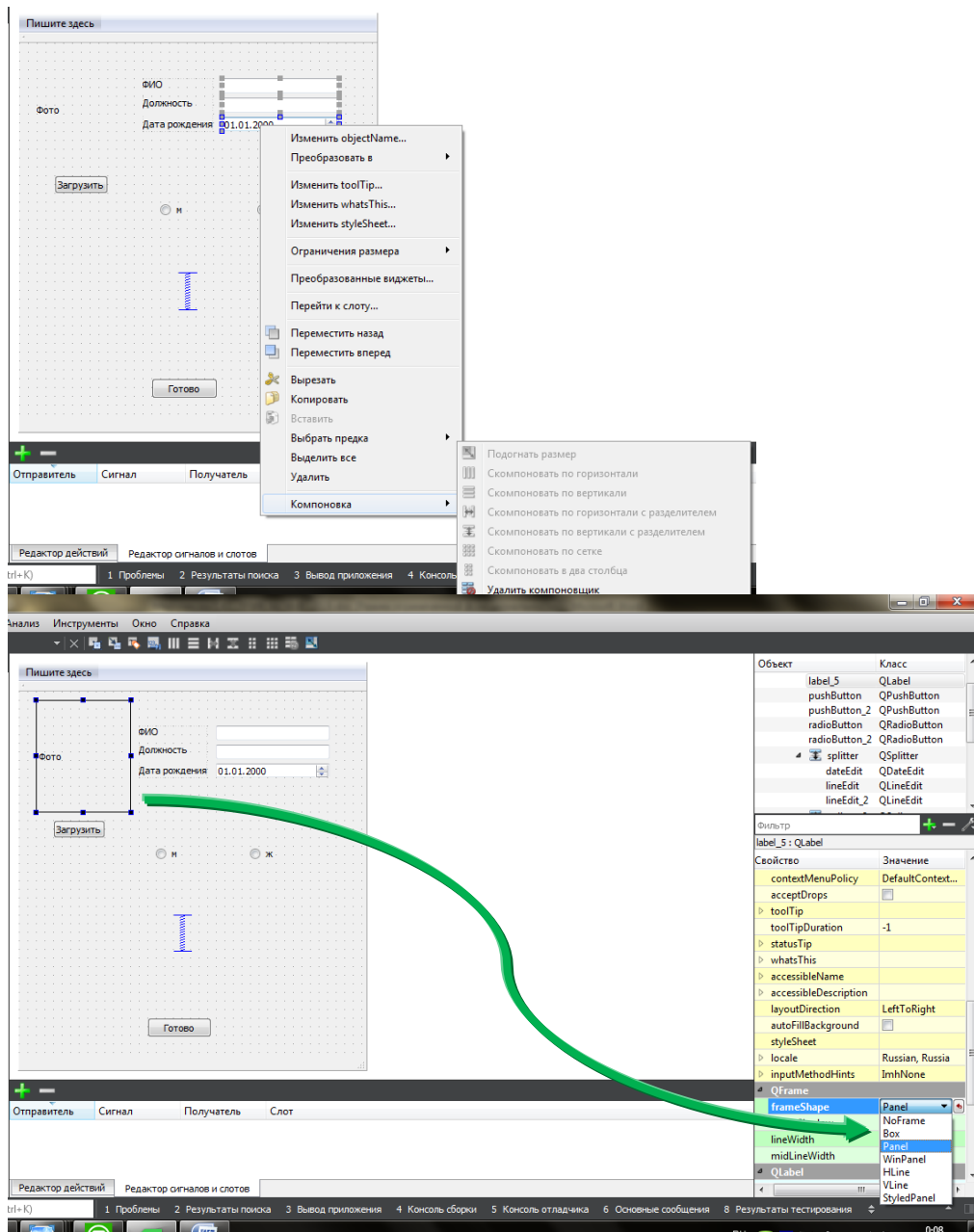
Название проекта – ваши фамилии.

1. Запустить QtCreator и создать приложение Qt Widgets.
2. Разместить на форме компоненты pushButton, label, lineEdit, dateEdit, radioButton и по желанию verticalSpacer или другие средства для компоновки:

