

Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Е. Ю. Мерзлякова

**Визуальное программирование и человеко-машинное
взаимодействие**

Практикум

Новосибирск 2022

УДК 004.5(075.8)

Утверждено редакционно-издательским советом СибГУТИ

Рецензент *д.т.н, проф. М.Г. Курносов*

Мерзлякова Е. Ю. Визуальное программирование и человеко-машинное взаимодействие/ Е. Ю. Мерзлякова; Сибирский государственный университет телекоммуникаций и информатики; каф. прикладной математики и кибернетики. – Новосибирск, 2022. – 22 с.

Практикум предназначен для студентов технических специальностей, изучающих дисциплину «Основы визуального программирования и человеко-машинного взаимодействия» и содержит методические указания к выполнению практических работ, теоретические сведения и задание для выполнения курсовой работы.

© Мерзлякова Е.Ю., 2022

© Сибирский государственный университет
телекоммуникаций и информатики, 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1. ПРАКТИКУМ.....	5
Практическое занятие 1.....	5
Практическое занятие 2.....	11
Практическое занятие 3.....	16
Практическое занятие 4.....	24
Практическое занятие 5.....	24
Практическое занятие 6.....	24
Практическое занятие 7.....	31
2. МЕТОДИКА РАЗРАБОТКИ ИНТЕРФЕЙСОВ.....	31
2.1 Проблемно-центрированная разработка интерфейса.....	31
2.2 СWT-анализ интерфейса.....	36
2.3 Анализ GOMS.....	38
2.4 Золотые правила построения интерфейсов.....	41
2.4.1 Правила Нильсена Молиха (Nielsen, Molich).....	41
2.4.2 Принципы организации графического интерфейса.....	42
3. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ.....	44
4. ЗАДАНИЕ ПО КУРСОВОМУ ПРОЕКТУ.....	45
ЗАКЛЮЧЕНИЕ.....	47
СПИСОК ЛИТЕРАТУРЫ.....	48

ВВЕДЕНИЕ

В настоящее время интерфейсы различных приложений используются повсеместно при выполнении разных, в том числе повседневных задач. Каждый день мы сталкиваемся с необходимостью взаимодействовать с разного рода программами в смартфоне, с помощью которых можно общаться, работать, совершать покупки продуктов и других вещей. Также среднестатистический человек регулярно пользуется банкоматами, в которых также имеется некоторый интерфейс, выполненный с учетом определенных правил. На работе часто люди пользуются программами для выполнения своих задач. В связи с этим особую роль играет вопрос о соблюдении правил построения интерфейсов, проведения различных видов тестирования и обучения этому студентов профильных специальностей.

В настоящем методическом указании к выполнению курсовой работы приведена основная информация для пошаговой разработки интерфейса. Обучающиеся смогут провести цикл разработки приложения по предложенным вариантам.

1. ПРАКТИКУМ

Практическое занятие 1

Целью занятия является знакомство со средой QtCreator, изучение механизма сигналов и слотов. Использование QAction.

Запустите QtCreator. Создайте приложение Qt Widgets как показано на рисунке 1.1.

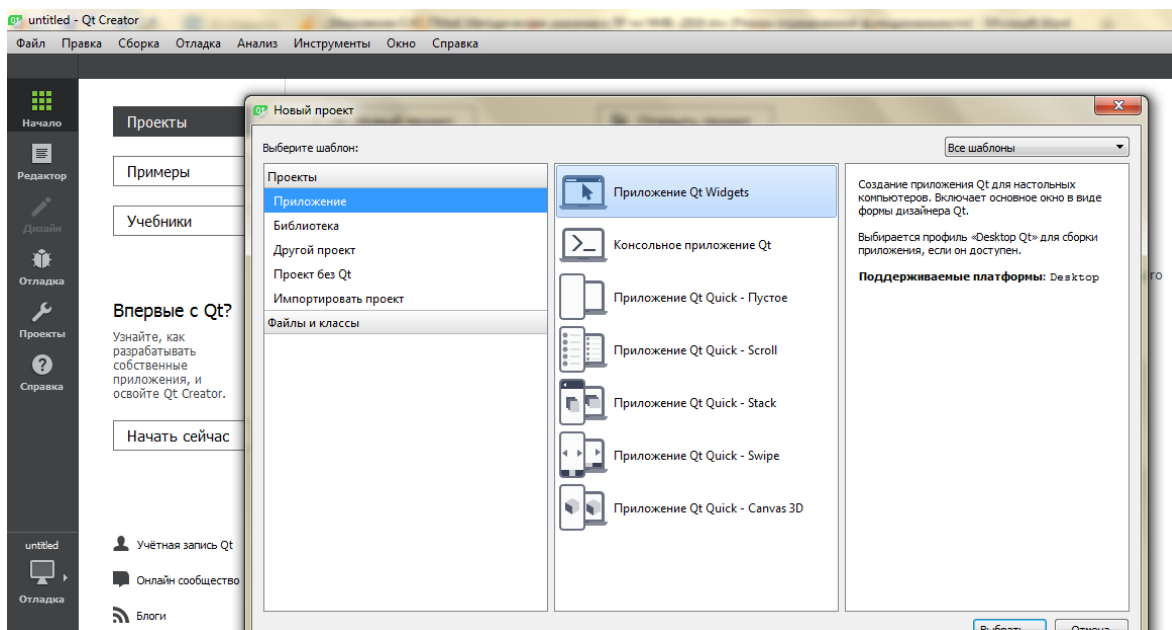


Рисунок 1.1 - Создание приложения

Откройте форму MainWindow и создайте 2 пункта главного меню: «Авторы» и «Выход» как показано на рисунке 1.2.

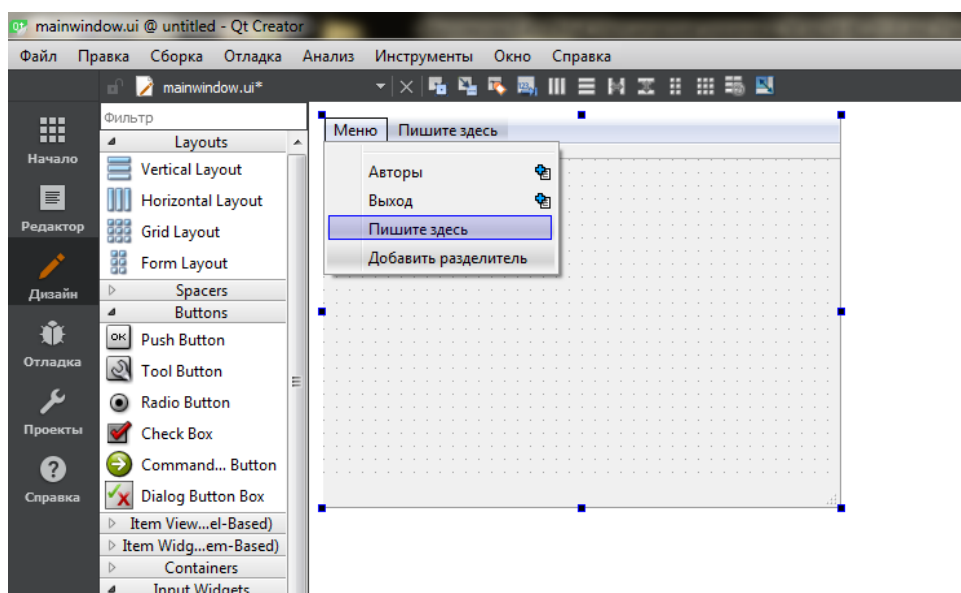


Рисунок 1.2 - Создание меню на главной форме

В свойстве формы «window title» задайте заголовок окна «Лабораторная работа №1» как показано на рисунке 1.3:

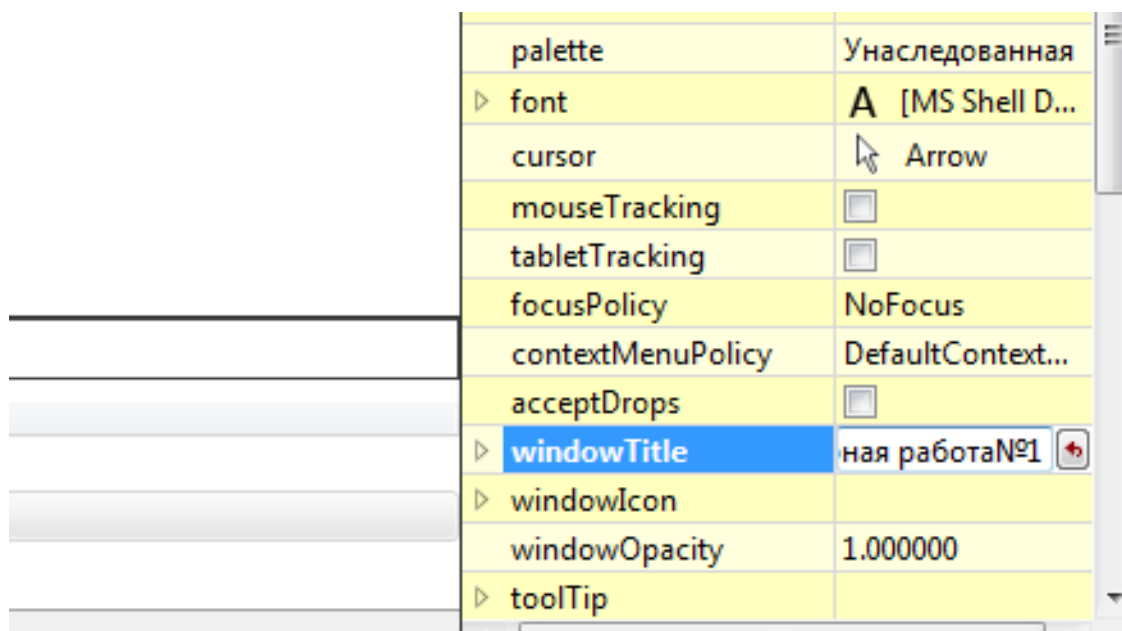


Рисунок 1.3 - Создание заголовка главной формы

В режиме редактирования сигналов и слотов установите соответствующий слот для меню «Выход». Для этого нужно выбрать режим «Изменение сигналов и слотов» как показано на рисунке 1.4:

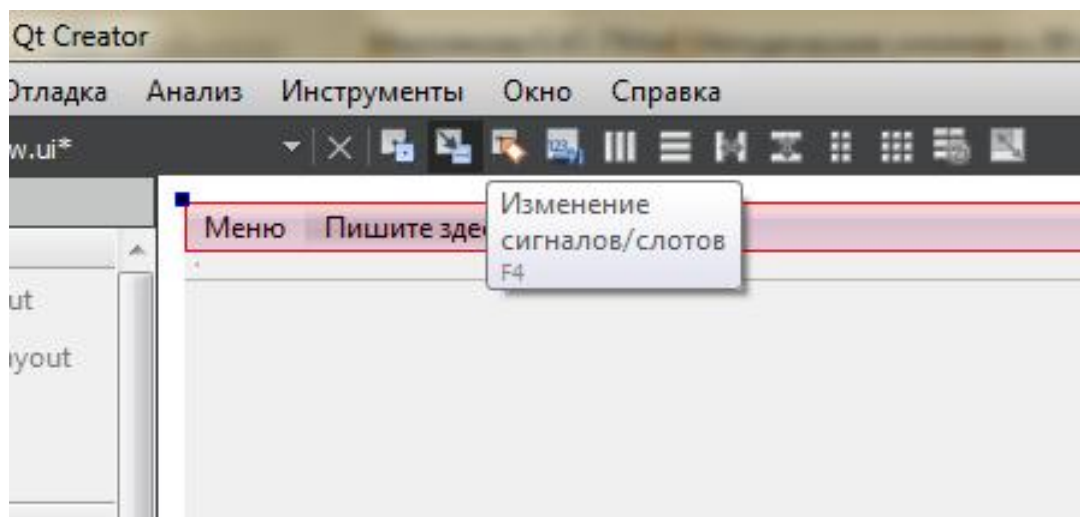


Рисунок 1.4 - Режим изменения сигналов и слотов

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 как показано на рисунке 1.5:

