Правительство Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Отчёт к лабораторной работе №1

по дисциплине

«Языки программирования»

Работу выполнила

Студент группы СКБ223 _____ Д. О. Демешко
подпись, дата

Работу проверил _____ С. А. Булгаков
подпись, дата

Содержание

Постановка задачи

Разработать программу с использованием библиотеки Qt позволяющую ознакомиться с имеющимися элементами пользовательского интерфейса. Окно программы должно содержать два элемента: органайзер (слева) и панель с текстом (справа). На вкладках органайзера расположить виджеты, сгруппировав их в соответствии с Qt: Widgets and Layouts. При наведении курсора мышки на виджет на панели теста появляется его описание.

Для реализации органайзера использовать класс QTabWidget. Для реализации панели с текстом использовать класс QFrame (задав ему режим отрисовки рамки) и класс QLabel (задав ему режим переноса строк).

1. Алгоритм решения задачи.

Для решения поставленной задачи был разработан набор классов-наследников от классов-наследников элементов графического интерфейса пользователей библиотеки QT, а также классы являющиеся вкладками органайзера. В основной функции программы создается объект класса Widget, на котором располагаются вкладки органайзера и панель с текстом.

2. Решение задачи

Все классы наследники были разработаны по одному принципу и отличаются только классом родителя.

Кдасс Widget разработан на основе класса QWidget.

Панель с текстом разработана с использованием классов QFrame и QLabel.

2.1. Класс Widget

Класс состоит из приватных полей класса QFrame* frame, QLabel* frame_text, конструктора и публичного слота changeInfo.

2.1.1 Конструктор класса Widget

На вход принимает QWidget* p. Инициализирует все поля класса, создает экземпляры, наследованные от классов, относящихся к Basic Widget(QComboBox, QCheckBox, …), Advanced Widget(QCalendarWidget, QColoumnView, QTreeView, …) и Organizer Widget(QGroupBox, QSplitter, …), совершает connect этих объектов со слотом changeInfo.

2.1.2 Деструктор класса Widget

Не делает ничего дополнительного, кроме как базовый деструктор класса QWidget.

2.1.3 Слот changeInfo

Принимает на вход QString и применяет метод setText с этим же аргументом к полю класса text.

2.2 Классы-наследники

Каждый класс-наследник наследуется от классов, относящихся либо к Basic, либо к Advanced, либо к Organizer Widget. В наследниках перегружается конструктор, защищенный метод mouseMoveEvent и создается сигнал info.

2.2.1 Конструкторы классов-наследников

Принимает на вход QWidget* p(либо другие параметры согласно документации). В конструкторе устанавливается setMouseTracking(true);

2.2.2 Защищенный метод mouseMoveEvent классов-наследников

Принимает на вход QMouseEvent *e. Вызывает info от строки с описанием класса директивой emit и выполняет mouseMoveEvent класса-родителя(QComboBox, QCheckBox, …) от указателя e.

3. Тестирование программы

Общий вид окна программы приведен на рисунке 1. Также на нем отображен общий вид вкладки Basic Widget.

