ФГБОУ ВПО «Воронежский государственный технический университет»
Кафедра компьютерных интеллектуальных технологий проектирования

324-2014

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 4-8 по дисциплине
“Объектно-ориентированное программирование”
для студентов направления 230100.62
«Информатика и вычислительная техника»
очной формы обучения

Воронеж 2014

Составители: канд. техн. наук А.Н. Юров,
канд. техн. наук М.В. Паринов,
ст. преп. В.А. Рыжков,
ст. преп. А.А. Филимонова

УДК 004.9

Методические указания к лабораторным работам № 4-8 по дисциплине “Объектно-ориентированное программмирование” для студентов направления 230100.62 «Информатика и вычислительная техника» очной формы обучения / ФГБОУ ВПО «Воронежский государственный технический университет»; сост. А.Н. Юров, М.В. Паринов, В.А. Рыжков, А.А. Филимонова. Воронеж, 2014. 36 с.

Методические указания содержат материал по созданию кроссплатформенных приложений в среде QT, а также практические задачи и перечень заданий для выполнения лабораторных работ по дисциплине «Объектно-ориентированное программирование».

Предназначены для студентов 1,2 курсов.

Методические указания подготовлены в электронном виде в тектовом редакторе MS Word 2010 и содержатся в файле МУ 2014_2.docx.

Табл. 6. Ил. 26. Библиогр.: 10 назв.

Рецензент канд. физ.-мат. наук, доц. Н.А. Тюкачев

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. М.И. Чижов

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

ВВЕДЕНИЕ

Возможности программных средств разработки по визуальному проектированию графических интерфейсов приложений существенно сокращают время сознания программных систем. В случае кроссплатформенного подхода к разработке приложений, программное обеспечение легче адаптировать к работе в той или иной операционной системе.

В методических указаниях представлен материал по созданию приложений в среде QT как для разработки программ в консольном режиме, так и с графическим интерфейсом с использованием QT SDK. Реализация проектов позволит усвоить концепции объектно-ориентированного программирования на практике. Все примеры могут быть апробированы на известных операционных системах: Windows, Linux, Mac OS X, Android и ряда других.

ЛАБОРАТОРНАЯ РАБОТА № 4
СТРУКТУРА КЛАССОВ QT. РАБОТА С ВИДЖЕТАМИ

Цель работы: освоить приемы по работе с виджетами - создание, удаление, изменение свойств объектов.

Задачи и требования к выполнению:

1.Изучить структуру классов QT, иметь общее представление об объектах, отвечающих за построение интерфейсной части приложения.

2.Изучить свойства и методы некоторых классов, построить на их основе приложение с графическим интерфейсом.

Теоретические сведения

Разработчики инструментального пакета QT при создании продукта руководствовались тем, что компоненты, которые будут использоваться программистами, должны иметь простую структуру их вызова, добавления в проект и изменения свойств таким образом, чтобы решить поставленные задачи на уровне интерфейса независимо от платформы ЭВМ и операционной системы.

Иерархия классов QT включает свыше 400 компонентов, поэтому для их детального изучения потребуется обращение к документации. На рис. 1 представлены лишь некоторые из них.

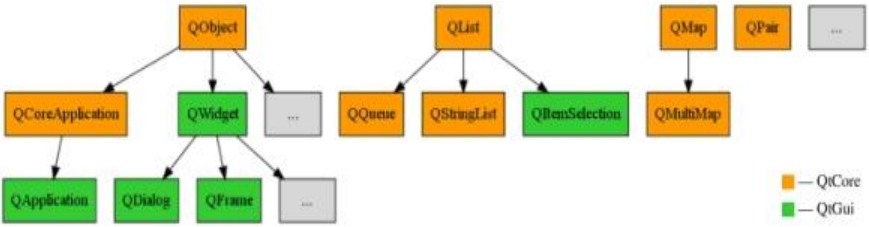

Рис. 1. Часть классов из пакета QT

Чтобы лучше представить структуру классов QT, необходимо дать определения для группы объектов.

Класс QObject, является базовым для всех классов Qt, позволяет строить иерархии объектов и обрабатывать события.

Класс QWidget является базовым для всех элементов управления. Этот класс содержит множество полей и методов, необходимых элементам управления, например, методы изменения размера или перемещения объекта. Объекты управления могут вкладываться друг в друга, при этом объект-контейнер может использовать менеджер размещения (QLayout).

Класс QFrame расширяет возможности QWidget в плане отображения рамки вокруг элемента управления и является базовым для них, нуждающихся в особой рамке. Наследниками класса QFrame являются, например, элемент панели инструментов (QToolBox), элемент видовой прокрутки (QAbstractScrollArea) или элемент вывода текста/изображений (QLabel).

Часто, вместо комбинации “элемент управления”, употребляется слово виджет. Под виджетами понимаются все объекты, из которых создается пользовательский интерфейс. Виджеты способны обрабатывать поступающие сигналы и события, отсылать свои сигналы. Все базовые виджеты Qt унаследованы от класса QWidget. В большинстве случаев тот или иной виджет определен некоторой областью, для которой доступна группа свойств, например, по изменению его размера, выбору позиции отображения или установка цвета. На рис. 2 приведена схема виджета с набором свойств, при условии, что виджет принадлежит клиентской области окна приложения.

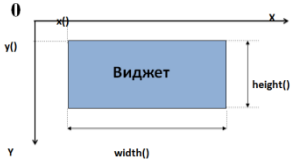

Рис. 2. Представление виджета в проекте приложения

Некоторые свойства, приведенные в табл. 1, могут быть переопределены у ряда элементов управления.

Таблица 1

Свойства элементов управления

Метод	Описание
void setEnabled(bool b)	В зависимости от параметра делает виджет доступным или недоступным
bool isEnabled()	Возвращает текущее состояние виджета
void setWindowTitle("")	Устанавливает надпись в заголовок окна
int height()	Возвращает высоту виджета
int width()	Возвращает ширину виджета
int x()	Возвращает абсциссу виджета
int y()	Возвращает ординату виджета
void move(int x, int y)	Перемещает виджет в заданные координаты
void resize(int w, int h)	Изменяет размеры виджета
void setGeometry(int x, int y, int w, int h)	Изменяет расположение и размеры виджета

Пример использования виджета в диалоговом окне приложения представлен листингом 1.