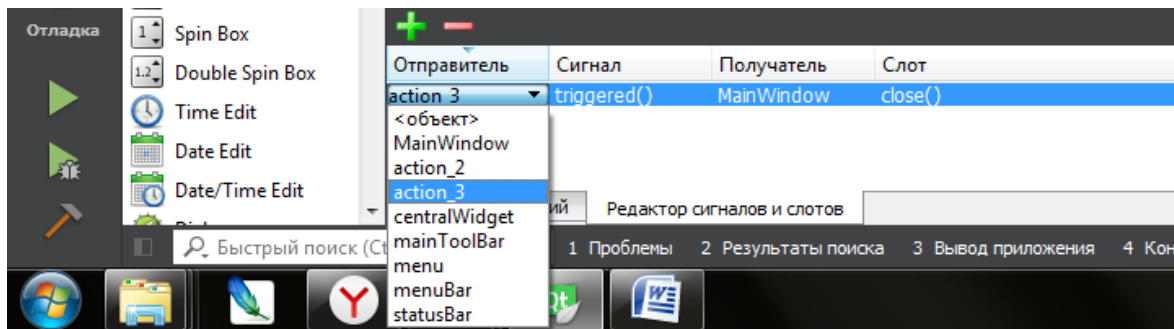


Рисунок 1.5 - Изменение сигналов и слотов

Запустите приложение и проверьте его работу.

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) как показано на рисунке 1.6:

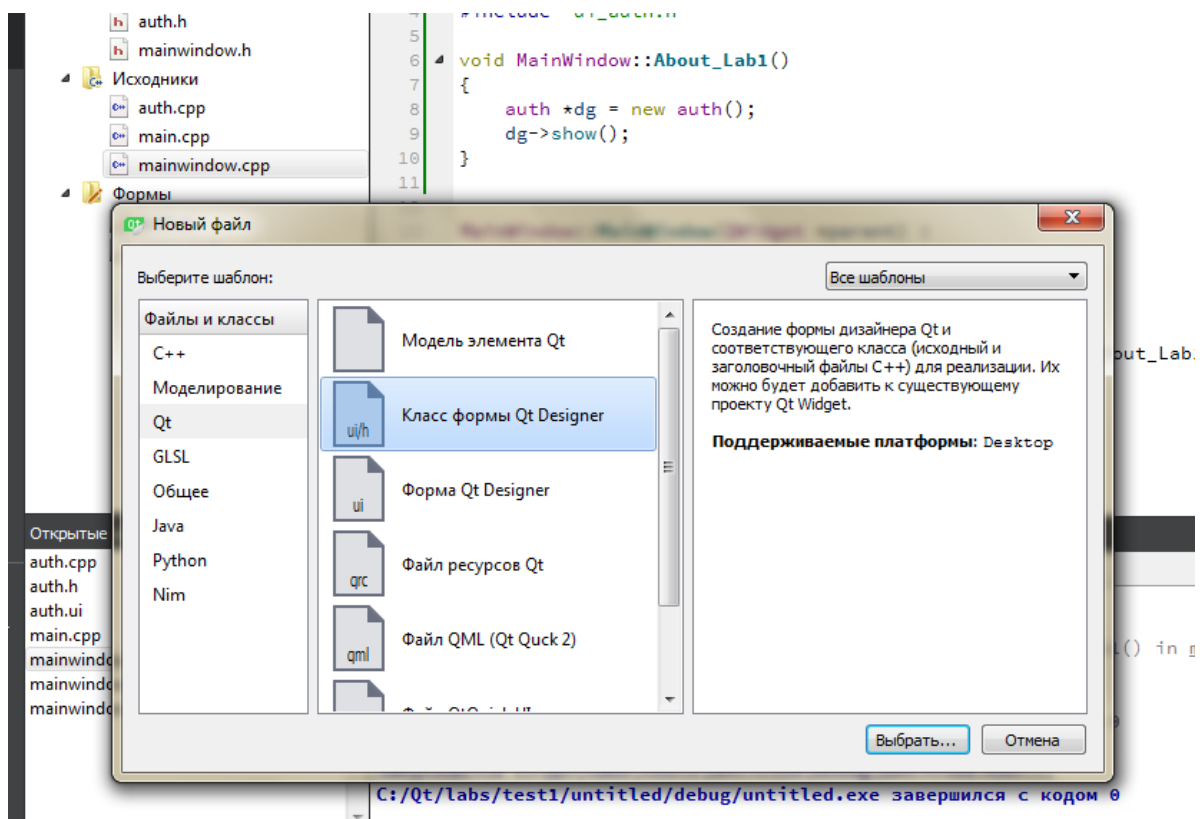


Рисунок 1.6 - Создание новой формы

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». Содержание данного слота показано на рисунке 1.7:

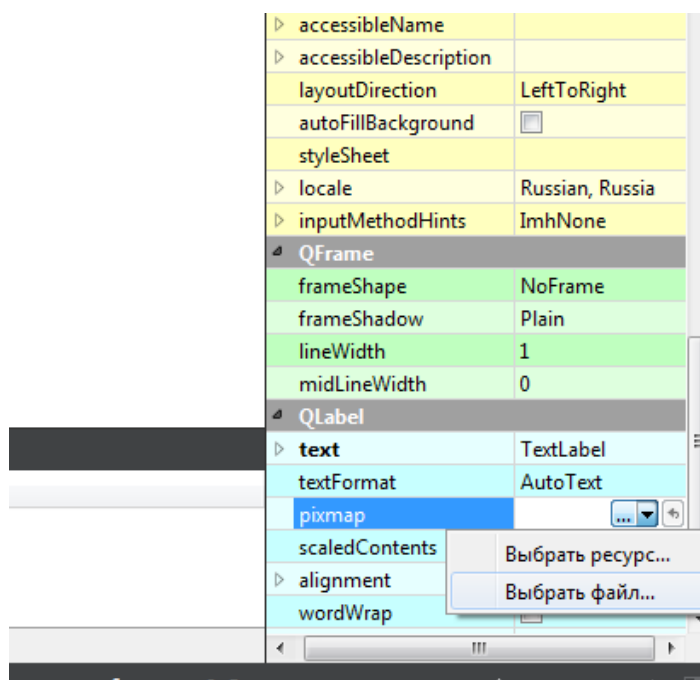


Рисунок 1.9 - Выбор файла в Label

Разместите данные об авторах с помощью виджета TextEdit (двойным щелчком в нем открывается редактор текста) как показано на рисунке 1.10:

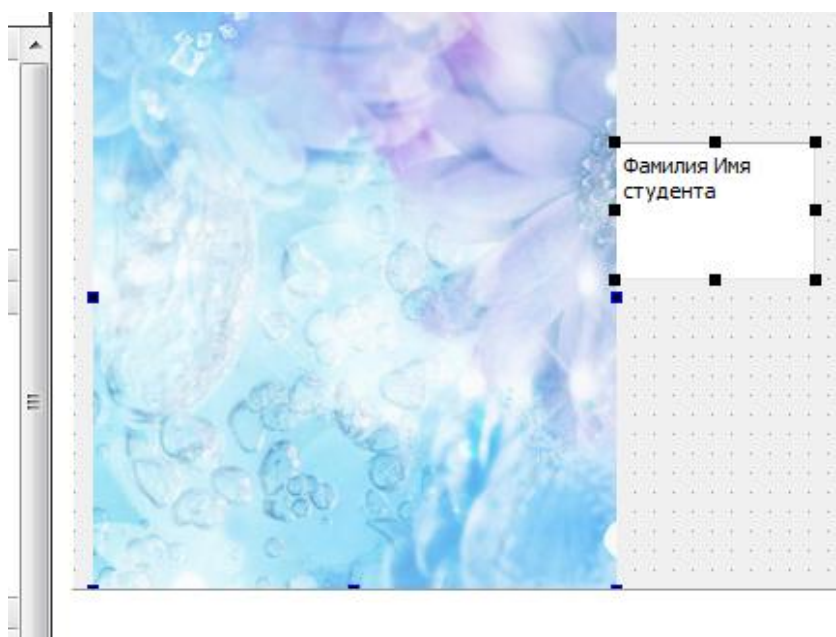


Рисунок 1.10 - Информация об авторах

Запустите приложение и проверьте его работу.

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). Работа с QAction и QMenu показана на рисунке 1.11 :

```
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);
```

Рисунок 1.11 - Работа с QAction и QMenu

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. Слоты About_Lab1() и slotOpen() показаны на рисунке 1.12:

```
#include "QFileDialog"

void MainWindow::About_Lab1()
{
    auth *dg = new auth();
    dg->show();
}

void MainWindow::slotOpen()
{
    QString filename = QFileDialog::getOpenFileName(0, "Открыть файл", QDi
    QFile file(filename);
    if (file.open(QIODevice::ReadOnly | QIODevice::Text))
    ui->textEdit->setPlainText(file.readAll());
}
```

Рисунок 1.12 - Слоты About_Lab1() и slotOpen()

Запустите программу и проверьте ее работу.

Аналогично создайте действие в меню для сохранения файла. Слот slotSave() для сохранения текста из TextEdit показан на рисунке 1.13:

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

Рисунок 1.13 - Слот slotSave()

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.

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

Добавьте созданные действия в главную панель инструментов как показано на рисунке 1.14:

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

Рисунок 1.14 – Добавление действий в главную панель инструментов

Запустите приложение и проверьте его работу.

Практическое занятие 2

Целью занятия является изучение класса QDialog и создание собственного диалогового окна.

Запустите QtCreator и создать приложение Qt Widgets. Приложение будет представлять из себя кнопку, по нажатию на которую отобразится диалоговое окно ввода имени и фамилии.