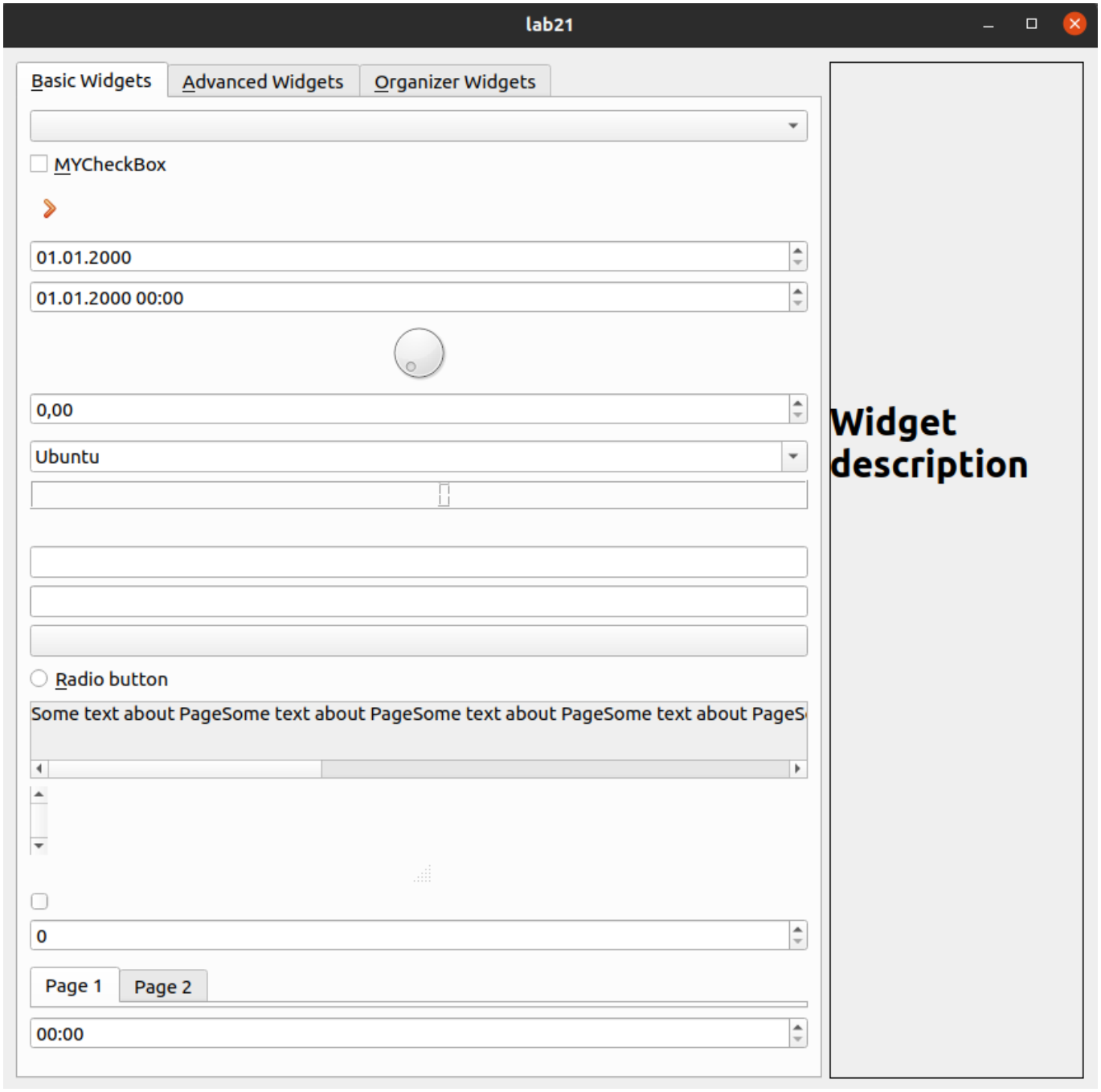

Рисунок 1

На рисунке 2 показан пример отображения информации о виджете при наведении на него курсором мыши.