Листинг 1. Приложение с диалоговым окном на QT.

```
//Проектный файл .pro
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = EasyDialog
TEMPLATE = app
SOURCES += main.cpp\
```

```
        dialog.cpp
HEADERS  += dialog.h
```

//Заголовочный файл .h

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
#include <QLabel>
#include <QVBoxLayout>

class Dialog : public QDialog
{
    Q_OBJECT
public:
    Dialog(QWidget *parent = 0);
    ~Dialog();

private:
    QLabel *mylabel;
    QVBoxLayout *contVBox;

};
#endif // DIALOG_H
```

//Файл с реализацией методов класса .cpp

```
#include "dialog.h"

Dialog::Dialog(QWidget *parent)
    : QDialog(parent)
{
   contVBox=new QVBoxLayout(this);
   mylabel=new QLabel();
   mylabel->setText("Новый текст");
   contVBox->addWidget(mylabel);
}
```

```
Dialog::~Dialog()
{
}
```

//Файл main.cpp

```
#include "dialog.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Dialog w;
    w.show();
    return a.exec();
}
```

Приложение в диалоговом окне выводит надпись “Новый текст”. Реализация вывода сделана в файле .cpp, где создается контейнер вертикальной компоновки виджетов (contVBox от QVBoxLayout), а также метка (mylabel QLabel). После чего производится заполнение текстом поля метки, следом - помещение ее в контейнер. Результаты работы приложения показаны на рис. 3.

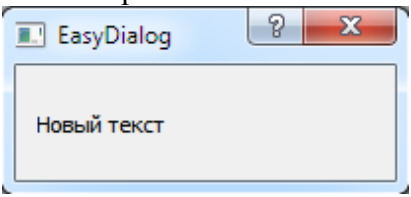

Рис. 3. Диалоговое окно с текстом

Набор элементов отображения в диалоговом окне может быть представлен рядом виджетов, описание которых приведено далее по тексту.

QLabel – виджет, способный отображать текстовую и графическую информацию (рис. 4).

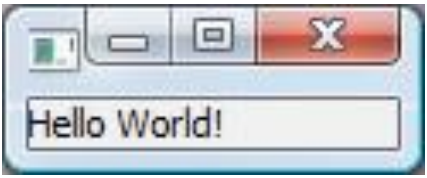

Рис. 4. Виджет QLabel

Набор свойств и примеры создания виджета QLabel приведен в табл. 2.

Таблица 2

Работа с виджетом QLabel

Строка программного кода	Описание
QLabel* lb = new QLabel("text");	Пример создания
QLabel* lb = new QLabel();	Пример создания
lb->setText("text");	Устанавливает текст в виджет, текст можно форматировать html тэгами
lb->setPixmap(QPixmap);	Устанавливает изображение в виджет

QLCDNumber – виджет, отображающий численные значения в виде семисегментного индикатора(рис. 5).

Рис. 5. Виджет QLCDNumber

Набор свойств и примеры создания виджета QLCDNumber приведен в табл. 3.

Таблица 3

Работа с виджетом QLCDNumber

Строка программного кода	Описание
QLCDNumber* num = new QLCDNumber();	Пример создания
num->display(int);	Задает целочисленное значение индикатора
num->display(double);	Задает вещественное значение индикатора

QSlider – Горизонтальный или вертикальный элемент сдвига(слайдер). Действие от данного элемента управления влияет на другие виджеты оконного интерфейса (рис. 6).

Рис. 6. Виджет QSlider

Набор свойств и примеры создания виджета QSlider приведен в табл. 4.

Таблица 4

Работа с виджетом QSlider

Строка программного кода	Описание
QSlider* slider = new QSlider(Qt::Orientation);	Пример создания, где "Qt::Orientation" либо "Qt::Horizontal" либо "Qt::Vertical"
slider->setOrientation(Qt::Orientation);	Изменить ориентацию ползунка
slider->setMinimum(int min);	Задание минимального значения
slider->setMaximum(int max);	Задание максимального значения
slider->setRange(int min, int max);	Задание диапазона
slider->setValue(int v);	Текущее значение
int v = slider->value();	Возвращает текущее значение

QSpinBox – счетчик (рис. 7).

Рис. 7. Виджет QSpinBox

Набор свойств и примеры создания виджета QSpinBox приведен в табл. 5.

Таблица 5

Работа с виджетом QSpinBox

Строка программного кода	Описание
QSpinBox* spin = new QSpinBox();	Пример создания
spin->setMinimum(int min);	Задание минимального значения
spin->setMaximum(int max);	Задание максимального значения
spin->setRange(int min, int max);	Задание диапазона
spin->setValue(int v);	Определяется текущее значение
spin->setPrefix("");	Задание префикса
spin->setSuffix("");	Задание суффикса

QLineEdit - Текстовое поле (рис. 8).

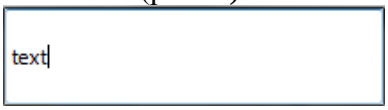

Рис. 8. Виджет QLineEdit

Набор свойств и примеры создания виджета QLineEdit приведен в табл. 6.

Таблица 6

Работа с виджетом QLineEdit

Строка программного кода	Описание
QLineEdit *text = new QLineEdit();	Пример создания
text->setReadOnly(bool);	Только для чтения
QString s = text->text();	Возвращение текста

Пример по использованию виджетов представлен фрагментом листинга 2.

Листинг 2. Фрагмент приложения по работе с виджетами.