Создайте виджет класса StartDialog (к названию добавить свои фамилии), предназначенный для запуска диалогового окна. Создание виджета показано на рисунке 1.15

```

#include "StartDialog.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    StartDialog startDialog;
    startDialog.show();

    return a.exec();
}

```

Рисунок 1.15 – Создание виджета класса StartDialog

Класс StartDialog, приведенный в данном листинге, будет унаследован от класса кнопки QPushButton. Сначала создадим класс StartDialog, как показано на рисунке 1.16:

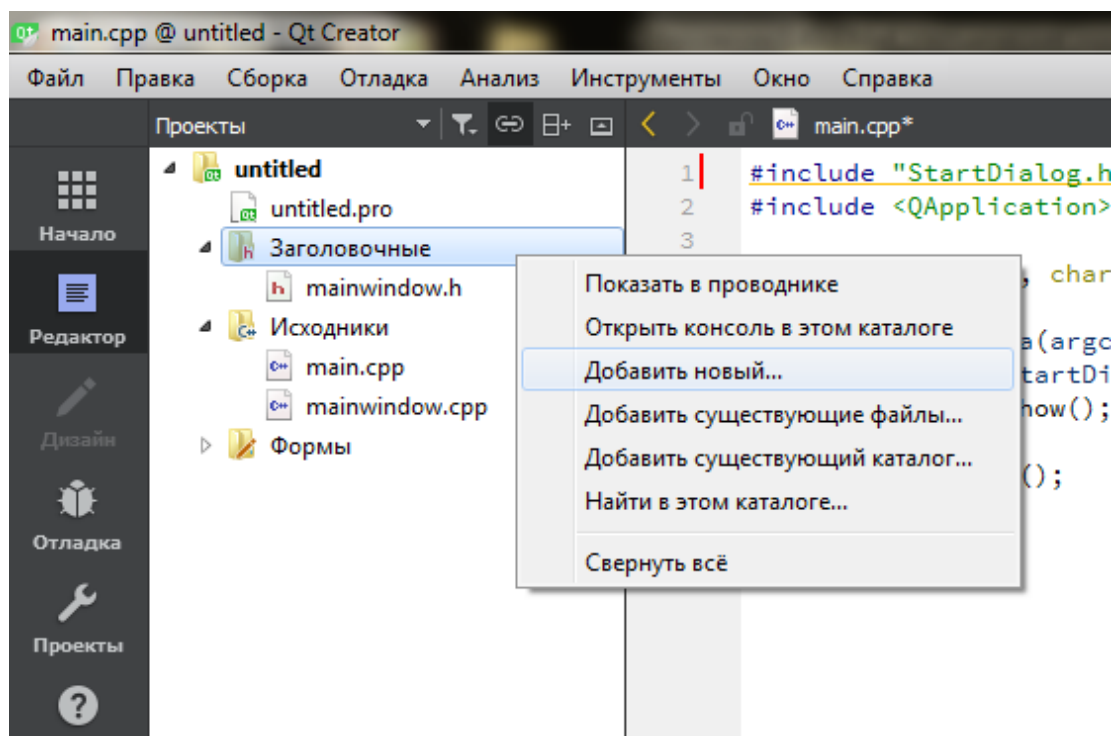


Рисунок 1.16 – Создание класса StartDialog

В названии класса необходимо добавить свои фамилии. На рисунке 1.17 показано, где нужно вписать фамилии:

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

Рисунок 1.17 – Добавление фамилий к названию создаваемого класса

Класс StartDialog унаследован от класса кнопки QPushButton. Сигнал clicked() методом connect() соединяется со слотом slotButtonClicked() как показано на рисунке 1.18:

```

StartDialog.h
slotButtonClicked(): void

#ifndef STARTDIALOG_H
#define STARTDIALOG_H

#include <QWidget>
#include <QPushButton>
#include <QMessageBox>
#include "InputDialog.h"

class StartDialog: public QPushButton {
    Q_OBJECT
public:
    StartDialog(QWidget* pwgt = 0) : QPushButton("Нажми", pwgt)
    {
        connect(this, SIGNAL(clicked()), SLOT(slotButtonClicked()));
    }
public slots:

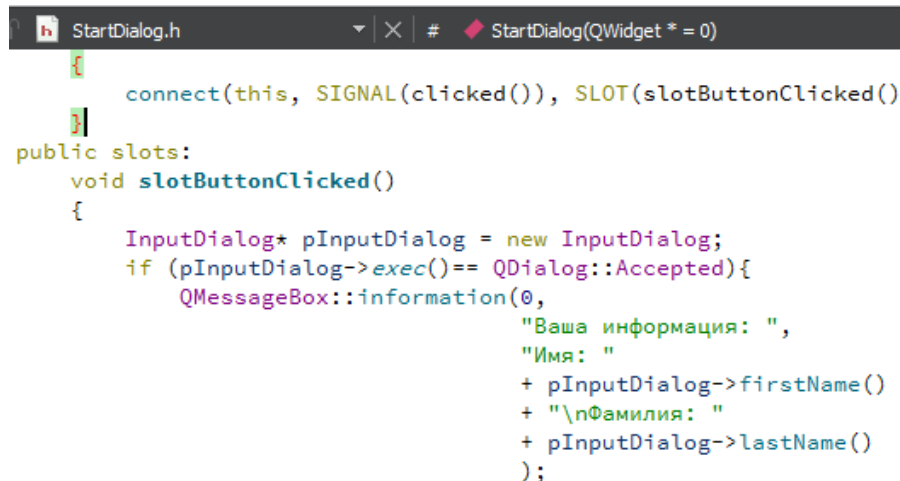
```

Рисунок 1.18 – Соединения сигнала clicked со слотом slotButtonClicked()

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 exec() возвращает значение нажатой пользователем кнопки.

В том случае, если пользователем была нажата кнопка Ок, должно быть отображено информационное окно с введенными в диалоговом окне данными. Как это сделать, показано на рисунке 1.19:



```

connect(this, SIGNAL(clicked()), SLOT(slotButtonClicked()))
}

public slots:
void slotButtonClicked()
{
    InputDialog* pInputDialog = new InputDialog;
    if (pInputDialog->exec() == QDialog::Accepted){
        QMessageBox::information(0,
                                "Ваша информация: ",
                                "Имя: "
                                + pInputDialog->firstName()
                                + "\nФамилия: "
                                + pInputDialog->lastName()
                                );
    }
}

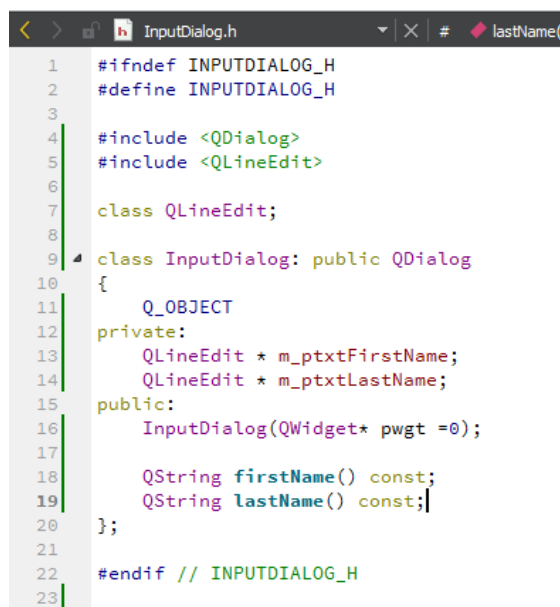
```

Рисунок 1.19 – Реализация информационного окна

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

```
delete pInputDialog;
```

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(), что показано на рисунке 1.20:



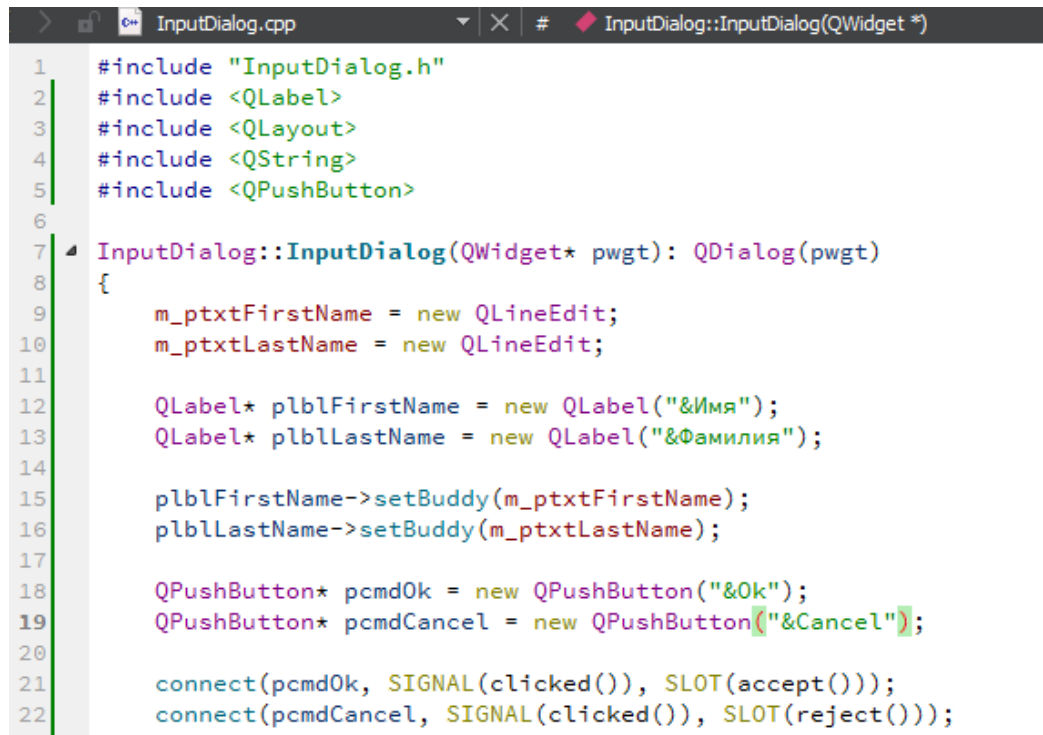
```

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

```

Рисунок 1.20 – Создание своего класса

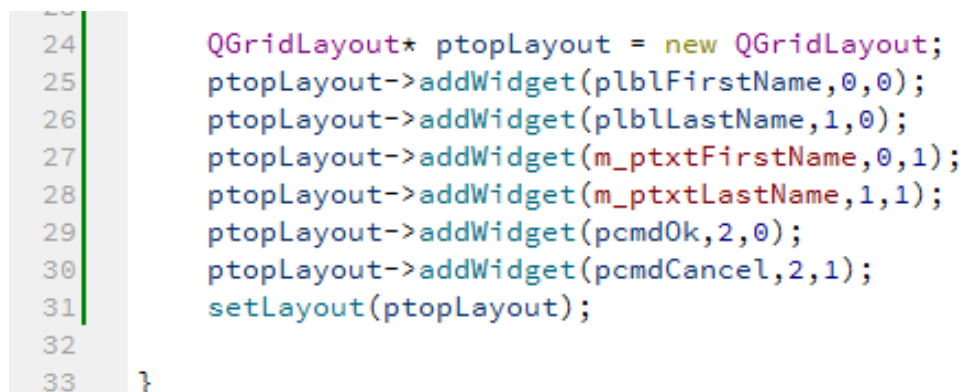
Модальное диалоговое окно всегда должно содержать кнопку Cancel. Сигналы clicked() кнопок Ok и Cancel соединяются со слотами accept() и rejected() соответственно. Это делается для того, чтобы метод exec() возвращал при нажатии кнопки Ok значение QDialog::Accepted, а при нажатии на кнопку Cancel – значение QDialog::Rejected. Создание кнопок показано на рисунке 1.21:



```
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()));
```

Рисунок 1.21 – Создание кнопок

Добавим менеджер компоновки, как показано на рисунке 1.22:



```
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 }
```

Рисунок 1.22 – Добавление менеджера компоновки

Метод firstName() возвращает введенное пользователем имя, как показано на рисунке 1.23:


```

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

```

Рисунок 1.23 – Реализация метода firstName()

Добавьте аналогично метод для фамилии и запустите приложение.

Практическое занятие 3

Целью данного задания является создание SDI- приложения (Single Document Interface, однодокументный интерфейс), простого текстового редактора и окна заставки. В названиях создаваемых классов необходимо добавлять свои фамилии.