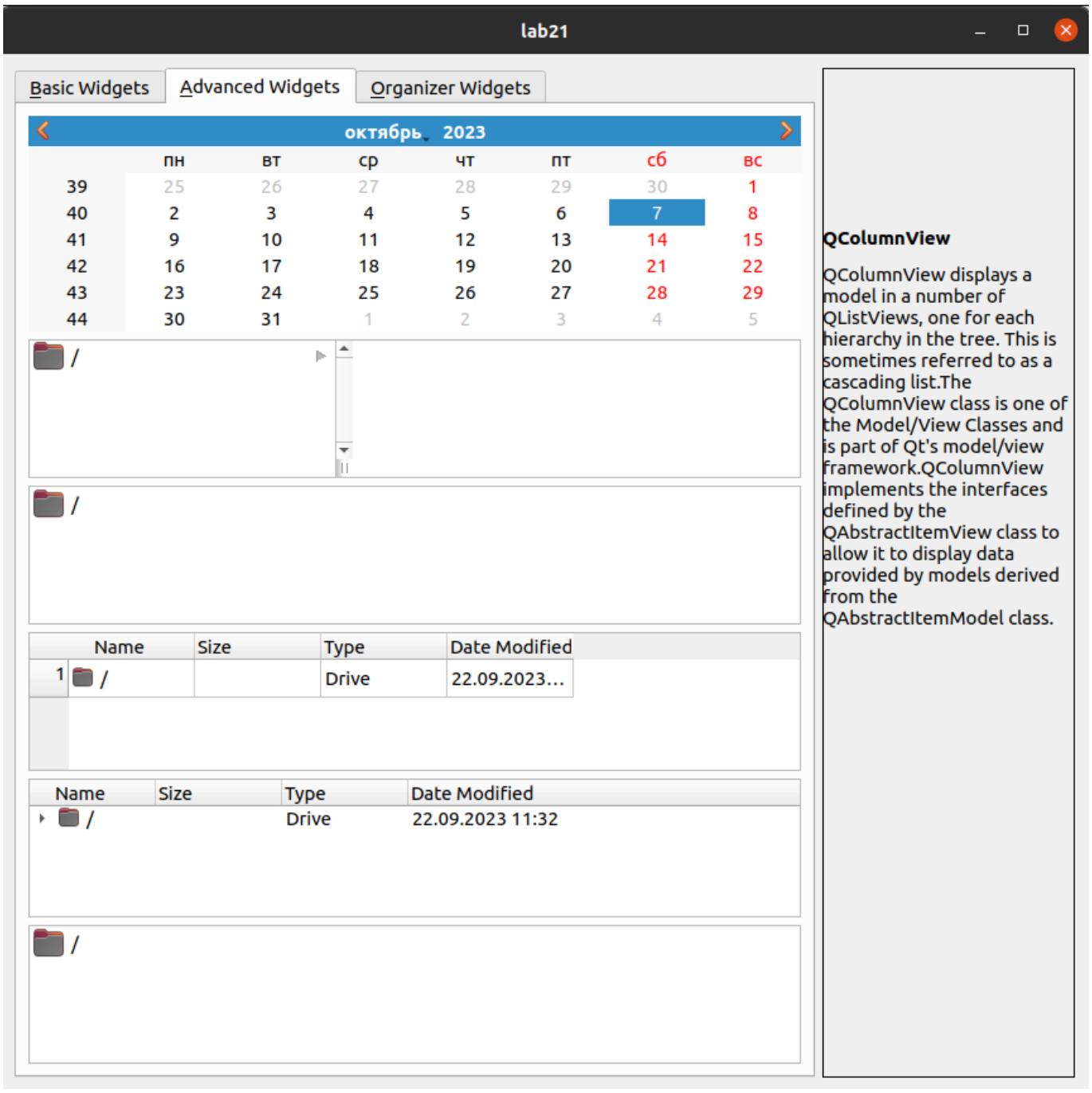

Рисунок 2

На рисунках 2 и 3 приведены общие виды вкладок Advanced Widget и Organizer Widget.

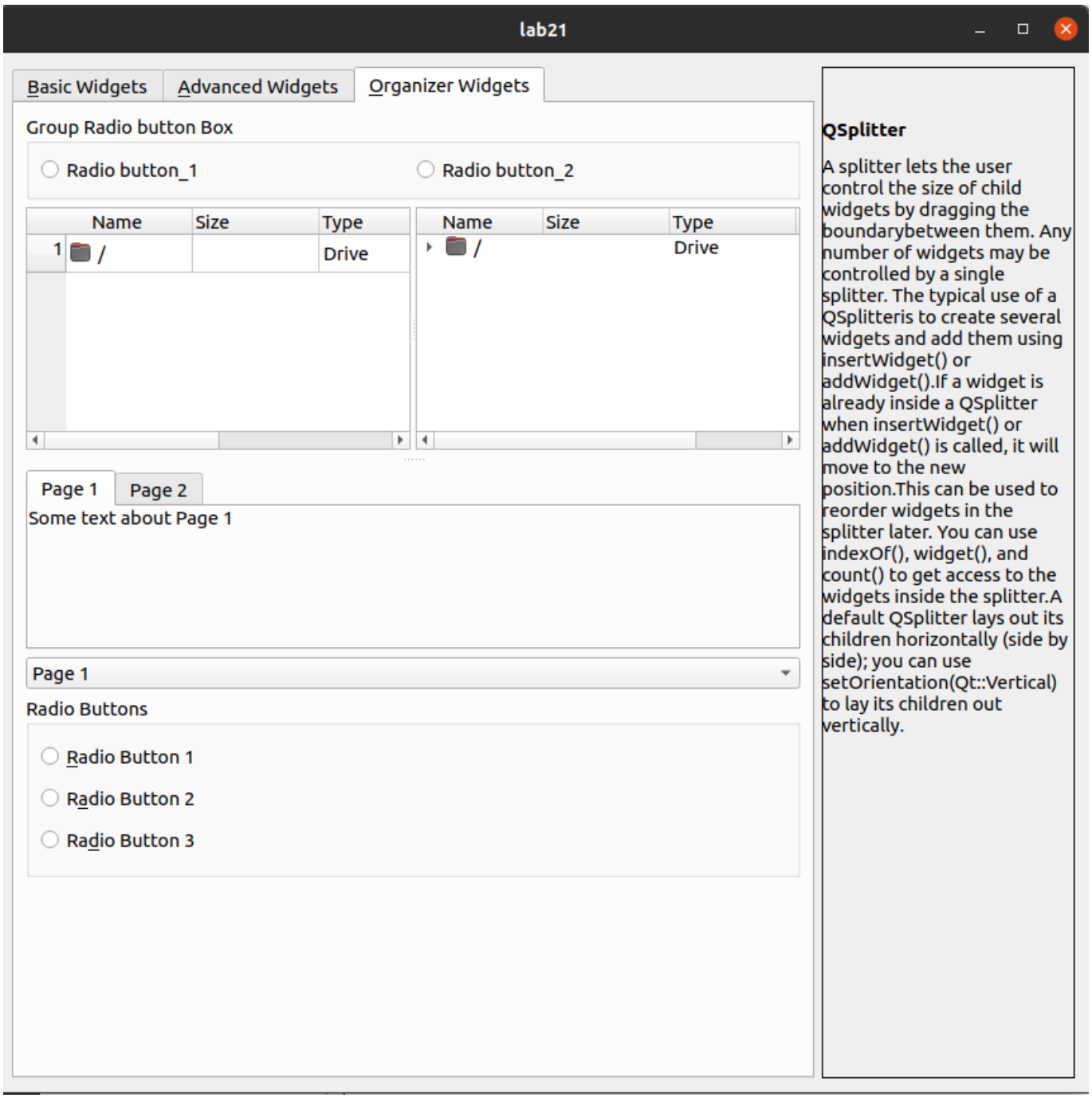
lab21
Basic Widgets
Advanced Widgets
Organizer Widgets
Group Radio button Box
Radio button_1
Radio button_2
Name
Size
Type
1
/
Drive
Name
Size
Type
/
Drive
Page 1
Page 2
Some text about Page 1
Page 1
Radio Buttons
Radio Button 1
Radio Button 2
Radio Button 3
QSplitter
A splitter lets the user control the size of child widgets by dragging the boundarybetween them. Any number of widgets may be controlled by a single splitter. The typical use of a QSplitteris to create several widgets and add them using insertWidget() or addWidget().If a widget is already inside a QSplitter when insertWidget() or addWidget() is called, it will move to the new position.This can be used to reorder widgets in the splitter later. You can use indexOf(), widget(), and count() to get access to the widgets inside the splitter.A default QSplitter lays out its children horizontally (side by side); you can use setOrientation(Qt::Vertical) to lay its children out vertically.