```
#include "mainwindow.h"
#include <QtGui>
MainWindow::MainWindow(QWidget *parent) :
QMainWindow(parent)
{
 // Объявление и вызов конструктора
QLabel *lb = new QLabel();
// Задание текста
lb->setText("text");
// Объявление и вызов конструктора
QLCDNumber *lcd = new QLCDNumber();
lcd->display(20); // Установка значения
// Объявление и вызов конструктора
QHBoxLayout *layout1 = new QHBoxLayout();
layout1->addWidget(lb);
// Добавление виджета на слой
layout1->addWidget(lcd);
// Объявление и вызов конструктора
QLineEdit *line = new QLineEdit();
line->setText("Type text"); // Задание текста
// Объявление и вызов конструктора
QVBoxLayout *layout2 = new QVBoxLayout();
layout2->addWidget(line); // Добавление виджета
// Добавление лейаута на лейаут
layout2->addLayout(layout1);
// Объявление и вызов конструктора виджета "central" -
//он будет основным виджетом приложения
QWidget*central=new QWidget(this);
// Установка лейаута на виджет
central->setLayout(layout2);
// Установка виджета "central" в качестве основного
//виджета приложения
setCentralWidget(central);     }
// Деструктор
MainWindow::~MainWindow() { }
```

Задания на самостоятельную работу:

Разработать приложение с графическим интерфейсом, в котором реализовать:

-работу над проектом без формы QT Designer;

-добавить любой виджет (QLabel, QPushButton, QLCDNumber, QSlider и т.п.);

-сделать его главным виджетом приложения с помощью метода setCentralWidget();

-найти виджеты и добавить их в приложение из справки по QT.

ЛАБОРАТОРНАЯ РАБОТА № 5
МЕХАНИЗМ РАБОТЫ СИГНАЛОВ И СЛОТОВ В QT

Цель работы: освоить подходы по организации взаимосвязей между объектами графического интерфейса посредством решений управления объектами - сигналов и слотов.

Задачи и требования к выполнению:

1.Изучить базовые определения и понять принципы управления объектами посредством механизмов, реализованных в QT вне зависимости от конкретной операционной системы.

2.Изучить приемы, а также ознакомиться с практикой применения сигналов-слотов в учебных проектах.

Теоретические сведения

Главной особенностью библиотеки Qt является технология сигналов и слотов (signals and slots). Сигнал –это метод без реализации, который производит формирует

сообщение (например, нажатие на кнопку). Слот - это функция, вызываемая в ответ на определенный сигнал. Класс, испускающий сигналы, не знает, который из слотов получит сигнал. Сигналы и слоты могут иметь любое количество аргументов любых типов.

По своей природе, слоты очень близки к обычным функциям-членам в языке C++. Они могут быть виртуальными, они могут подвергаться перегрузке, они могут быть публичными, защищенными или приватными и они могут вызываться напрямую, как и обычные функции-члены.

Отличие состоит в том, что слот может быть подключен к сигналу. В этом случае, функция-слот вызывается автоматически всякий раз, когда посылается сигнал. На рис. 9 представлена схема взаимного влияния одних объектов интерфейса на другие посредством механизма управления, реализованного в QT.

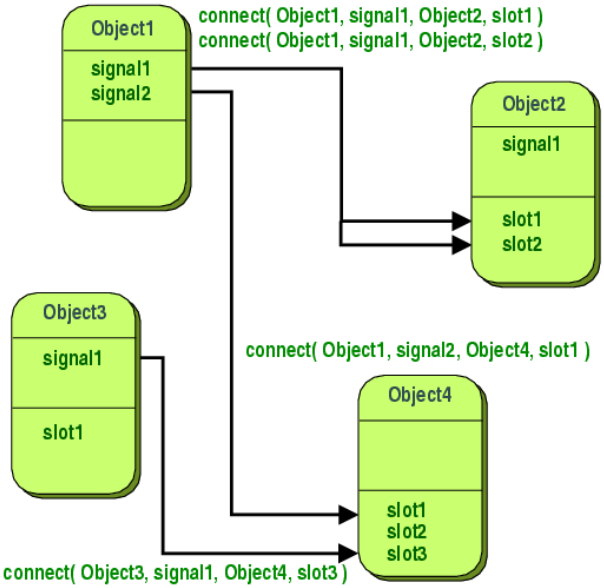

Рис. 9. Схема организации взаимосвязей между объектами в QT

Соединение сигналов и слотов осуществляется:

-в классах наследниках от QObject функцией connect();

-в прочих местах программы с помощью статической функции-члена QObject::connect();

Аргументами этой функции являются:

-указатель на объект-отправитель;

-сигнал объекта-отправителя;

-указатель на объект-получатель;

-слот объекта-получателя;

-тип соединения.

При соединении сигналов и слотов можно передавать параметры от сигнала к слоту, если существует соответствующая пара сигнал/слот.

Свойства для сигналов:

-методы сигналов не возвращают значений, поэтому перед названием метода указывается void (примеры: void mySignal(), void mySign(QString &str));

-сигналы не обязательно соединять со слотом, можно связывать сигналы между собой;

-отправку сигналов можно заблокировать с помощью метода blockSignals() с параметром true.

Пример по использованию механизма сигнал-слот представлен фрагментом листинга 3.

Листинг 3. Фрагмент класса с реализаций сигналов-слотов.

Файл widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QPushButton>
```

```
#include <QHBoxLayout>
#include <QLCDNumber>
#include <QDebug>
#include <QMessageBox>

class Widget : public QWidget
{
    Q_OBJECT

public:

    QPushButton *myButton4;
    QPushButton *myButton1;
    QHBoxLayout *myBox;
    QLCDNumber *myLCD;

    int temp;
    bool flag1;
    bool flag4;
    Widget(QWidget *parent = 0);
    ~Widget();

 private slots:
    void MyAction4();
    void MyAction1();
};

    #endif // WIDGET_H
```

Файл widget.cpp

```
#include "widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
```

```
{

    myButton4=new QPushButton("4");
    myButton1=new QPushButton("1");
    myLCD=new QLCDNumber();
    myBox=new QHBoxLayout(this);
    myBox->addWidget(myButton4);
    myBox->addWidget(myButton1);
    myBox->addWidget(myLCD);


QObject::connect(myButton4,SIGNAL(clicked()),
this,SLOT(MyAction4()));

QObject::connect(myButton1,SIGNAL(clicked()),
this,SLOT(MyAction1()));
}

Widget::~Widget()
{
}

void Widget::MyAction4()
{
  flag1=false;
  if (flag4==false) {flag4=true; temp=4;}
  qDebug()<<"Нажата кнопка 4";
  myLCD->display(temp);
  temp*=10;
  temp+=4;
  }
void Widget::MyAction1()
{
  flag4=false;
  if (flag1==false) {flag1=true; temp=1;}
```

```
qDebug()<<"Нажата кнопка 1";
myLCD->display(temp);
temp*=10;
temp++;
  }
```

В данном примере показана реализация вывода цифры (1 или 4) в виджет QLCDNumber по нажатию кнопки с соответствующей цифрой. Кроме того, в окно среды QT поступает информационное сообщение. Такие возможности предоставляет класс QDebug. На рис. 10 показаны результаты работы приложения.