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). Окно смены имени класса показано на рисунке 1.24:

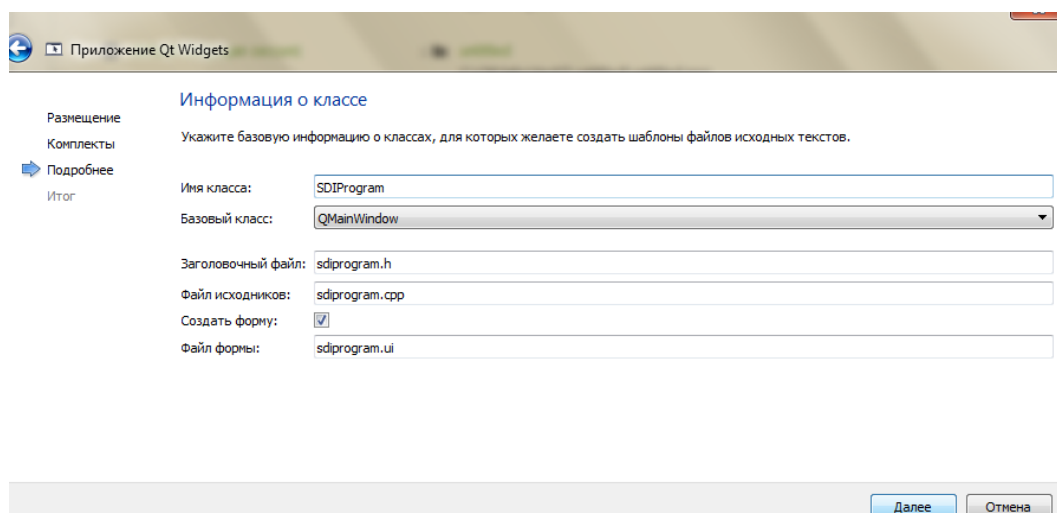


Рисунок 1.24 – Окно смены имени класса

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 показан на рисунке 1.25:

```
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
20 #endif // DOCWINDOW_H
```

Рисунок 1.25 – Содержание файла DocWindow.h

Слоты slotLoad(), slotSave() и slotSaveAs() необходимы для проведения операций чтения и записи файлов, их определение показано на рисунке 1.26:

```
17
18 public slots:
19 void slotLoad();
20 void slotSave();
21 void slotSaveAs();
22
23 };
```

Рисунок 1.26 – Определение слотов

В конструктор класса (DocWindow.cpp) передается указатель на виджет предка, как показано на рисунке 1.27:

```
#include "docwindow.h"

DocWindow::DocWindow(QWidget* pwgt): QTextEdit(pwgt)
{
}

}
```

Рисунок 1.27 – Передача указателя на виджет предка

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()`. Реализация метода `slotLoad` показана на рисунке 1.28:

```
}  
  
void DocWindow::slotLoad()  
{  
    QString str = QFileDialog::getOpenFileName();  
    if (str.isEmpty()){  
        return;  
    }  
}
```

Рисунок 1.28 – Реализация метода `slotLoad()`

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()`, что показано на рисунке 1.30:

```
QFile file(str);  
if(file.open(QIODevice::ReadOnly)){  
    QTextStream stream(&file);  
    setPlainText(stream.readAll());  
    file.close();  
}
```

Рисунок 1.29 – Установка текста в виджете

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

```
m_strFileName=str;  
emit changeWindowTitle(m_strFileName);
```

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()`. Реализация данного слота показана на рисунке 1.30:

```
void DocWindow::slotSaveAs()
{
    QString str = QFileDialog::getSaveFileName(0, m_strFileName);
    if (!str.isEmpty()){
        m_strFileName=str;
        slotSave();
    }
}
```

Рисунок 1.30 – Реализация слота для сохранения файла

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, как показано на рисунке 1.31:

```
}
void DocWindow::slotSave()
{
    if(m_strFileName.isEmpty()){
        slotSaveAs();
        return;
    }
}
```

Рисунок 1.31 – Проверка строки с именем файла

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

```
QFile file(m_strFileName);
```

Вызовите 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()`. Вызов метода `open` показан на рисунке 1.32

```
if(file.open(QIODevice::WriteOnly)){
    QTextStream(&file)<<toPlainText();
```

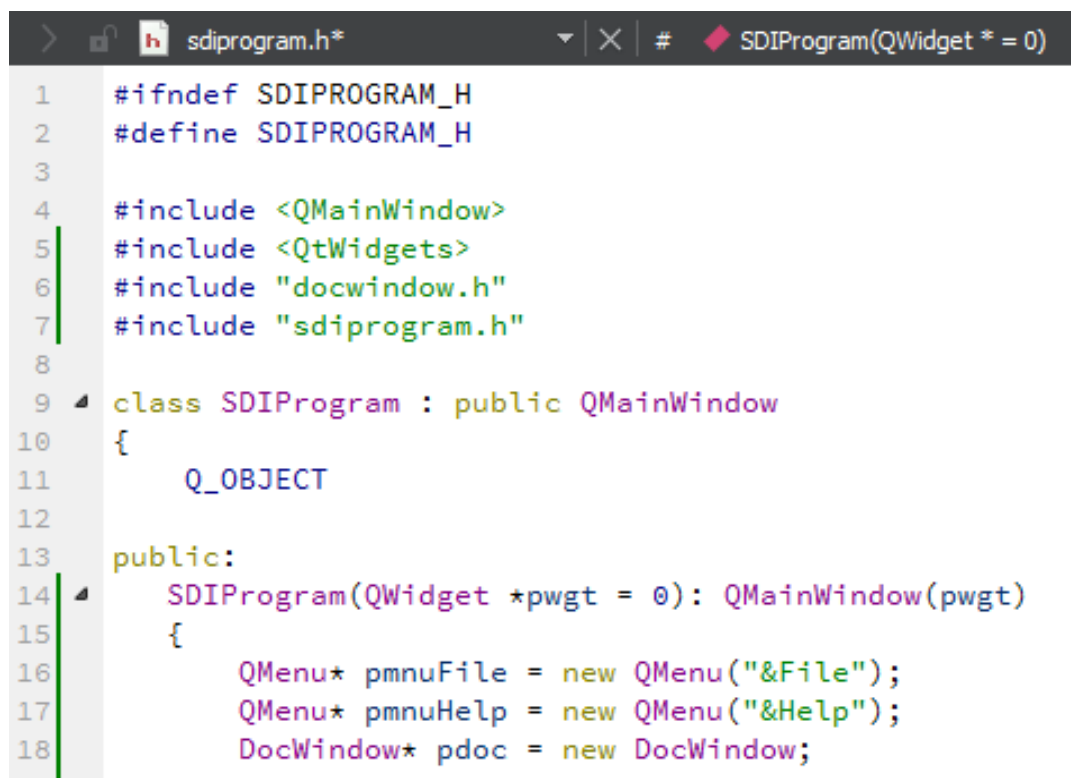
Рисунок 1.32 – Вызов метода open()

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:

```
file.close();
emit changeWindowTitle(m_strFileName);
```

Добавьте окно информационного сообщения после успешного сохранения файла.

Класс SDIProgram 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, что показано на рисунке 1.33:



```
> sdiprogram.h* SDIProgram(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 SDIProgram : public QMainWindow
10 {
11     Q_OBJECT
12
13 public:
14     SDIProgram(QWidget *pwgt = 0): QMainWindow(pwgt)
15     {
16         QMenu* pmnuFile = new QMenu("&File");
17         QMenu* pmnuHelp = new QMenu("&Help");
18         DocWindow* pdoc = new DocWindow;
```

Рисунок 1.33 – Создание виджетов в конструкторе класса SDIProgram

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(), как показано на рисунке 1.34:

```
pmnuFile->addAction("&Open...",  
                    pdoc,  
                    SLOT(slotLoad()),  
                    QKeySequence("CTRL+O")  
                    );  
pmnuFile->addAction("&Save",
```

Рисунок 1.34 – Создание объектов действий

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 qApp).

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.

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

```
menuBar()->addMenu(pmnuFile);  
menuBar()->addMenu(pmnuHelp);
```

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

```
setCentralWidget(pdock);
```

Для изменения текстового заголовка программы после загрузки файла или сохранения его под новым именем сигнал changeWindowTitle(), отправляемый виджетом окна редактирования, соединяется со слотом slotChangeWindowTitle(), как показано на рисунке 1.35:

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

Рисунок 1.35 – Соединение сигнала со слотом

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

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

Осталось добавить слоты slotAbout() и slotChangeWindowTitle(), как показано на рисунке 1.36:

```
public slots:
    void slotAbout()
    {
        QMessageBox::about(this, "Application", "SDI Example");
    }

    void slotChangeWindowTitle(const QString& str)
    {
        setWindowTitle(str);
    }
```

Рисунок 1.36 – Слоты slotAbout() и slotChangeWindowTitle()

Теперь можно запустить программу и проверить ее работу.

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

Добавим окно заставки, как показано на рисунке 1.37. 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:

```

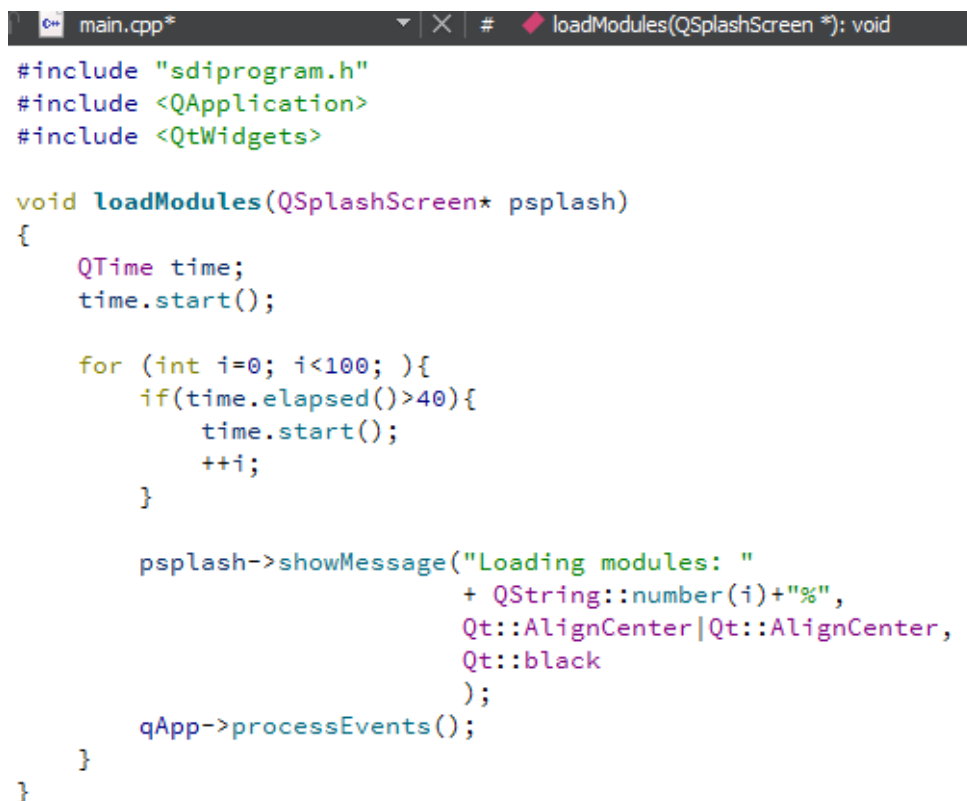
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QSplashScreen splash(QPixmap("s.png"));
    splash.show();
    SDIProgram w;
    loadModules(&splash);
    splash.finish(&w);
    w.show();

    return a.exec();
}

```

Рисунок 1.37 – Добавление окна заставки

Функция `loadModules()`, показанная на рисунке 1.38, является эмуляцией загрузки модулей программы, в нее передается адрес объекта окна заставки, чтобы функция могла отображать информацию о процессе загрузки. Объект класса `QTime` 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:



```

main.cpp*
loadModules(QSplashScreen *): void

#include "sdiprogram.h"
#include <QApplication>
#include <QtWidgets>

void loadModules(QSplashScreen* psplash)
{
    QTime time;
    time.start();

    for (int i=0; i<100; ){
        if(time.elapsed()>40){
            time.start();
            ++i;
        }

        psplash->showMessage("Loading modules: "
                             + QString::number(i)+"%",
                             Qt::AlignCenter|Qt::AlignCenter,
                             Qt::black
                             );
        qApp->processEvents();
    }
}

```

Рисунок 1.38 – Функция `loadModules()`

Запустите приложение, проверьте его работу.