Рисунок 3

Приложение А

А.1 Исходный код main.cpp

```
#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    w.show();
    return a.exec();
}
```

А.2 Исходный код mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>

class QFrame;
class QLabel;

class Widget : public QWidget
{
    Q_OBJECT

    QFrame *frame;
    QLabel *frame_text;
public:
    Widget(QWidget *p = 0);

public slots:
    void changeInfo(QString s);
```

```
};
#endif // MAINWINDOW_H
```

A.3 Исходный код mainwindow.cpp

```
#include "mainwindow.h"
#include "tab1.h"
#include "tab2.h"
#include "tab3.h"

#include <QTabWidget>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QLabel>
#include <QMouseEvent>
#include <QFileSystemModel>

Widget::Widget(QWidget *p) : QWidget(p)
{
    QHBoxLayout *l = new QHBoxLayout(this);
    QTabWidget * tab = new QTabWidget(this);
    tab -> setMinimumSize(600, 800);
    l -> addWidget(tab);

    QWidget *tab1 = new QWidget(tab);
    tab->addTab(tab1, "&Basic Widgets");
    QVBoxLayout * lt1 = new QVBoxLayout(tab1);

    myComboBox * combo = new myComboBox(tab1);
    connect(combo, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
    lt1 -> addWidget(combo);

    myCheckBox * check = new myCheckBox("&MYCheckBox");
    connect(check, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
```

```
lt1 -> addWidget(check);

myCommandLinkButton * command = new myCommandLinkButton(tab1);
connect(command, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(command);

myDateEdit * edit = new myDateEdit(tab1);
connect(edit, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(edit);

myDateTimeEdit * edit_t = new myDateTimeEdit(tab1);
connect(edit_t, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(edit_t);

myDial * dial = new myDial(tab1);
connect(dial, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(dial);

myDoubleSpinBox * spin = new myDoubleSpinBox(tab1);
connect(spin, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(spin);

myFocusFrame * focus_frame = new myFocusFrame(tab1);
connect(focus_frame, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(focus_frame);

myFontComboBox * font_combo = new myFontComboBox(tab1);
connect(font_combo, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(font_combo);

myLCDNumber * number = new myLCDNumber(tab1);
connect(number, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
```

```
lt1 -> addWidget(number);

myLabel * label = new myLabel(tab1);
connect(label, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(label);

myLineEdit * line_edit = new myLineEdit(tab1);
connect(line_edit, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(line_edit);

myMenu * menu = new myMenu(tab1);
connect(menu, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(menu);

myProgressBar * progress = new myProgressBar(tab1);
connect(progress, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(progress);

myPushButton * push_btn = new myPushButton(tab1);
connect(push_btn, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(push_btn);

myRadioButton * radio_btn = new myRadioButton("&Radio button");
connect(radio_btn, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(radio_btn);

myScrollArea * scroll = new myScrollArea(tab1);
connect(scroll, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
QLabel *text1_1 = new QLabel;
text1_1 -> setText("Some text about Page"
                   "Some text about Page"
                   "Some text about Page"
```