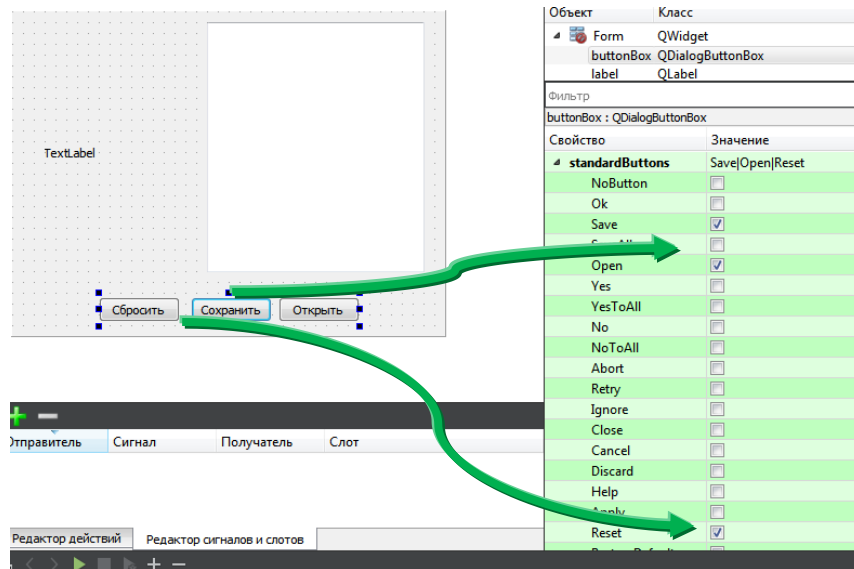


Данная форма будет принимать информацию о сотруднике: ФИО, фотография (на форме это label с надписью «фото» и кнопкой «загрузить»), должность, дата рождения и пол. *Вы должны добавить еще одно поле информации о сотруднике на свое усмотрение, не повторяясь с другими студентами.*

3. Используя менеджеры компоновки и различные средства оформления виджетов (пример показан на рисунке), оформите внешний вид приложения на свой вкус:



4. Придайте приложению определенный стиль. Для этого нужно создать свой файл css и подключить его к программе, либо оформить стиль готовыми средствами QtCreator.
5. Теперь необходимо создать и добавить вторую форму. Переходим в режим редактора и правой кнопкой добавляем Qt / Класс Формы Qt Designer. Выбираем Widget, **меняем стандартное имя Form на свои фамилии** и добавляем в проект. В задании эта форма будет указана со стандартным именем.
На второй форме добавляем label и textEdit для вывода информации и buttonBox (с тремя кнопками) для сохранения информации, загрузки и очистки полей:



6. Добавим новую форму в уже существующее главное окно. Для этого в `mainwindow.h` подключим заголовочный файл формы:

```
#include "form.h"
```

И создадим указатель на будущую форму:

```
private:
Form *myform;
```

7. В конструкторе главного окна добавим создание нового объекта - формы.

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    myform = new Form(); // создаем форму
}
```

8. Для того чтобы по нажатию на кнопку первой формы показать вторую форму, в том же конструкторе подключим сигнал `clicked()` кнопки к слоту `show()` формы:

```
connect(ui->pushButton, SIGNAL(clicked()), myform, SLOT(show()));
```

9. Запустите приложение, по нажатию на кнопку вы увидите вторую форму.

10. Необходимо будет передавать введенные данные из первой формы во вторую. Для этого в `mainwindow.h` добавим сигнал, который будет

отправлять введенные данные, и слот, который будет реагировать на нажатие кнопки и вызывать этот сигнал.

```
signals:
void sendData(QString str);

private slots:
void onButtonSend();
```

11. Реализуйте обработчик нажатия кнопки «Загрузить» на первой форме. По нажатию на эту кнопку нужно запустить диалоговое окно открытия файла и загрузить выбранное изображение в label. Путь открытой картинки выводите в дополнительный lineEdit.

```
void MainWindow::on_pushButton_2_clicked()
{
    QString filename = QFileDialog::getOpenFileName(0, "Выберите изображение", QDir::currentPath(), "*.png *.jpg *.gif *.jpeg");
    ui->lineEdit_3->setText(filename);
    QImage image1(filename);
    ui->label_5->setPixmap(QPixmap::fromImage(image1));
}
```

12. Затем определим слот, в котором отправляются данные из первой формы. Для этого нужно решить, какие данные и как должны формироваться для отправки. Данные о сотруднике – это текстовая информация, которую мы собираем из виджетов на форме. Фотография сотрудника – это в данном случае также текстовая информация в виде пути к открытому файлу. Нажимаем правой кнопкой мыши на кнопке «Готово» главной формы и выбираем «перейти к слоту». В нем нужно сформировать строку данных:

```
QString st = ui->lineEdit_3->text()+"*"+ ui->lineEdit->text()+
"\n"+ ui->lineEdit_2->text() + "\n" + ui->dateEdit->text();
```

Здесь lineEdit_3 содержит путь к файлу. Эту часть строки мы отделяем символом «*» для того, чтобы потом успешно разделить строку, выделив этот путь от остальной информации.

13. Добавляем пол сотрудника в нашу строку, в зависимости от выбранного radioButton:

```
if (ui->radioButton->isChecked()==true) st+="\nпол: мужской";
else st+="\nпол: женский";
```

14. И наконец, вызываем сигнал, в котором передаем введенные данные:

```
emit sendData(st);
```

15. Запись emit вызывает сигнал sendData, который в свою очередь передаёт "куда-то" данные из QLineEdit. Теперь в конструкторе необходимо подключить к слоту onButtonSend сигнал клика по кнопке:

```
connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(onButtonSend()));
```

16. Теперь во второй форме необходимо к чему-то подключить этот сигнал, а также его обработать. В form.h объявляем слот-приемник:

```
public slots:  
void recieveData(QString str);
```

Данный слот будет принимать данные, которые мы передаем сигналом sendData из обработчика нажатия на кнопку первой формы. Данные приходят в виде сформированной, как было показано выше, строки. Эту строку нужно разбить, помня о разделителе «*»: В form.cpp определяем логику работы слота:

```
void Form::recieveData(QString str)  
{  
    QStringList lst = str.split("*"); // Объявляем список строк lst. Разбиваем полученную строку на несколько строк.  
    ui->textEdit->setText(lst.at(1)+"\n"+lst.at(0));  
    if (lst.size()>1) { QImage image1(lst.at(0));  
        ui->label->setPixmap(QPixmap::fromImage(image1));  
    }  
}
```

Таким образом, в textEdit выводится информация о сотруднике и путь к файлу с фотографией. Затем изображение выводится в label по пути, взятому из переданной строки.

17. Подключаем к этому слоту сигнал из главной формы. Для этого в конструкторе в mainwindow.cpp пишем:

```
connect(this, SIGNAL(sendData(QString)), myform, SLOT(recieveData(Q  
String))); //подключение сигнала к слоту нашей формы
```

18. Запустите программу, проверьте ее работу.

19. Осталось написать обработчики нажатия на кнопки второй формы:

Сбросить, Сохранить, Открыть. Нажав правой кнопкой мыши на группу кнопок, вы перейдете в обработчик `on_buttonBox_clicked`. В нем необходимо определить какая кнопка была нажата и запрограммировать ее действие.

20. По нажатию на «Сбросить» (Reset), нужно очищать поля формы. Кнопка в обработчике определяется так:

```
if (button->text() == "Reset")
```

21. Напишите сохранение информации в файл из textEdit, используя примеры из предыдущих лабораторных.
22. Чтобы реализовать загрузку информации из сохраненного файла обратно в форму, необходимо воспользоваться примерами из предыдущих лабораторных и сначала загрузить весь текст в textEdit. Затем, нужно взять строку с информацией и разбить ее на отдельные строки с помощью split(), выделить путь к файлу из четвертой строки и загрузить изображение в label:

```
QStringList inf = ui->textEdit->toPlainText().split("\n");
QImage image2(inf.at(4));
ui->label->setPixmap(QPixmap::fromImage(image2));
}
```

Разделителем в split() здесь служит знак перевода строки.

23. Чтобы не возникало ошибок, реализуйте защиту от пустых полей при вводе информации на главной форме.
24. Запустите программу, проверьте ее работу. Убедитесь, что форма названа по вашим фамилиям, реализовано дополнительное поле (см. пункт 2) и внешний вид приложения демонстрирует применение стилей.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №7

Цель: Научиться работать с графикой в Qt

Требование: Задание по вариантам. Выполняется по подгруппам 1-2 человека. Можно придумать свой вариант, предварительно согласовав с преподавателем.

Задание:

1. Создать графическую сцену.
2. Поместить на сцену различные элементы для составления картинки по варианту. Обязательно использовать и геометрические фигуры, и картинки. Они должны перемещаться с помощью мыши.
3. Ограничить края сцены «стенами» в виде каких-либо элементов.
4. Поместить на сцену движущийся элемент по заданию. Он должен перемещаться с заданной скоростью, сталкиваться со «стенами» и фигурами на сцене. Используйте таймер и функцию обнаружения столкновений.

Варианты по формуле $(n \bmod 10) + 1$, где n – номер студента из подгруппы.

- 1) Башня и движущийся рыцарь
- 2) Дом и движущаяся мышь
- 3) Яблоня и движущаяся птица

- 4) Машина и движущийся мяч
- 5) Грядки и движущаяся ворона
- 6) Пальма и движущаяся обезьяна
- 7) Поезд и движущийся лист
- 8) Обед и движущаяся муха
- 9) Новогодняя ель и движущаяся снежинка
- 10) Человек и движущийся телефон

СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

1. Бланшет Ж., Саммерфилд М. QT 4: программирование GUI на C++. КУДИЦ-Пресс, 2008.
2. Саммерфилд М. Qt Профессиональное программирование. Символ-Плюс, 2011. 552 с.
3. Шлее М. Qt 4.5. Профессиональное программирование на C++. БХВ-Петербург, 2009. 896 с.
4. <http://doc.crossplatform.ru/qt/4.6.x/examples.html> - Примеры программ на Qt, учебное пособие.
5. http://www.opennet.ru/docs/RUS/qt3_prog/qt3.html - Разработка графического интерфейса с помощью библиотеки Qt3.

Екатерина Юрьевна Мерзлякова

ЧЕЛОВЕКО-МАШИННОЕ ВЗАИМОДЕЙСТВИЕ

Методические указания к практическим занятиям

Редактор

Корректор:.....

Подписано в печать.....

Формат бумаги 62 x 84/16, отпечатано на ризографе, шрифт № 10,

изд. л.....,заказ №.....,тираж — экз., СибГУТИ.

630102, г. Новосибирск, ул. Кирова, 86.