Практическое занятие 4

Цель: Научиться проводить SWT-анализ.

Необходимо выбрать программу, успешно выполняющую минимум две задачи. Программа должна иметь недостатки интерфейса, быть «не идеальной». Проведите SWT анализ двух задач выбранной программы и составьте отчет, который содержит:

- а) описание анализируемой программы
- б) формулировку задачи 1, список действий к задаче 1
- г) анализ действий задачи 1, список выявленных проблем и пути их решения
- д) формулировку задачи 2, список действий к задаче 2
- е) анализ действий задачи 2, список выявленных проблем и пути их решения

Практическое занятие 5

Цель: Научиться проводить GOMS-анализ.

Необходимо выбрать программу, успешно выполняющую минимум две задачи. Программа должна иметь недостатки интерфейса, быть «не идеальной».

Проведите GOMS анализ двух задач выбранной программы и составьте отчет, который содержит:

- а) описание анализируемой программы
- б) формулировку задачи 1
- в) цель, подцели задачи, методы и операции задачи 1
- г) подсчет времени выполнения операций задачи 1
- д) формулировку задачи 2
- е) цель, подцели задачи, методы и операции задачи 2
- ж) подсчет времени выполнения операций задачи 2

Практическое занятие 6

Целью занятия является освоение механизма обмена данными между формами, обучение использовать стили в оформлении интерфейса.

Запустить QtCreator и создать приложение Qt Widgets.

Разместить на форме компоненты pushButton, label, lineEdit, dateEdit, radioButton и по желанию verticalSpacer или другие средства для компоновки как показано на рисунке 1.39

Пишите здесь

ФИО

Фото

Должность

Дата рождения

01.01.2000

Загрузить

М Ж

Готово

Рисунок 1.39 – Размещение компонентов на форме

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

Используя менеджеры компоновки и различные средства оформления виджетов (как это сделать, показано на рисунке 1.40), оформите внешний вид своего приложения.

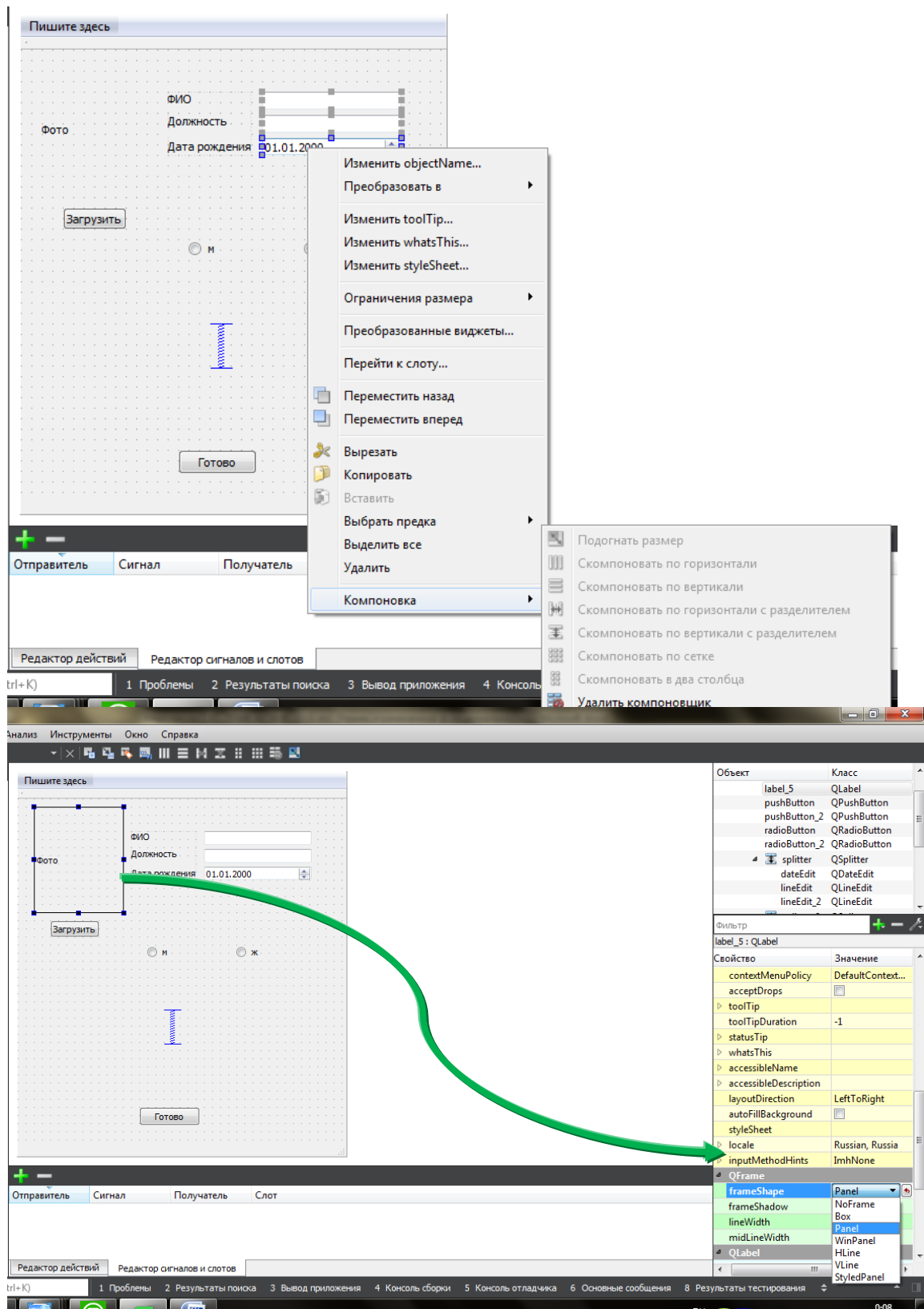


Рисунок 1.40 – Средства оформления виджетов

Придайте приложению определенный стиль. Для этого нужно создать свой файл css и подключить его к программе, либо оформить стиль готовыми средствами QtCreator.

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, как показано на рисунке 1.41:

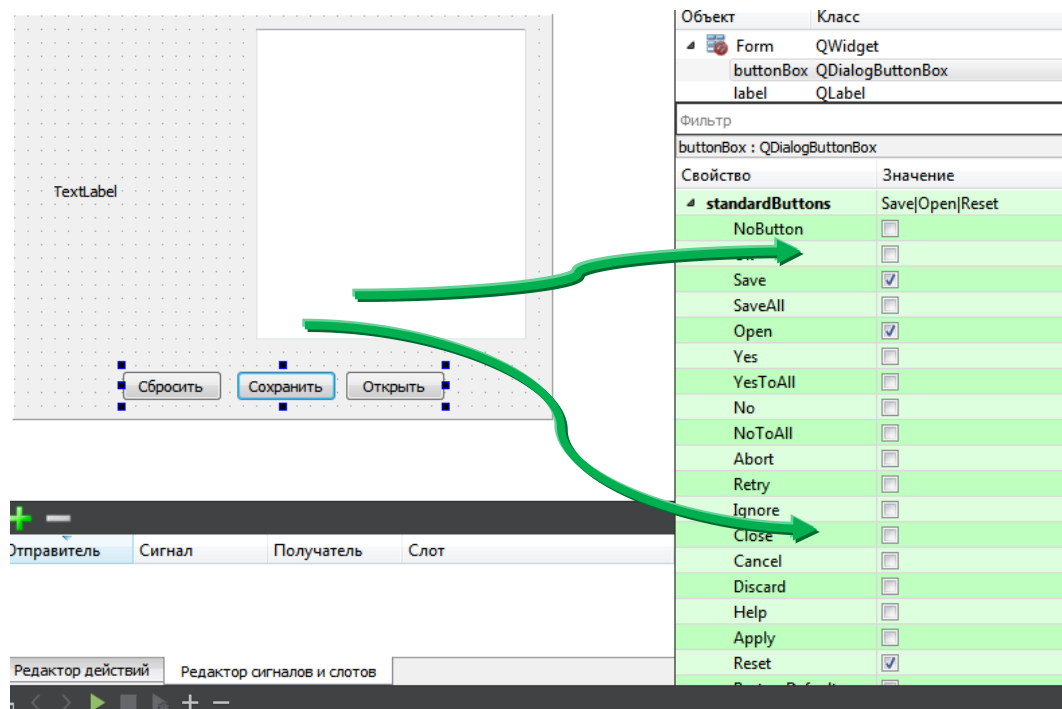


Рисунок 1.41 – Добавление label, textEdit и buttonBox

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;
```

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
}

```

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()));
```

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

Вам 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();

```

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. Данные действия показаны на рисунке 1.42:

```

void MainWindow::on_pushButton_2_clicked()
{
    QString filename = QFileDialog::getOpenFileName(0, "Выберите изображение", QI
        ui->lineEdit_3->setText(filename);
        QImage image1(filename);
        ui->label_5->setPixmap(QPixmap::fromImage(image1));
}

```

Рисунок 1.42 – обработчик нажатия кнопки «Загрузить»

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, как показано на рисунке 1.43:

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

Рисунок 1.43 – Формирование строки данных

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.

Добавляем пол сотрудника в нашу строку, в зависимости от выбранного radioButton, как показано на рисунке 1.44:

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

Рисунок 1.44 – Добавление данных в строку

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

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

Запись emit вызывает сигнал sendData, который в свою очередь передаёт "куда-то" данные из lineEdit. Теперь в конструкторе необходимо подключить к слоту onButtonSend сигнал клика по кнопке, как показано на рисунке 1.45:

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

Рисунок 1.45 – Подключение сигнала к слоту

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, как показано на рисунке 1.46:

```

void Form::recieveData(QString str)
{
    QStringList lst = str.split("*"); // Объявляем список строк lst.
    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));
    }
}

```

Рисунок 1.46 – Реализация слота recieveData()

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.

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

```

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

```

Запустите программу, проверьте ее работу.

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.

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

```

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

```

Напишите сохранение информации в файл из textEdit, используя примеры из предыдущих лабораторных.

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, как показано на рисунке 1.47. Разделителем в split() здесь служит знак перевода строки.

```

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

```

Рисунок 1.47 – Выделение пути к файлу для загрузки в label

Чтобы не возникало ошибок, реализуйте защиту от пустых полей при вводе информации на главной форме.

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 занятия состоит в изучении работы с графикой в Qt.

1. Создать графическую сцену.
2. Поместить на сцену различные элементы для составления картинki по варианту. Обязательно использовать и геометрические фигуры, и картинki. 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

2. МЕТОДИКА РАЗРАБОТКИ ИНТЕРФЕЙСОВ

2.1 Проблемно-центрированная разработка интерфейса.