```
                "Some text about Page"
                "Some text about Page"
                "Some text about Page"
                "Some text about Page"
                "Some text about Page"
                "Some text about Page"
                "Some text about Page"
                "Some text about Page");
scroll->setWidget(text1_1);
lt1 -> addWidget(scroll);

myScrollBar * scroll_b = new myScrollBar(tab1);
connect(scroll_b, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(scroll_b);

mySizeGrip * size_grip = new mySizeGrip(tab1);
connect(size_grip, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(size_grip);

mySlider * slider = new mySlider(tab1);
connect(slider, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(slider);

mySpinBox * spin_box = new mySpinBox(tab1);
connect(spin_box, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(spin_box);

myTabBar * tab_bar = new myTabBar(tab1);
connect(tab_bar, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(tab_bar);

myTabWidget * tab_widget = new myTabWidget(tab1);
```

```
connect(tab_widget, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
QWidget *widget_on_tab_1_1 = new QWidget;
QWidget *widget_on_tab_2_1 = new QWidget;
QLabel *text_on_page_1_1 = new QLabel(widget_on_tab_1_1);
QLabel *text_on_page_2_1 = new QLabel(widget_on_tab_2_1);
text_on_page_1_1 -> setText("Some text about Page 1");
text_on_page_2_1 -> setText("Some text about Page 2");
tab_widget -> addTab(widget_on_tab_1_1, "Page 1");
tab_widget -> addTab(widget_on_tab_2_1, "Page 2");
lt1 -> addWidget(tab_widget);

myTimeEdit * time_edit = new myTimeEdit(tab1);
connect(time_edit, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(time_edit);

myToolBox * tool_box = new myToolBox(tab1);
connect(tool_box, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(tool_box);

myWidget * widget = new myWidget(tab1);
connect(widget, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt1 -> addWidget(widget);



QWidget *tab2 = new QWidget(tab);
tab->addTab(tab2, "&Advanced Widgets");
QVBoxLayout * lt2 = new QVBoxLayout(tab2);
QFileSystemModel *model = new QFileSystemModel;
model->setRootPath(QDir::rootPath());

myCalendarWidget * calendar_widget = new myCalendarWidget(tab2);
connect(calendar_widget, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
```

```
lt2 -> addWidget(calendar_widget);

myColumnView * col_view = new myColumnView(tab2);
connect(col_view, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
col_view->setModel(model);
lt2 -> addWidget(col_view);

myListView * list_view = new myListView(tab2);
connect(list_view, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
list_view->setModel(model);
lt2 -> addWidget(list_view);

myTableView * table_view = new myTableView(tab2);
myTreeView * tree_view = new myTreeView(tab2);
connect(tree_view, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
connect(table_view, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
table_view->setModel(model);
tree_view->setModel(model);
lt2->addWidget(table_view);
lt2->addWidget(tree_view);

myUndoView * undo_view = new myUndoView(tab2);
connect(undo_view, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
undo_view->setModel(model);
lt2 -> addWidget(undo_view);



QWidget *tab3 = new QWidget(tab);
tab->addTab(tab3, "&Organizer Widgets");
QVBoxLayout * lt3 = new QVBoxLayout(tab3);

myGroupBox * group_box = new myGroupBox("Group Radio button Box");
```

```
connect(group_box, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
myRadioButton * rad_btn_1 = new myRadioButton(tr("Radio button_1"));
myRadioButton * rad_btn_2 = new myRadioButton(tr("Radio button_2"));
QHBoxLayout *lv3 = new QHBoxLayout();
lv3->addWidget(rad_btn_1);
lv3->addWidget(rad_btn_2);
group_box->setLayout(lv3);
lt3 -> addWidget(group_box);

mySplitter * splitter = new mySplitter(tab3);
connect(splitter, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
myTableView * t_v = new  myTableView;
myTreeView * tr_v = new myTreeView;
t_v -> setModel(model);
tr_v -> setModel(model);
splitter -> addWidget(t_v);
splitter -> addWidget(tr_v);
lt3 -> addWidget(splitter);

mySplitter * sp = new mySplitter;
mySplitterHandle * splitter_h = new mySplitterHandle(Qt::Orientation::Vertical, sp);
connect(splitter_h, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt3 -> addWidget(splitter_h);

myTabWidget * tab_widget_2 = new myTabWidget(tab3);
QWidget *widget_on_tab_1 = new QWidget;
QWidget *widget_on_tab_2 = new QWidget;
QLabel *text_on_page_1 = new QLabel(widget_on_tab_1);
QLabel *text_on_page_2 = new QLabel(widget_on_tab_2);
text_on_page_1 -> setText("Some text about Page 1");
text_on_page_2 -> setText("Some text about Page 2");
tab_widget_2 -> addTab(widget_on_tab_1, "Page 1");
```