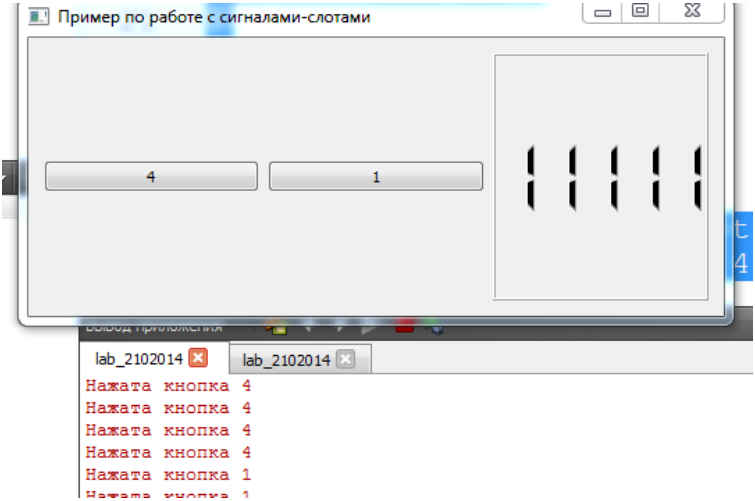

Рис. 10. Результаты работы приложения с организацией взаимосвязей между объектами

Задания на самостоятельную работу:

Разработать приложение с графическим интерфейсом, в котором реализовать:

- используя форму диалога, добавить требуемые виджеты на форму, получив тем самым прототип калькулятора;
- используя механизм сигналов-слотов, разработать приложение, в котором будут производиться простейшие арифметические действия, а также осуществляться перевод чисел в различные системы счисления.

ЛАБОРАТОРНАЯ РАБОТА № 6

СРЕДСТВО ВИЗУАЛЬНОЙ РАЗРАБОТКИ ИНТЕРФЕЙСОВ В QT- QT DESIGNER

Цель работы: приобрести практические навыки работы по графическому редактору QT Designer.

Задачи и требования к выполнению:

1.Изучить средства QT Designer по созданию и наполнению форм элементами управления.

2.Изучить визуальные механизмы добавления взаимосвязей между элементами управления (средства автоматизации концепции сигнал-слот).

3.Изучить структуру свойств некоторых виджетов средствами QT Designer.

Теоретические сведения

Qt Designer направлен на ускоренную разработку интерфейсных возможностей управления и контроля в приложениях. Для этих в дизайнере предусмотрены инструментальные средства такие как «Панель виджетов», в которой доступны для использования элементы интерфейса - виджеты: «выпадающий список» ComboBox, «поле ввода» LineEdit, «кнопка» PushButton и многие другие. Каждый

виджет имеет свой набор свойств, определяемый соответствующим ему классом библиотеки Qt. Свойства виджета могут быть изменены при помощи «Редактора свойств». Для каждого класса свойств виджета существует свой специализированный редактор. Характерной особенностью Qt Designer является поддержка визуального редактирования сигналов и слотов. На рис. 11 представлен графический редактор QT Designer с описанием его функциональных блоков.

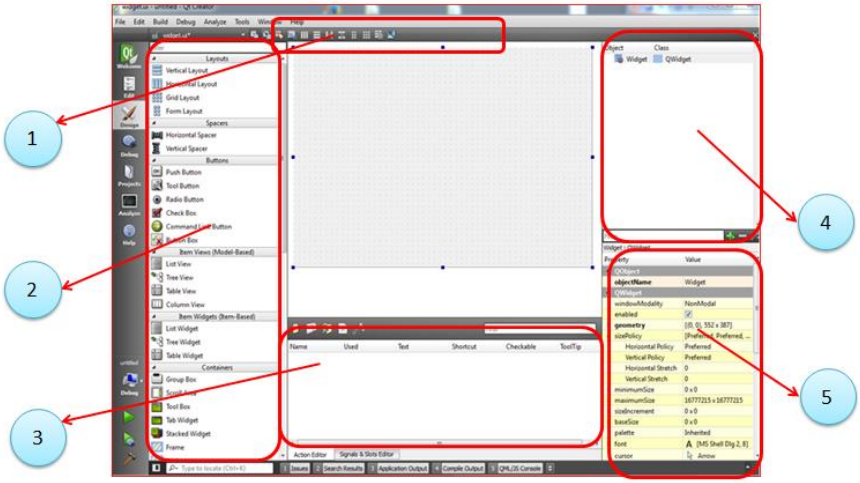

Рис. 11. Графический редактор QT Designer

1-Панель выбора работы с виджетами, событиями, компоновкой виджетов;

2-Набор виджетов , объединённые в группы по назначению;

3-Установка и связь событий в приложении;

4-Перечень объектов из классов, которые внесены в приложение;

5-Свойства отдельно выбранного объекта

Добавление виджета на форму осуществляется “переносом” в предполагаемое положение элемента управления в интерфейсное окно приложения как показано на рис. 12.

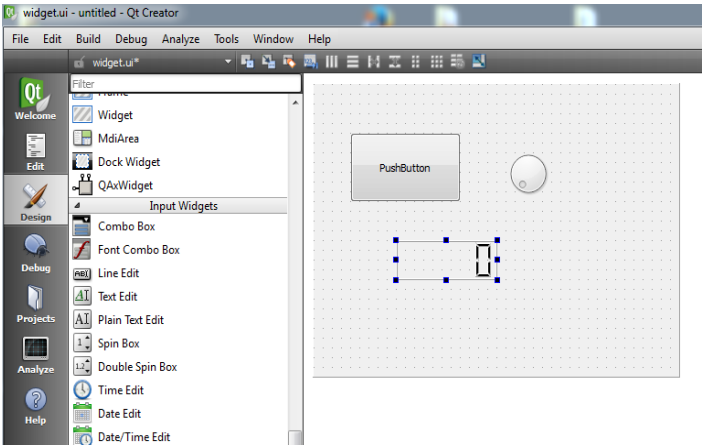

Рис. 12. Добавление виджетов на форму

Связи между компонентами можно реализовать также средствами QT Designer. Например, для виджета “PushButton” выбирается пукт меню “перейти к слоту”, а в дополнительном диалоге указывается сигнал и действие, которое последует после сигнала (рис. 13).