Одним из наиболее эффективных подходов к разработке интерфейса с пользователем, предлагаемых в литературе по человеко-машинному взаимодействию, является подход, сфокусированный на задачах, которые нужно решать пользователю – проблемно-центрированный подход. При таком подходе процесс разработки структурируется исходя из специфических задач, которые пользователь должен будет решать с помощью разрабатываемой системы. Эти задачи выбираются на ранней стадии разработки, затем они используются для выявления требований к дизайну, чтобы облегчить выработку решений и их оценку по мере развития проекта. Основные этапы разработки проблемно-центрированного дизайна:

- анализ задач и пользователей;
- выбор репрезентативных задач;
- заимствование;
- черновое описание дизайна;
- обдумывание дизайна;
- создание прототипа;
- тестирование дизайна с пользователями;
- итерирование;
- реализация;
- отслеживание эксплуатации;
- изменение дизайна.

На этапе **анализа задач и пользователей** предстоит определить, кто и зачем собирается использовать разрабатываемую систему. Осведомлённость об уровне знаний пользователя позволяет дизайнеру ответить на вопросы о выборе названий пунктов меню, о том, какие материалы включить в обучающий комплект и контекстную помощь, и даже о том, какие возможности должна обеспечивать система. Такие различия среди пользователей, как уровень их квалификации, интерес к изучению новых систем, заинтересованность в успехе разработанной системы, могут обуславливать многие решения дизайнера, например, какой заложить уровень обратной связи с пользователем, где предпочесть команды, подаваемые с помощью клавиатуры, а где – выбираемые из меню и т.д. Эффективный анализ задач и пользователей требует персонального контакта между командой разработчиков и теми людьми, которые в дальнейшем будут пользоваться системой.

После выработки хорошего понимания пользователей и их задач более традиционный процесс разработки может абстрагироваться от этих фактов и привести к общей спецификации системы и её пользовательского интерфейса. Проблемно-центрированный дизайн предполагает другой подход. Разработчик должен выделить несколько **репрезентативных задач**, которые будут решаться при использовании системы. Это должны быть задачи, которые пользователи описали разработчикам. Первоначально они могут быть описаны буквально в нескольких словах, но, т.к. речь идёт о реальных задачах, в любой момент в дальнейшем эти описания могут быть расширены до любой степени детальности, чтобы ответить на любые вопросы, касающиеся дизайна интерфейса или анализа предложенных решений. Отобранные задачи должны достаточно полно покрывать всю функциональность системы, и дизайнеру полезно сделать проверочный список всех функций и путём сопоставления его с перечнем задач удостовериться, что желаемое покрытие достигнуто.

На этапе **заимствования** необходимо найти существующие интерфейсы, с помощью которых пользователи могут выполнить требуемую работу, и затем строить идеи новой системы на их базе насколько это законно и возможно

практически. Например, если они используют электронные таблицы, то, может быть, ваш дизайн должен выглядеть как электронная таблица. Существующую парадигму будет легче изучить и удобнее применять для пользователей, т.к. они уже заранее будут знать, как работает большая часть интерфейса. Копирование известных решений также полезно для низкоуровневых деталей интерфейса, таких как размещение кнопок и названия полей меню. Например, пусть спецификация вашей системы требует, чтобы в какой-то момент могла быть выполнена проверка правописания. Тогда вы должны посмотреть на элементы управления в подсистемах проверки правописания в текстовых процессорах, которые в данный момент используют будущие пользователи вашей системы. Будет лучше, если в вашей системе данный интерфейс будет построен таким же образом.

Черновое описание разрабатываемой вами системы должно быть положено на бумагу (обязательно). Это описание не следует оформлять в виде компьютерной программы (пока), даже если вы умеете пользоваться какими-либо системами автоматизации разработки. Такие системы вынуждают вас прикрепляться к конкретным решениям, которые ещё слишком рано делать. На этой стадии команда разработчиков может проводить множество дискуссий по поводу того, какие возможности должна включать в себя система и как они должны представляться пользователям. Дискуссия должна следовать проблемно-центрированному подходу. То есть, если предлагается введение новой возможности, то необходимо определить, решению какой из репрезентативных задач эта новая возможность будет способствовать. Возможности, которые не способствуют решению любой из задач, как правило, должны отбрасываться. Либо же список репрезентативных задач должен быть дополнен реальной задачей, которая требует этой возможности программы.

Никакая авиационная компания не будет разрабатывать и строить реактивный самолёт без предварительного инженерного анализа, который предсказывает основные технические характеристики. Стоимость строительства и риск неудачи слишком велики. Точно так же стоимость построения законченного пользовательского интерфейса и его тестирование с достаточным количеством пользователей для выявления всех проблем неприемлемо высока. Существует несколько структурных подходов, которые можно использовать, чтобы исследовать сильные и слабые стороны интерфейса до его программного воплощения. Данный этап называется этапом **обдумывания дизайна**. Один из методов состоит в подсчёте количества нажатий клавиш, движений мыши и мыслительных операций (решений), необходимых для выполнения задач, предписанных разрабатываемой системе. Это позволяет оценить трудоёмкость выполнения задач по времени и выявить задачи, требующие слишком много шагов. Такой метод называется GOMS анализом. Другой метод основан на приёме, названном познавательный сквозной контроль (cognitive walkthrough, CWT), и позволяет находить места в дизайне, где пользователь может делать ошибки. Как и GOMS, CWT

анализирует взаимодействия пользователей с интерфейсом при решении отдельных задач.

После этапа обдумывания дизайна по его описанию на бумаге, приходит очередь **создания макета или прототипа интерфейса**, представляющего собой дальнейшую детализацию предстоящей работы, которую можно показать пользователям. Для этого могут использоваться специальные инструменты для создания прототипов, которые позволяют отделить графическую часть интерфейса от логики функционирования системы. На данном этапе не нужно реализовывать систему целиком. Усилия должны быть сосредоточены на частях её интерфейса, нужных для решения репрезентативных задач.

Опыт показывает, что независимо от того, как тщательно был сделан анализ дизайна на предыдущих этапах, существуют проблемы, которые выявляются только при тестировании дизайна с пользователями. **Тестирование** должно проводиться с людьми, чей уровень образования и специальной подготовки примерно соответствует тому, что будет у реальных пользователей системы. Следует попросить пользователей выполнить одну или несколько репрезентативных задач, решение которых поддерживает система. Здесь оказывается эффективным приём "думать вслух". Вы просите пользователя не только делать необходимые действия для решения поставленной задачи, но и говорить, что он делает и о чём размышляет. Эти комментарии необходимо записывать, так как они дают неоценимые данные для улучшения дизайна системы. Информация, собранная при тестировании системы с пользователями, даёт ответ на многие вопросы: сколько времени потребовало выполнение тех или иных действий и задач в целом, какие допускались ошибки, что вызвало затруднение или удивление у пользователя, даже если это и не привело к ошибке. Записанные комментарии пользователя позволяют понять, почему были допущены ошибки. Без комментариев вы только фиксируете сам факт ошибки, но вынуждены потом догадываться (додумывать за пользователя), почему это произошло. При анализе действий пользователя и его комментариев может выясниться, что он мыслит не так, как дизайнер системы. Это позволит внести корректировки, позволяющие приблизить интерфейс системы к её предполагаемому пользователю.

Тестирование с пользователями всегда показывает какие-то проблемы с дизайном. Помните, что цель тестирования состоит не в том, чтобы доказать правильность интерфейса, а в том, чтобы улучшить его. Необходимо проанализировать результаты тестирования, соизмеряя стоимость корректировок с серьёзностью возникших проблем, затем доработать интерфейс и протестировать его снова. Серьёзные проблемы могут даже потребовать пересмотра понимания задач и пользователей, т.е. откатить вас на первый этап проблемно-центрированного подхода. Необходимо на каждой **итерации** помнить, что различные возможности и особенности интерфейса не самостоятельны. Например, переделывание меню для устранения проблемы, возникшей при выполнении одной задачи, может создать проблемы для других

задач. Некоторые из таких взаимовлияний могут быть найдены при анализе без пользователей с помощью SWT или аналогичных приёмов. Другие же не выявятся без тестирования с пользователями. Когда следует прекратить итерации? Если были установлены специфические требования практичности, то итерации прекращаются, как только эти требования выполнены. В противном случае прекращение итераций – это управленческое решение, принимаемое на основе баланса стоимости и полезности дальнейших улучшений против необходимости выхода на рынок с законченным продуктом или сроков предоставления его пользователям.

Ключевой руководящий принцип в программной **реализации** интерфейса состоит в обеспечении возможности его дальнейшего изменения. Постарайтесь предусмотреть некоторую настройку с помощью небольших изменений значений констант и переменных. Например, если вы пишете собственную подпрограмму для реализации специализированного меню, то не закладывайте жёстко в программу такие параметры, как размер, цвет, количество элементов. Постарайтесь также предусмотреть возможность небольших изменений кода, используя ясное разделение на модули. Если доработки в будущем потребуют замены специализированного меню какой-либо более общей функциональной возможностью, доработки кода должны быть тривиальны. Всё это звучит как обычные требования к хорошему стилю программирования и в действительности ими и является. Но это особенно важно для пользовательского интерфейса, который часто занимает более половины кода коммерческого продукта.

Фундаментальный принцип рассматриваемого подхода состоит в том, что команда разработчиков не должна быть изолирована от всей остальной деятельности, связанной с функционированием системы. Если этот принцип уважается, то разработчик должен иметь контакт с пользователями не только в процессе разработки, но и после выпуска системы. **Отслеживание эксплуатации** - это ключевой момент для всех серьёзных организаций, заинтересованных в своём постоянном присутствии на рынке. Один из методов введения разработчиков в контакт с пользователями – организация поочерёдных дежурств на горячей линии связи с потребителями. Другая важная вещь для больших систем – организация собраний групп пользователей и конференций. Такая работа позволяет лучше понять реакцию пользователей на продукт, который они продают. Эта информация в дальнейшем позволяет улучшить описание задач для новой версии продукта и углубить понимание разработчиком предметной области.

На существующем сегодня рынке компьютеров и программ вряд ли существуют продукты, которые поддерживают свою жизнеспособность без регулярных усовершенствований. Независимо от того, насколько удачно система была спроектирована первоначально, с большой вероятностью она будет терять адекватность с течением лет. Меняются задачи, меняются пользователи, приходит время **изменения дизайна**. Приёмы работы меняются

из-за нового оборудования и программных продуктов. Пользователи приобретают новые навыки и ожидаемые реакции. Разработчики должны стоять вровень с этими изменениями, не только отслеживая состояние той рабочей среды, для которой была предназначена их система, но и развитие всего общества, технологий и методов, потребностей. Следующая версия системы должна не только устранять обнаруженные ошибки и недостатки, но и давать новые возможности пользователям.

Практичность (англоязычный термин – usability) – это одна из важнейших характеристик систем, изучению которой посвящено множество исследований. Управленческая деятельность в серьёзной корпорации может потребовать от вас представить различные показатели, которые количественно измеряют практичность. **Требования практичности** – это целевые значения для таких характеристик, как скорость выполнения репрезентативных задач и допустимое количество ошибок. Эти показатели могут использоваться, чтобы мотивировать разработчиков и обосновывать решения по распределению ресурсов. Целевые значения могут быть выбраны так, чтобы побить конкурентов или обеспечить функциональные нужды для хорошо определённых задач. Например, в целях успешной конкуренции от интерфейса может потребоваться не только полнота и удобство для пользователя, но и достижение результатов типа сокращения среднего времени выполнения задач пользователя на 20 % по сравнению с известными аналогами. Типичным примером специальной разработки в области интерфейса, значительно улучшающей скорость работы, является алгоритм T9 для набора текстов на телефонной клавиатуре.