```
tab_widget_2 -> addTab(widget_on_tab_2, "Page 2");
connect(tab_widget_2, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
lt3 -> addWidget(tab_widget_2);

myGroupBox *groupBox_0 = new myGroupBox("Radio Buttons");
myRadioButton *radioBtn_0 = new myRadioButton("&Radio Button 1");
myRadioButton *radioBtn_1 = new myRadioButton("R&adio Button 2");
myRadioButton *radioBtn_2 = new myRadioButton("Ra&dio Button 3");
QVBoxLayout *vLayout_0 = new QVBoxLayout;
vLayout_0->addWidget(radioBtn_0);
vLayout_0->addWidget(radioBtn_1);
vLayout_0->addWidget(radioBtn_2);
groupBox_0->setLayout(vLayout_0);

myGroupBox *groupBox_1 = new myGroupBox("Checkboxes");
myCheckBox *checkBox_0 = new myCheckBox("&Checkbox 1");
myCheckBox *checkBox_1 = new myCheckBox("C&heckbox 2");
myCheckBox *checkBox_2 = new myCheckBox("Tr&istate button");
checkBox_2->setTristate(true);
QVBoxLayout *vLayout_1 = new QVBoxLayout;
vLayout_1->addWidget(checkBox_0);
vLayout_1->addWidget(checkBox_1);
vLayout_1->addWidget(checkBox_2);
groupBox_1->setLayout(vLayout_1);

myComboBox *pageComboBox = new myComboBox(tab3);
pageComboBox->addItem(tr("Page 1"));
pageComboBox->addItem(tr("Page 2"));

myStackedWidget * stacked_widget = new myStackedWidget(tab3);
myStackedWidget * stacked_widget_2 = new myStackedWidget(tab3);
```

```
    stacked_widget->addWidget(groupBox_0);
    stacked_widget->addWidget(groupBox_1);

    lt3->addWidget(pageComboBox);
    lt3->addWidget(stacked_widget);
    lt3->addWidget(stacked_widget_2);
    connect(stacked_widget_2, SIGNAL(info(QString)), this, SLOT(changeInfo(QString)));
    connect(pageComboBox, SIGNAL(activated(int)), stacked_widget, SLOT(setCurrentIndex(int)));

    frame = new QFrame(this);
    frame -> setFrameShape(QFrame::Box);
    frame -> setMinimumSize(200, 600);
    l -> addWidget(frame);

    frame_text = new QLabel(frame);
    frame_text -> setMinimumSize(200, 600);
    frame_text -> setWordWrap(true);
    frame_text -> setText("<h1>Widget description");
}

void Widget::changeInfo(QString s) {
    frame_text -> setText(s);
}
```

Приложение Б

Б.1 Исходный код tab1.h

```
#ifndef TAB1_H
#define TAB1_H

#include <QWidget>
#include <QComboBox>
#include <QCheckBox>
#include <QCommandLinkButton>
#include <QMouseEvent>
#include <QLabel>
#include <QDateEdit>
#include <QDateTimeEdit>
#include <QDial>
#include <QDoubleSpinBox>
#include <QFocusFrame>
#include <QFontComboBox>
#include <QLCDNumber>
#include <QLineEdit>
#include <QMenu>
#include <QProgressBar>
#include <QPushButton>
#include <QRadioButton>
#include <QScrollArea>
#include <QScrollBar>
#include <QSizeGrip>
#include <QSlider>
#include <QSpinBox>
#include <QTabBar>
#include <QTabWidget>
#include <QTimeEdit>
```

```
#include <QToolBox>
#include <QTextEdit>
#include <QString>


class myComboBox : public QComboBox
{
   Q_OBJECT
public:
   myComboBox(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class myCheckBox : public QCheckBox
{
   Q_OBJECT
public:
   myCheckBox(const QString parent);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class myCommandLinkButton : public QCommandLinkButton
{
   Q_OBJECT
public:
   myCommandLinkButton(QWidget *parent = 0);
```

```
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myDateEdit : public QDateEdit
{
    Q_OBJECT
public:
    myDateEdit(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myDateTimeEdit : public QDateTimeEdit
{
    Q_OBJECT
public:
    myDateTimeEdit(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myDial : public QDial
{
    Q_OBJECT
public:
```

```
    myDial(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myDoubleSpinBox : public QDoubleSpinBox
{
    Q_OBJECT
public:
    myDoubleSpinBox(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myFocusFrame : public QFocusFrame
{
    Q_OBJECT
public:
    myFocusFrame(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myFontComboBox : public QFontComboBox
{
    Q_OBJECT
```