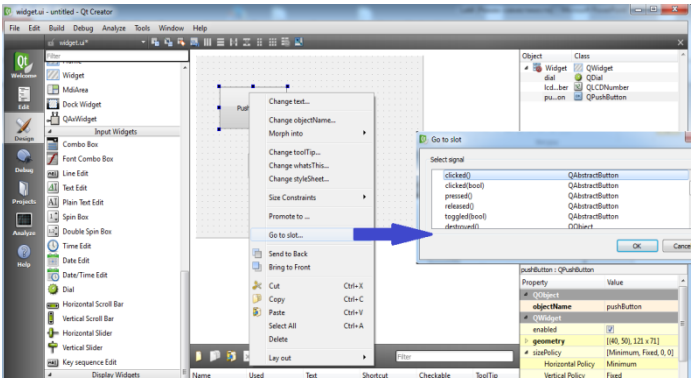

Рис. 13. Реализация события на заданный элемент управления

Задания на самостоятельную работу:

Разработать приложение с графическим интерфейсом, в котором реализовать:

-форму диалогового окна;

-добавить дизайнером требуемые виджеты на форму, получив тем самым прототип интерфейса;

-подготовить модуль ведения диалога с пользователем;

-реализовать функционал по игре в города.

ЛАБОРАТОРНАЯ РАБОТА № 7
ИСПОЛЬЗОВАНИЕ В ПРИЛОЖЕНИЯХ СТАНДАРТНЫХ ДИАЛОГОВЫХ ОКОН

Цель работы: приобрести практические навыки работы с диалоговыми окнами в среде QT.

Задачи и требования к выполнению:

1.Изучить возможности базовых классов по вызову стандартных диалогов управления в QT.

2.Изучить приемы передачи заполненных структур данных из стандартных диалоговых окон в методы разрабатываемого проекта.

Теоретические сведения

Применение стандартных окон значительно ускоряет разработку приложений, в которых необходимо использовать диалоговые окна выбора файлов, шрифта, цвета и т. д.

Преимущества:

-однотипность пользовательского интерфейса;

-узнаваемость;

-сокращение времени на разработку программного обеспечения;

-дружественность интерфейса.

Типы стандартных диалоговых окон показаны на рисунках 14-17.

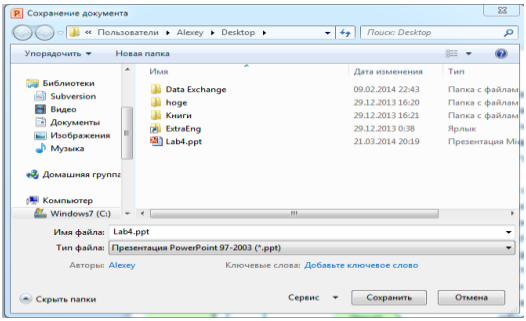

Рис. 14. Диалог записи-чтения файлов

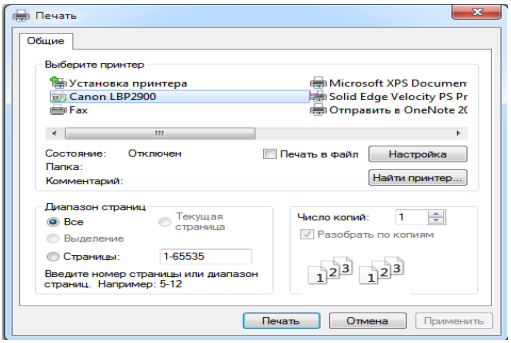

Рис. 15. Диалог выбора устройства печати

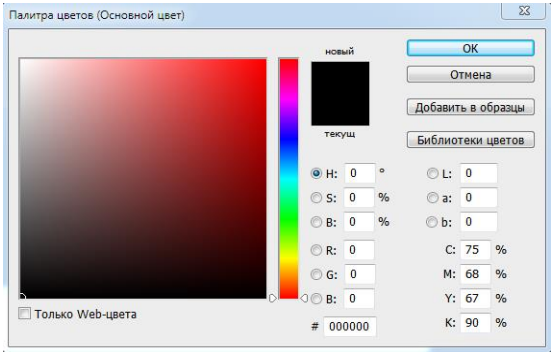

Рис. 16. Диалог выбора цвета для закраски некоторой области

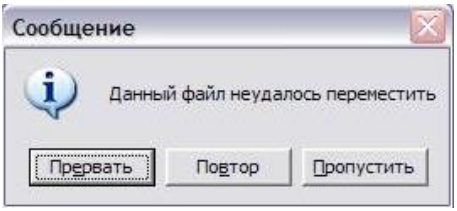

Рис. 17. Информационное сообщение

QMessageBox - класс, который позволяет вывести текстовое сообщение в диалоговое окно и ожидает реакции со стороны пользователя. Все окна, предоставляемые классом QMessageBox, — модальные. Они могут содержать кнопки, заголовок и текст сообщения. Фрагмент листинга 4 выводит сообщение одной строкой (при этом используется разметка текста) в диалоговое окно с тремя кнопками, как показано на рис. 18.