2.2 CWT-анализ интерфейса.

CWT анализ – это формализованный способ представления мыслей и действий людей, когда они пользуются интерфейсом в первый раз. Аббревиатура CWT означает Cognitive Walkthrough (познавательный сквозной контроль). Всё начинается с того, что у вас есть прототип или детальное описание интерфейса, и вы знаете, кто будет пользователем системы. Вы выбираете одну из задач, решение которых интерфейс должен поддерживать, затем формируете полный письменный список действий, необходимых для выполнения задачи при помощи интерфейса. Далее вы пытаетесь рассказать «правдоподобную историю» о каждом действии, которое должен выполнить пользователь. Для этого необходимо давать мотивацию каждого действия пользователя исходя из его предполагаемых знаний и подсказок и реакций интерфейса. Для каждого действия могут обнаруживаться проблемы, мешающие правильному его выполнению. Эти проблемы записываются, но затем вы предполагаете, что они исправлены, и переходите к следующему действию. В результате у вас остаётся список проблем, что является руководством к действию по исправлению (улучшению) интерфейса.

CWT-анализ позволяет обнаружить несколько типов проблем с интерфейсом.

1. Поставить под сомнение ваши первоначальные и не вполне обоснованные предположения о том, как мыслит пользователь.
2. Выявлять элементы управления, которые очевидны для разработчика, но могут быть непонятны пользователю.
3. Выявлять затруднения с надписями и подсказками.
4. Обнаруживать неадекватную обратную связь, что может заставить пользователя сомневаться в результате и повторять всё с начала, хотя всё было сделано правильно.
5. Показывать недостатки в текущем описании интерфейса.

CWT-анализ фокусируется в основном на проблемах, которые пользователи испытывают при первом взаимодействии, не проходя предварительных тренировок. Такая постановка вопроса чрезвычайно важна для некоторых критических систем, таких как банкоматы или терминалы оплаты. Но та же самая ситуация возникает и в сложных программных системах, когда пользователь выполняет какую-либо задачу впервые. Пользователи часто изучают сложные программы постепенно, углубляясь в детали интерфейса по мере возникновения в том необходимости. Поэтому проблемы "первого знакомства" могут возникнуть даже при взаимодействии с давно используемой программой. Если система построена с уважением принципов CWT-анализа, то она позволяет пользователю плавно подниматься с уровня новичка до уровня эксперта.

Многие упускают необходимость сформировать полный и точный список действий для выполнения задачи. То есть они сами точно не знают, как выполнить задачу, и блуждают по интерфейсу, пытаясь отыскать правильную последовательность действий, а потом они оценивают сложность этого блуждания. Иногда, действительно, бывает полезно оценить сложность такого блуждания, но это не метод CWT. В CWT вы должны начинать, имея в руках полный список отдельных элементарных действий, необходимых для выполнения задания. Если в ходе анализа выясняется, что пользователь испытывает затруднения в определении и выполнении одного из действий, то вас интересует не то, как пользователь выйдет из положения, а сам факт того, что проблема возникла и интерфейс нуждается в доработке.

Основные рекомендации по проведению CWT-анализа сводятся к следующему. Вы определили задачу, класс пользователей, интерфейс и корректную последовательность действий. Далее рекомендуется собрать вместе группу разработчиков и других заинтересованных лиц (в учебных целях вы можете действовать в одиночку). И после этого начинается процесс анализа. Вы пытаетесь рассказать историю о том, почему пользователь выбрал бы каждое действие в списке корректных действий. Вы критикуете историю, чтобы сделать её правдоподобной. Рекомендуется держать в уме четыре вопроса:

- Будут ли пользователи пытаться произвести тот или иной эффект, который даёт действие?
- Видят ли пользователи элемент управления (кнопку, меню, переключатель и т.д.) для осуществления действия?
- Если пользователи нашли элемент управления, поймут ли они, что он производит тот эффект, который им нужен?
- После того как действие сделано, будет ли понятен пользователям тот отклик, который они получают, чтобы перейти к следующему действию с уверенностью?

По результатам CWT-анализа необходимо исправить интерфейс. Большинство исправлений будут очевидны. Сложный случай возникает тогда, когда у пользователя вообще нет никаких причин думать, что действие должно быть сделано. Самое верное решение этой проблемы – исключить это действие; пусть система сама выполнит его. Если так не получается, то необходимо перестроить задачу так, чтобы пользователи получали бы логическое побуждение к выполнению "проблемного" действия.

CWT-анализ позволяет выявить много проблем с интерфейсом, но не даёт ответа на вопрос, насколько сложен интерфейс, т.е. сколько времени тратит пользователь на выполнение задачи.

2.3 Анализ GOMS.

GOMS анализ оценивает время работы с интерфейсом обученного пользователя. Даже если интерфейс успешно прошел CWT-анализ, это не означает, что он оптимален с точки зрения трудоёмкости. Если есть несколько альтернативных вариантов построения интерфейса, то анализ GOMS позволяет выбрать тот из них, который требует меньше времени для решения задачи пользователя. В отличие от CWT, GOMS предполагает, что пользователь уже знает интерфейс, т.е. видит его не первый раз.

Аббревиатура GOMS означает Goals, Operations, Methods, Selections (цели, операции, методы, правила выбора). Модель GOMS состоит из описания методов, необходимых для достижения заданных целей. Методы представляются последовательностями шагов, состоящих из операций, которые выполняет пользователь. Метод может быть назван подцелью, т.е. методы образуют иерархическую структуру. Если существует более одного метода для достижения цели, то включаются правила выбора, с помощью которых в зависимости от контекста выбирается один метод.

В терминологии данного вида анализа цель – это то, что мы раньше называли (репрезентативной) задачей, метод – это список действий пользователя, операции – элементарные действия пользователя.

Операции в GOMS – это элементарные действия, которые нельзя разложить на более мелкие. Причём учитываются как внешние действия, т.е. те, что приводят к видимым физическим эффектам, так и внутренние, связанные с

мышлением пользователя. Например, действие (шаг метода) "нажать кнопку <button>" представляется в виде последовательности следующих операций:

- визуально определить местонахождение кнопки (мыслительная операция);
- навести на кнопку указатель мыши (внешняя операция);
- щелкнуть кнопкой мыши (внешняя операция).

Рассмотрим анализ интерфейсов для типичной конфигурации персонального компьютера, где в качестве устройств ввода-вывода выступают монитор, клавиатура и мышь. Практически все интерфейсные взаимодействия в этом случае можно описать следующими операциями:

K – нажатие клавиши;

B – клик кнопкой мыши;

P – наведение указателя мыши;

R – ожидание ответной реакции компьютера;

H – перенос руки с клавиатуры на мышь или наоборот;

D – проведение с помощью мыши прямой линии (например, выделение или прокрутка текста);

M – мыслительная подготовка (к осуществлению одной из перечисленных операций).

Разные пользователи выполняют указанные операции за разное время. Однако, напомним, что GOMS исследует работу опытного пользователя. Многочисленные исследования выявили средние значения времени операций, выполняемых опытными пользователями. Приведём их значения:

K 0.2 с

B 0.2 с

P 1.1 с

H 0.4 с

M 1.35 с

Время выполнения задачи, посчитанное с использованием этих значений, является хорошей средней оценкой сложности интерфейса и действительно работает на практике.

Время ожидания *R* зависит от характера действия, выполняемого компьютером. Речь идёт только о тех задержках, которые связаны с работой интерфейса (например, ожидание открытия нового интерфейсного объекта). При анализе GOMS, как правило, не учитывается время, расходуемое компьютером на выполнение целевой вычислительной функции (например, преобразование одного формата в другой). Такие вычислительные операции относятся уже к функциональному программному обеспечению; скорость их выполнения определяется быстродействием аппаратуры, объёмом обрабатываемых данных и эффективностью алгоритмов, но не связана со сложностью интерфейса. С другой стороны, очень быстрые реакции

компьютера, кажущиеся для человека практически мгновенными (например, эхо-печать символа после нажатия клавиши), также не учитываются. Следует учитывать только ожидания, которые сбивают ритм выполнения других операций. Время таких ожиданий можно оценить при работе с реальной программой следующим образом: Если реакция компьютера достаточно быстрая, но, всё же, ощутима, то "стандартное" время R рекомендуется установить 0.25 с.

Время операции D также может быть разным в зависимости от величины перемещения и других сопутствующих деталей. Его рекомендуется приблизительно оценить при работе с реальным приложением. В качестве "стандартного" значения можно взять 2 с.

Общий алгоритм действий при проведении GOMS анализа:

- Цель разбивается на подцели,
- для каждой подцели расписываются методы,
- методы могут раскрываться далее через внутренние методы
- Формируется конечная последовательность операций
- добавляются мыслительные подготовки M .
- Считается время в секундах

Вся последовательность операций разбивается на семантические группы. Например, набор на клавиатуре слова "слон" выполняется четырьмя операциями нажатия на клавиши $KKKK$ и рассматривается как отдельная семантическая единица. Перед ней нужно поставить операцию M . Действительно, перед тем как напечатать слово, человек должен сформировать его внутренний образ. Это и отражается добавлением операции M . Но после того как образ сформирован, слово печатается без дальнейших раздумий между отдельными буквами. Пример другой семантической единицы – открытие меню. Оно состоит из двух операций PB . Но прежде чем они могут быть выполнены, нужно решить, какой пункт меню вам нужен и найти, где он находится. Поэтому перед PB (но не между P и B) нужно поставить M . Если затем с помощью ещё одной пары операций PB в меню выбирается элемент, то M ставить не нужно, т.к. идея о том, что нужно выбрать из меню, уже сформировалась у человека ранее.

Наконец, когда вся последовательность операций выписана, мы получаем общее время выполнения (оценка среднего времени) задачи простым суммированием времён отдельных операций. Но это не единственный результат. Проведённый анализ часто позволяет понять, как можно улучшить интерфейс для сокращения времени решения задачи.

Рассмотренных операций может оказаться недостаточно для некоторых интерфейсов. Например, если вы используете сенсорные панели или графические планшеты, то понадобятся специальные, связанные с ними операции. Их временные характеристики можно определить экспериментально или найти в специальной литературе.

2.4 Золотые правила построения интерфейсов

За годы изучения проблем человеко-машинного взаимодействия, специалисты выявили несколько наиболее существенных правил построения интерфейсов, и назвали их "золотыми правилами". Эти правила могут также использоваться для экспертной оценки существующих интерфейсов.

2.4.1 Правила Нильсена Молиха (Nielsen, Molich).

Простой и естественный диалог. Простота означает, что не должно присутствовать не относящейся к теме или редко используемой информации. Старайтесь добиваться максимального следования задачам пользователя, минимизируя сложность поиска соответствия между семантиками задачи и интерфейсом. Важно представить точно ту информацию, в которой нуждается пользователь, следуя принципу "лучше меньше да лучше". Вся редко используемая информация должна быть спрятана. Естественность означает, что информация, которая выводится на экран, должна появляться в естественном порядке, соответствующем ожиданиям пользователя. Вся взаимосвязанная информация должна группироваться в одном месте.