```
public:
    myFontComboBox(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myLCDNumber : public QLCDNumber
{
    Q_OBJECT
public:
    myLCDNumber(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myLabel : public QLabel
{
    Q_OBJECT
public:
    myLabel(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myLineEdit : public QLineEdit
{
```

```
    Q_OBJECT
public:
    myLineEdit(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myMenu : public QMenu
{
    Q_OBJECT
public:
    myMenu(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myProgressBar : public QProgressBar
{
    Q_OBJECT
public:
    myProgressBar(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myPushButton : public QPushButton
```

```
{
    Q_OBJECT
public:
    myPushButton(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myRadioButton : public QRadioButton
{
    Q_OBJECT
public:
    myRadioButton(const QString parent);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

class myScrollArea : public QScrollArea
{
    Q_OBJECT
public:
    myScrollArea(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};
```

```
class myScrollBar : public QScrollBar
{
   Q_OBJECT
public:
   myScrollBar(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class mySizeGrip : public QSizeGrip
{
   Q_OBJECT
public:
   mySizeGrip(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class mySlider : public QSlider
{
   Q_OBJECT
public:
   mySlider(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};
```

```
class mySpinBox : public QSpinBox
{
   Q_OBJECT
public:
   mySpinBox(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class myTabBar : public QTabBar
{
   Q_OBJECT
public:
   myTabBar(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class myTabWidget : public QTabWidget
{
   Q_OBJECT
public:
   myTabWidget(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
```

```
};

class myTimeEdit : public QTimeEdit
{
   Q_OBJECT
public:
   myTimeEdit(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class myToolBox : public QToolBox
{
   Q_OBJECT
public:
   myToolBox(QWidget *parent = 0, Qt::WindowFlags f = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class myWidget : public QWidget
{
   Q_OBJECT
public:
   myWidget(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
```

```
    void info(QString);
};
#endif // TAB1_H
```

Б.2 Исходный код tab1.cpp

```
#include "tab1.h"

myComboBox::myComboBox(QWidget * parent) : QComboBox(parent) {
    setMouseTracking(true);
}

void myComboBox::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QComboBox</b><p>The QComboBox widget is a combined button and popup list."
        "A QComboBox provides a means of presenting a list of options to the user in a way that takes up the minimum amount of screen space."
        "A combobox is a selection widget that displays the current item, and can pop up a list of selectable items. A combobox may be editable,"
        "allowing the user to modify each item in the list."
        "Comboboxes can contain pixmaps as well as strings; the insertItem() and setItemText() functions are suitably overloaded. "
        "For editable comboboxes, the function clearEditText() is provided, to clear the displayed string without changing the combobox's contents.<p>");
    QComboBox::mouseMoveEvent(e);
}

myCheckBox::myCheckBox(const QString parent) : QCheckBox(parent) {
    setMouseTracking(true);
}

void myCheckBox::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QCheckBox</b><p>The QCheckBox widget provides a checkbox with a text label."
        "A QCheckBox is an option button that can be switched on (checked) or off (unchecked). "
        "Checkboxes are typically used to represent features in an application that can be enabled or "
        "disabled without affecting others, but different types of behavior can be implemented. "
```

```
        "For example, a QButtonGroup can be used to group check buttons logically, allowing exclusive checkboxes. "
        "However, QButtonGroup does not provide any visual representation.<p>");
    QCheckBox::mouseMoveEvent(e);
}

myCommandLinkButton::myCommandLinkButton(QWidget * parent) : QCommandLinkButton(parent)
{
    setMouseTracking(true);
}

void myCommandLinkButton::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QCommandLinkButton</b><p>The command link is a new control that was introduced by Windows Vista. "
        "It's intended use is similar to that of a radio button in that it is used to choose between a set of mutually exclusive options."
        "Command link buttons should not be used by themselves but rather as an alternative to radio buttons in Wizards and dialogs "
        "and makes pressing the next button redundant. The appearance is generally similar to that of a flat pushbutton, but it allows "
        "for a descriptive text in addition to the normal button text. By default it will also carry an arrow icon, indicating that pressing "
        "the control will open another window or page.<p>");
    QCommandLinkButton::mouseMoveEvent(e);
}

myDateEdit::myDateEdit(QWidget * parent) : QDateEdit(parent) {
    setMouseTracking(true);
}

void myDateEdit::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QDateEdit</b><p>The QDateEdit class provides a widget for editing dates based on the QDateTimeEdit widget."
        "Many of the properties and functions provided by QDateEdit are implemented in QDateTimeEdit:"
```