Листинг 4. Фрагмент листинга по отображению диалогового окна с информационным сообщением.

```
QMessageBox* pmbx =new QMessageBox
("MessageBox","<b>A</b> <i>Simple</i>
<u>Message</u>",QMessageBox::Information,
                QMessageBox::Yes,
                QMessageBox::No,
 QMessageBox::Cancel | QMessageBox::Escape);
 int n = pmbx->exec();
 delete pmbx;
```

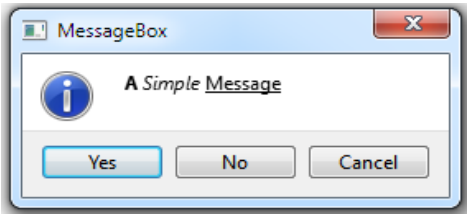

Рис. 18. Информационное сообщение с разметкой текста

Класс QProgressDialog**.** Для диалогового окна прогресса Qt предоставляет класс QProgressDialog, унаследованный от класса QDialog. Это окно информирует пользователя о начале продолжительной операции и дает возможность визуально оценить время работы. Окно может содержать кнопку Cancel (Отмена) для прерывания начатой операции. Фрагмент листинга 5 реализует вывод указанного типа диалогового окна (при этом в приложении остается возможность обрабатывать сообщения внутри проекта), а на рис. 19 показаны результаты работы.

Листинг 5. Фрагмент листинга по отображению диалогового окна с выводом шкалы по продолжительности некоторых действий.

```
#include <QProgressDialog>

int n = 100000;
    QProgressDialog* pprd = new
QProgressDialog("Progressing the data...",
"&Cancel", 0, n);
    pprd->setMinimumDuration(0);
    pprd->setWindowTitle("Please Wait");
    for (int i = 0; i < n; ++i)
    {
        pprd->setValue(i) ;
        qApp->processEvents();
        if (pprd->wasCanceled())
        {
             break;
        }
    }
    pprd->setValue(n) ;
    delete pprd;
```

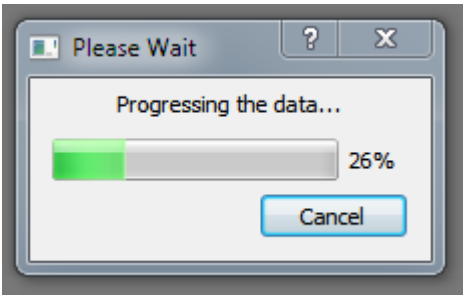

Рис. 19. Диалог по отображению продолжительности действий

Диалоговое окно выбора шрифта QFontDialog. Окно предназначено для выбора одного из зарегистрированных в системе шрифтов, а также для задания его стиля и размера. Реализация этого диалогового окна содержится в классе QFontDialog, определенном в заголовочном файле QFontDialog. Для того чтобы показать диалоговое окно, в большинстве случаев, можно обойтись методом QFontDialog::getFont(). Пример реализации представлен фрагментом листинга 6, а результаты работы показаны на рис. 20. Необходимость подключения заголовочных файлов с классами в проекте обязательна, как и в предыдущих случаях, иначе сами классы и их методы в проекте останутся недоступными.

Листинг 6. Фрагмент листинга по отображению диалогового окна выбора шрифта.

```
#include <QFont>
#include <QFontDialog>
bool ok;
    QFont font = QFontDialog::getFont(&ok, 0);
    if (ok)
 {
//    пользователь    нажимает    OK,    и    шрифт
устанавливается в //выбранный
```

```
  }
else
{
  //  пользователь  нажимает  отменить,  шрифт
остаётся прежним
    }
```

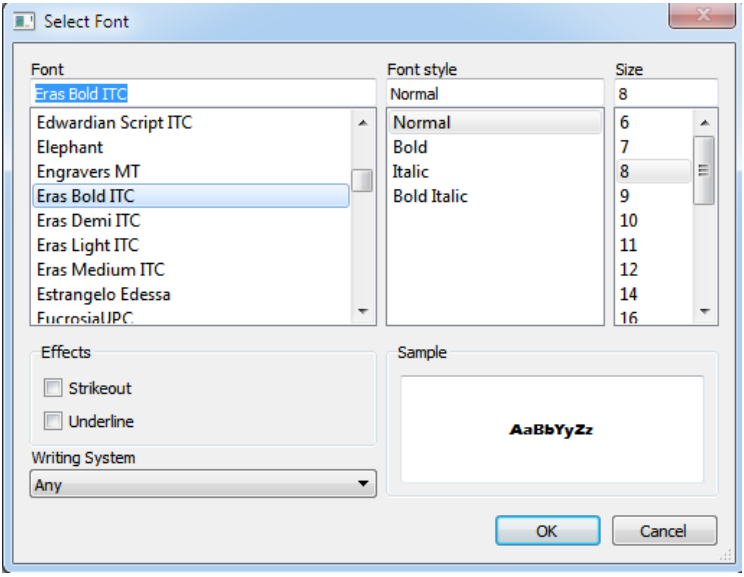

Рис. 20. Диалоговое окно выбора шрифта QFontDialog

Класс QColorDiaiog реализует диалоговое окно выбора цвета. Для того чтобы показать это окно, вызывается статический метод getcolor(). Первым параметром в метод можно передать цветовое значение для инициализации. Вторым параметром является указатель на виджет предка. После закрытия диалогового окна метод возвращает номер цвета. Чтобы узнать, какой кнопкой было закрыто окно — Ok или Cancel (Отмена), необходимо вызвать метод isValid() из возвращенного этим методом объекта типа QColor. Пример реализации представлен фрагментом листинга 7, а результаты работы показаны на рис. 21. Необходимость подключения

заголовочных файлов с классами в проекте обязательна, как и в предыдущих случаях, иначе сами классы и их методы в проекте останутся недоступными.