Говорите на языке пользователя. Используйте слова и понятия из мира пользователя. Не используйте специфических инженерных терминов. Например, для большинства людей представление цвета в виде RGB-компонент и их шестнадцатеричная кодировка являются совершенно непонятными вещами. Лучшим и довольно простым решением будет показать вместо кода (или наряду с ним) образец цвета.

Минимизируйте загрузку памяти пользователя. Не заставляйте пользователя помнить вещи от одного действия к следующему. Оставляйте информацию на экране до тех пор, пока она не перестанет быть нужной. Например, хорошим стилем считается делать только один ряд закладок.

Будьте последовательны. У пользователей должна быть возможность изучить действия в одной части системы и применить их снова, чтобы получить похожие результаты в других местах.

Обеспечьте обратную связь. Дайте пользователю возможность видеть, какой эффект оказывают его действия на систему.

Обеспечьте хорошо обозначенные выходы. Если пользователь попадает в часть системы, которая его не интересует, у него всегда должна быть возможность быстро выйти оттуда, ничего не повредив.

Обеспечьте быстрые клавиши и ярлыки. Элементы быстрого доступа могут помочь опытным пользователям избегать длинных диалогов и информационных сообщений, которые им не нужны.

Хорошие сообщения об ошибках. Хорошее сообщение об ошибке помогает пользователю понять, в чём проблема и как это исправить.

Предотвращайте ошибки. Всегда, когда вы пишете сообщение об ошибке, вы должны спросить себя, можно ли избежать этой ошибки? Хороший пример

построения интерфейса, предотвращающего ошибки – стандартная программа «Калькулятор», в которой при переключении в двоичный режим интерфейс делает недоступными числовые кнопки, кроме 0 и 1, а также делает недоступными некоторые функции, имеющие смысл только для чисел с плавающей точкой. Этот простой приём позволяет минимизировать возможные ошибки пользователя.

Снабдите программу системой помощи. Справка должна быть максимально подробной и рассчитана на абсолютно неопытного пользователя.

2.4.2. Принципы организации графического интерфейса

Принцип кластеризации. Организуйте экран в виде визуально разделённых блоков с похожими элементами управления, предпочтительно с названием для каждого блока. Элементы управления включают меню, диалоговые боксы, экранные кнопки и любые другие графические элементы, позволяющие пользователю взаимодействовать с компьютером. Подобные команды должны быть в одном меню: это позволяет им быть визуально близко и идти под одним заголовком. Команды, относящиеся к некоторой конкретной области функциональности, могут также быть показаны в диалоговых боксах, в визуально определяемых блоках. Тот же самый принцип распространяется на специальные управляющие экраны с многочисленными кнопками и значками, такие как сенсорные панели. Кнопки для конкретной функции должны группироваться вместе, а затем выделяться цветом, обрамлением или окружающим пробелом.

Существует две важные причины для кластеризации. Во-первых, она помогает пользователю находить нужную команду. Если вы ищете возможность изменить формат абзаца, то вам легче найти соответствующий диалоговый бокс в меню "Формат", а не в случайно распределённом по экрану наборе из сотни кнопок. Во-вторых, группирование команд позволяет пользователю приобрести концептуальную организацию знаний о программе. Полезно, например, знать, что начертание и размер являются атрибутами шрифта, в то время как интервал и отступ – атрибутами абзаца.

Принцип "видимость отражает полезность". Делайте часто используемые элементы управления заметными, видимыми и легко доступными; и наоборот, прячьте или сжимайте редко используемые элементы. Пример реализации этого принципа в современных системах – панели инструментов, т.е. наборы иконок для часто используемых функций. Обоснование этого принципа заключается в том, что человек может быстро находить что-то среди небольшого числа объектов. Для редко используемых функций можно позволить поиск в длинных списках.

Принцип интеллектуальной последовательности. Используйте похожие экраны для похожих функций. Это похоже на заимствование, но в этом случае вы заимствуете что-то из одной части дизайна и применяете это к другой части. Обоснование этого принципа очевидно: раз пользователи выучили

расположение элементов управления на экране (например, где находится кнопка "Помощь"), они могут легко приложить это знание к другим экранам внутри той же системы. Этот подход позволяет вам сосредоточить усилия на разработке лишь нескольких привлекательных работоспособных экранов, а затем их слегка модифицировать для использования в других частях приложения. Но экраны не должны выглядеть одинаково, если в действительности они должны отражать совершенно разные вещи. Предупреждение о критической ошибке в системе реального времени должно иметь вид, значительно отличающийся от экрана помощи или информационного сообщения.

Принцип "цвет как приложение". Не полагайтесь на цвет как носитель информации; используйте его умеренно, чтобы лишь акцентировать информацию, передаваемую другими средствами. Хорошее правило для большинства интерфейсов – разрабатывать их в чёрно-белом варианте, убедиться, что они работоспособны, а затем добавить минимальный цвет для их окончательного облика. Цвет, безусловно, полезен, когда необходимо выделить предупреждение или информационное сообщение, но обязательно дайте дополнительные ключи для пользователей, не способных воспринимать изменения цвета.

Принцип уменьшения беспорядка. Не помещайте на экран слишком много всего. Этот неформально определяемый принцип – хорошая проверочная точка, чтобы подтвердить, что ваш дизайн отражает перечисленные выше принципы. Если видимы только наиболее часто используемые элементы, все они сгруппированы в небольшое число визуальных кластеров, использован минимум цвета, то экран должен получиться графически привлекательным. Не пытайтесь наделить каждое меню собственным шрифтом или работать с большим набором размеров. Как правило, пользователи заметят не столько различия, сколько беспорядок.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

- 1) Назовите основные этапы построения интерфейса.
- 2) Как происходит тестирование дизайна приложений?
- 3) В чем заключается принцип CWT-анализа?
- 4) Как и для чего проводят GOMS-анализ?
- 5) Что включают в себя золотые правила построения интерфейсов?

ЗАДАНИЕ ПО КУРСОВОМУ ПРОЕКТУ

Необходимо выбрать тему для разработки интерфейса и сообщить ее преподавателю. *Менять вариант после этого не разрешается.*

Предлагаются следующие темы:

- 1) Ежедневник работающего студента
- 2) Журнал учителя начальной школы
- 3) Журнал куратора студенческих групп
- 4) Дневник школьника
- 5) Обучающая игра по английскому для начинающих
- 6) Создание заметок с уведомлениями
- 7) Рисование животных по инструкции
- 8) Составление меню по заданным калориям
- 9) Составление таблиц по заданным параметрам
- 10) Конструктор аватарок
- 11) Индивидуальный фитнес план
- 12) Уроки кулинарии
- 13) Игра «Пазлы»
- 14) Раскраска по номерам
- 15) Составление меню на неделю
- 16) Игра «Стрелялка»
- 17) Журнал преподавателя кафедры
- 18) Игра «Мемо карточки».
- 19) Тренажер по таблице умножения
- 20) Симулятор «Младенец»
- 21) Составление кроссвордов
- 22) Игра на поиск предметов в картинке
- 23) Создание фотоальбомов
- 24) Дневник здоровья
- 25) Планирование уборки дома

После того, как тема выбрана, нужно выполнить следующие этапы разработки приложения, действуя строго по плану:

Часть1

- 1) анализ задач и пользователей;

Найти двух человек, которые могут быть заинтересованы в решении предложенной задачи. Дайте их краткое описание (возраст, образование, профессия, навыки в выбранной сфере, навыки владения компьютером).

- 2) выбор репрезентативных задач;

Перечислите репрезентативные задачи; затем все задачи, решение которых будет поддерживать разрабатываемая программа.

3) заимствование;

Найдите приложения или сайты, с которых можно заимствовать какие-либо решения интерфейса, приведите ссылку на источник. Эти приложения не обязательно должны выполнять точно такие же задачи. Заимствовать можно что угодно, даже расположение кнопок. Выберите и напишите, что именно Вы будете заимствовать из данных приложений и зачем.

4) черновое описание дизайна;

Опишите черновой вариант дизайна словами и графически (иллюстрации с пояснениями). Черновой вариант должен отражать все внешние элементы интерфейса и их назначение.

После завершения данных этапов, необходимо сдать работу на проверку преподавателю. Получив зачет по первой части, можно приступить ко второй и третьей.

Часть 2

Запрограммировать приложение в среде Qt Creator. Обязательно использование минимальной базы данных и выполнение нескольких функций по выбранной теме. Для оценки «хорошо» и «отлично» необходимо реализовать работу с дополнительными файлами, использовать стили и многостраничную справку.

Часть 3

Проведите CWT и GOMS анализы интерфейса на примере двух репрезентативных задач, а также подробный анализ интерфейса на соответствие Правилам Нильсена-Молиха и правилам организации графического интерфейса, приводя примеры для обоснования результата анализа. По итогу проведенных анализов необходимо составить список проблем, предложить способы их решения и исправить интерфейс.

После выполнения трех частей курсовой работы, каждому студенту нужно сдать преподавателю бумажный отчет, который включает следующее содержание:

1. Краткое задание
2. Выполнение первой части задания
3. Описание функций разработанного приложения и скриншоты интерфейса в соответствии со второй частью задания
4. Выполнение третьей части задания
5. Вывод о предложенных доработках интерфейса.

ЗАКЛЮЧЕНИЕ

Методические указания по выполнению практических заданий помогут поэтапно научиться работать в среде QtCreator и создавать приложения для решения различных задач.

В ходе выполнения курсовой работы обучающийся освоит основные этапы цикла разработки приложений и научится выполнять их на практике. Получение знаний и навыков в области анализа интерфейсов позволит в дальнейшем разрабатывать приложения на более высоком уровне.

СПИСОК ЛИТЕРАТУРЫ

- 1) Акчурина Э.А. Человеко-машинное взаимодействие. Учебное пособие. Солон, 2008. 96 с.
- 2) Логунова О.С., Ячиков И.М., Ильина Е.А. Человеко-машинное взаимодействие: теория и практика. Учебное пособие. Феникс, 2006. 285 с.
- 3) Шлее М. Qt 5.10. Профессиональное программирование на C++. БХВ-Петербург, 2018. 1052 с.
- 4) Human Computer Interaction: Concepts, Methodologies, Tools, and Applications. / P. Zaphiris, C. S. Ang eds. London, Hershey: Information Science Reference, 2009. 4 Vols. 2734 P.
- 5) Human-Computer Interaction. Fundamentals / A. Sears, J. A. Jacko eds. CRC Press, 2009. 331 P.
- 6) Andrews K. Human-Computer Interaction. Lecture Notes. Graz University of Technology, 2009. 181 P.
- 7) Sharp H., Rogers Y., Preece J. Interaction Design: Beyond Human-Computer Interaction. Wiley, 2007. 776 P.
- 8) Berkshire Encyclopedia of Human-Computer Interaction / W. S. Bainbridge ed. Berkshire Publishing Group LLC, 2004. 2 Vols. 931 P.
- 9) Dix A., Finlay J., Abowd G., Beale R. Human Computer Interaction, 3rd Edition. Prentice Hall, 2004. 817 P.

Учебное издание

Екатерина Юрьевна Мерзлякова

**Визуальное программирование и человеко-машинное
взаимодействие**

Редактор
Корректор

Подписано в печать 01.01.2021.
Формат бумаги 62 × 84/16, отпечатано на ризографе, шрифт № 10,
п. л. – 2,3, заказ № , тираж – 50.
Отдел рекламы и PR СибГУТИ
630102, г. Новосибирск, ул. Кирова, 86, офис 107, тел. (383) 269-83-18