Листинг 7. Фрагмент листинга по отображению диалогового окна выбора цвета.

```
#include <QColor>
#include <QColorDialog>
QColor color = QColorDialog::getColor();
    if (!color.isValid())
    {
    // Cancel
    }
    QColorDialog::getColor(Qt::red,0);
```

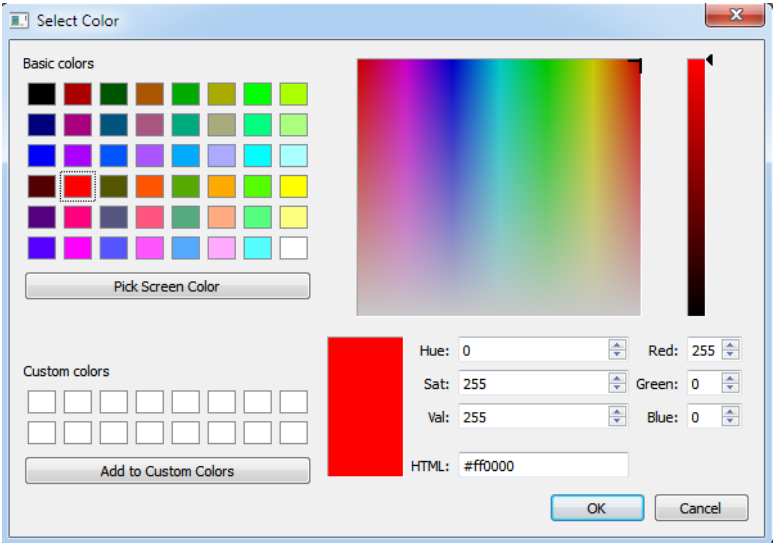

Рис. 21. Диалоговое окно выбора цвета в Qt

Задания на самостоятельную работу:

Разработать приложение с графическим интерфейсом, в котором реализовать:

-с помощью диалогового окна выбора цвета переопределить цвет всех кнопок приложения (для кнопок использовать класс QPushButton);

-с помощью диалога шрифтов переопределить текстовые записи в приложении.

ЛАБОРАТОРНАЯ РАБОТА № 8
ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА В QT

Цель работы: освоить разработку и использование вспомогательных элементов управления, а также правила их применения в учебных проектах.

Задачи и требования к выполнению:

1.Изучить возможности по использованию элементов меню, строки состояния, проектированию панели инструментов в приложении.

2.Изучить возможности по внесению в разрабатываемый проект подсказок к элементам управления, интерактивные средства помощи по программному продукту.

Теоретические сведения

Интерфейс определяет уровень интерактивности при работе с приложением. При разработке приложений целесообразно продумать компоновку элементов управления согласно цели проекта.

Основное условие при разработке приложений - интерфейсную часть изменить или доработать относительно базового варианта, предложенного QT Creator, а прочие методы и классы, которые разрабатывались под конкретные задачи, разместить в отдельных файлах.

Предоставление помощи

Главная задача помощи состоит в обеспечении пользователя всей необходимой информацией о приложении и его элементах, что делает работу пользователя более удобной.

Всплывающая подсказка

При задержке указателя мыши над элементами интерфейса автоматически появляется небольшое текстовое окошко, поясняющее назначение виджета.

```
    QPushButton* pcmd = new QPushButton
("&Ok");
    pcmd->setToolTip("Button");
    ////////////////////////////
    ui->MyButton->setToolTip("Подсказка");
```

Результаты работы программного кода, приведенного выше, показаны на рис. 22.

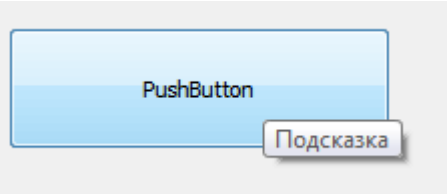

Рис. 22. Подсказка к элементу управления QPushButton

Подсказка "Что это"

Иногда требуется отобразить больше информации о виджете, чем способна предоставить воздушная подсказка. Подсказка What's this (Что это) является промежуточной между воздушной подсказкой и системой помощи. Для подключения подсказки необходимо в главном окне вывести значок по получению контекстной помощи, а непосредственно

к элементу управления привязать функцию для вывода подсказки.

```
    int main(int argc, char *argv[]) {
     QApplication a(argc, argv);
    Widget w;
    w.setWindowFlags(Qt::WindowContextHelpBut
tonHint);
    w.show();
    return a.exec(); }
    ////////////////////////////////////////
////////////
    ui->MyButton-
>setWhatsThis("<I>Пояснение</I><BR><B>      к
виджету</B>");
```

Результаты работы программного кода, приведенного выше, показаны на рис. 23.

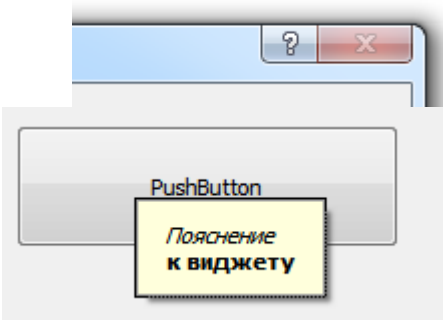

Рис. 23. Подсказка к элементу управления QPushButton

Создание меню в Qt

Меню является важной и неотъемлемой частью практически любого приложения. Оно находится в верхней части главного окна приложения и представляет собой секцию для расположения большого количества команд, из которых пользователь может выбирать нужную. В приложениях могут быть применены меню четырех основных типов:

-меню верхнего уровня;

-всплывающее меню;

-отрывное меню, которое можно отделять от основного;

-контекстное меню.

Пример с реализацией в проекте меню представлен листингом 8, а результаты показаны на рис. 24.

Листинг 8. Реализация меню в приложении Qt

```
#include "widget.h"
#include <QApplication>
#include <QMenuBar>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    //Widget w;
    //w.show();
     QMenuBar mnuBar;
     QMenu*          pmnu          =    new
QMenu("&Menu");
     pmnu->addAction("&About Qt",
                      &a,
                      SLOT(aboutQt()),
                      Qt::CTRL + Qt::Key_Q
                     );
     pmnu->addSeparator();
     QAction*  pCheckableAction  =  pmnu-
>addAction("&CheckableItem");
     pCheckableAction-
>setCheckable(true);
     pCheckableAction->setChecked(true);
     QMenu*     pmnuSubMenu     =     new
QMenu("&SubMenu", pmnu);
     pmnu->addMenu(pmnuSubMenu);
```

```
        pmnuSubMenu->addAction("&Test");
        QAction*  pDisabledAction  =  pmnu-
>addAction("&DisabledItem");
        pDisabledAction->setEnabled(false);
        pmnu->addSeparator();
        pmnu->addAction("&Exit",          &a,
SLOT(quit()));
        mnuBar.addMenu(pmnu);
        mnuBar.show();
       return a.exec();
    }
```

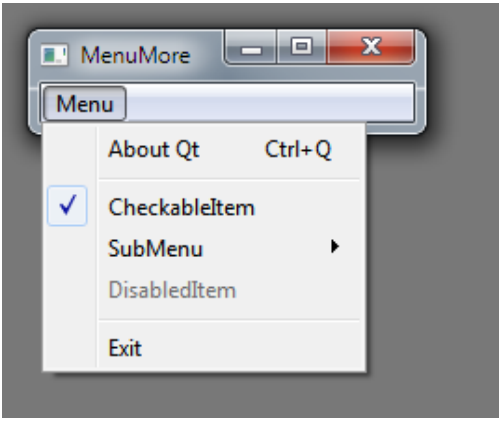

Рис. 24. Создание меню в Qt

Qt предоставляет возможность реализации отрывных меню (tear-off menu). Нажатие мышью на прерывистую линию приводит к тому, что всплывающее меню отделяется от меню верхнего уровня, превращаясь в отдельное окно, которое свободно перемещается. Такое меню очень удобно, например, для настройки конфигураций программы. Фрагмент кода представлен на листинге 9, а его результаты - на рис. 25.

Листинг 9. Реализация отрывного меню в приложении Qt

```
    QMenuBar pmnuBar;
```

```
    QMenu*    pmnu     = new QMenu("&Menu");
    pmnu->setTearOffEnabled(true);
    pmnu->addAction("Item&1");
    pmnu->addAction("Item&2");
    pmnu->addAction("Item&3");
    pmnu->addAction("&Exit",                &a,
SLOT(quit()));
    pmnuBar.addMenu(pmnu);
    pmnuBar.show();
```

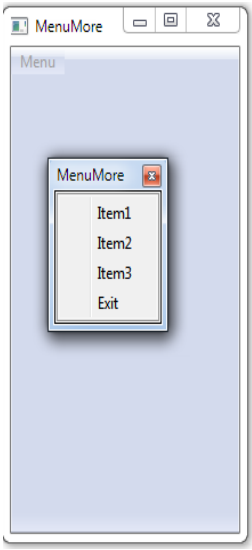

Рис. 25. Создание отрывного меню в Qt

Строка состояния QStatusBar в Qt. Этот виджет располагается в нижней части главного окна и отображает, как правило, текстовые сообщения для предоставления информации о состоянии приложения или выдачи справки о командах меню или кнопках панелей инструментов. Строку состояния реализует класс QStatusBar, определенный в заголовочном файле QStatusBar. Фрагмент кода по реализации

строки состояния в приложении представлен на листинге 10, а его результаты - на рис. 26.

Листинг 10. Строка состояния в приложении Qt

```
    #include "mainwindow.h"
    #include "ui_mainwindow.h"
    MainWindow::MainWindow(QWidget *parent) :
        QMainWindow(parent),
        ui(new Ui::MainWindow)
    {
        ui->setupUi(this);
    }
    MainWindow::~MainWindow()
    {
        delete ui;
    }
    void MainWindow::on_action_triggered()
    {
      ui->statusBar->showMessage("Отображение
сообщения",0);
    }
```

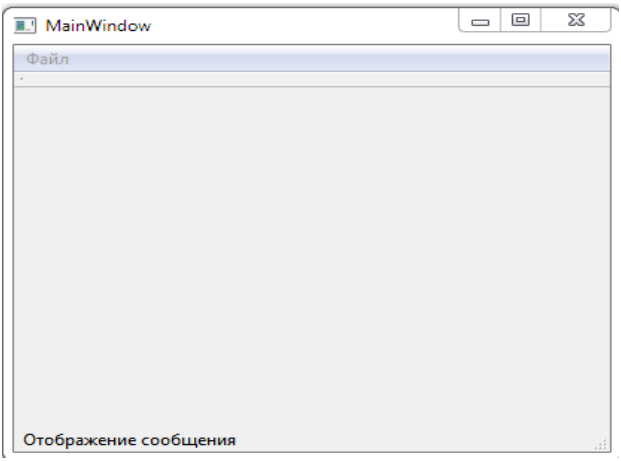

Рис. 26. Строка состояния в главном окне приложения

Задания на самостоятельную работу:

Подготовить приложение по вычислению значений функции y=x⅔ в интервале, определяемым с клавиатуры, использовав системное меню, строку состояния, панели инструментов, подсказки и прочие решения на QT.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Прата С. Язык программирования C++. Лекции и упражнения / С. Прата. 5-е изд. – М.: Вильямс, 2007. – 1184 с.
2. Страуструп Б Язык программирования C++ / Б. Страуструп. - М.: Бином, 2011. – 1136 с.
3. Шилдт Г. C++ Базовый курс / Г. Шилдт. 3-е изд. – М.: Вильямс, 2010. – 624 с.
4. Бланшет Ж. QT4: программирование GUI на C++ / Ж. Бланшет, М. Саммерфилд. - М.: Кудиц-Пресс, 2007. - 641 с.
5. Иванова Г.С. Создание пользовательских интерфейсов в программах на C++ с использованием библиотеки QT: учеб. пособие / Г. С. Иванова. - М.: МГТУ имени Н.Э. Баумана, 2011. - 52 с.
6. Шлее М. Qt 4.8 / Профессиональное программирование на C++/ М. Шлее. -СПб.: БХВ-Петербург, 2012. - 912 с.
7. Roberge J. A laboratory course in C++ structures. 2ed / J. Roberge, S. Brandl, D. Whittington. Jones and Bartlett, 2003. - 411 p.
8. London J. Modeling Derivatives in C++ / London J. Wiley, 2005. - 841p.
9. Документация библиотеки Qt [Электронный ресурс]. – Режим доступа: http://qt-project.org/doc/
10. Документация библиотеки Qt [Электронный ресурс]. – Режим доступа: http://qt-doc.ru/

СОДЕРЖАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 4-8 по дисциплине
“Объектно-ориентированное программирование”
для студентов направления 230100.62
«Информатика и вычислительная техника»
очной формы обучения

Составители:
Юров Алексей Николаевич
Паринов Максим Викторович
Рыжков Владимир Анатольевич
Филимонова Анастасия Анатольевна

В авторской редакции

Компьютерный набор А.Н. Юрова

Подписано к изданию 28.11.2014.
Уч.-изд. л. 2,2. «С»

ФГБОУ ВПО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14