```
        "date holds the date displayed by the widget."
        "minimumDate defines the minimum (earliest) date that can be set by the user."
        "maximumDate defines the maximum (latest) date that can be set by the user."
        "displayFormat contains a string that is used to format the date displayed in the widget.<p>");
    QDateEdit::mouseMoveEvent(e);
}

myDateTimeEdit::myDateTimeEdit(QWidget * parent) : QDateTimeEdit(parent) {
    setMouseTracking(true);
}

void myDateTimeEdit::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QDateTimeEdit</b><p>The QDateTimeEdit class provides a widget for editing dates and times."
        "QDateTimeEdit allows the user to edit dates by using the keyboard or the arrow keys to increase and decrease date and time values"
        "The arrow keys can be used to move from section to section within the QDateTimeEdit box. Dates and times appear in accordance with "
        "the format set; see setDisplayFormat().<p>");
    QDateTimeEdit::mouseMoveEvent(e);
}

myDial::myDial(QWidget * parent) : QDial(parent) {
    setMouseTracking(true);
}

void myDial::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QDial</b><p>The QDial class provides a rounded range control (like a speedometer or potentiometer)."
        "QDial is used when the user needs to control a value within a program-definable range, and the range either wraps around (for example, "
        "with angles measured from 0 to 359 degrees) or the dialog layout needs a square widget."
        "Since QDial inherits from QAbstractSlider, the dial behaves in a similar way to a slider. When wrapping() is false (the default setting) "
```

```
        "there is no real difference between a slider and a dial. They both share the same signals, slots and member functions. Which one you use "
        "depends on the expectations of your users and on the type of application.<p>");
    QDial::mouseMoveEvent(e);
}

myDoubleSpinBox::myDoubleSpinBox(QWidget * parent) : QDoubleSpinBox(parent) {
    setMouseTracking(true);
}

void myDoubleSpinBox::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QDoubleSpinBox</b><p>The QDoubleSpinBox class provides a spin box widget that takes doubles."
        "QDoubleSpinBox allows the user to choose a value by clicking the up and down buttons or by pressing Up or Down on "
        "the keyboard to increase or decrease the value currently displayed. The user can also type the value in manually. "
        "The spin box supports double values but can be extended to use different strings with validate(), textFromValue() and valueFromText().<p>");
    QDoubleSpinBox::mouseMoveEvent(e);
}

myFocusFrame::myFocusFrame(QWidget * parent) : QFocusFrame(parent) {
    setMouseTracking(true);
}

void myFocusFrame::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QFocusFrame</b><p>The QFocusFrame widget provides a focus frame which can be outside of a widget's normal paintable area."
        "Normally an application will not need to create its own QFocusFrame as QStyle will handle this detail for you. A style writer can "
        "optionally use a QFocusFrame to have a focus area outside of the widget's paintable geometry."
        "In this way space need not be reserved for the widget to have focus but only set on a QWidget with QFocusFrame::setWidget. "
```

```
        "It is, however, legal to create your own QFocusFrame on a custom widget and set its geometry
manually via QWidget::setGeometry "

        "however you will not get auto-placement when the focused widget changes size or
placement.<p>");

    QFocusFrame::mouseMoveEvent(e);

}


myFontComboBox::myFontComboBox(QWidget * parent) : QFontComboBox(parent) {

    setMouseTracking(true);

}


void myFontComboBox::mouseMoveEvent(QMouseEvent *e) {

    emit info("<b>QFontComboBox</b><p>The QFontComboBox widget is a combobox that lets the user
select a font family."

        "The combobox is populated with an alphabetized list of font family names, such as Arial,
Helvetica, and Times New Roman. "

        "Family names are displayed using the actual font when possible."

        "For fonts such as Symbol, where the name is not representable in the font itself, a sample of the
font is displayed next to the family name.<p>");

    QFontComboBox::mouseMoveEvent(e);

}


myLCDNumber::myLCDNumber(QWidget * parent) : QLCDNumber(parent) {

    setMouseTracking(true);

}


void myLCDNumber::mouseMoveEvent(QMouseEvent *e) {

    emit info("<b>QLCDNumber</b><p>The QLCDNumber widget displays a number with LCD-like
digits."

        "It can display a number in just about any size. It can display decimal, hexadecimal, octal or
binary numbers. "

        "It is easy to connect to data sources using the display() slot, which is overloaded to take any of
five argument types."

        "There are also slots to change the base with setMode() and the decimal point with
setSmallDecimalPoint().<p>");
```

```
    QLCDNumber::mouseMoveEvent(e);
}

myLabel::myLabel(QWidget * parent) : QLabel(parent) {
    setMouseTracking(true);
}

void myLabel::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QLabel</b><p>The QLabel widget provides a text or image display."
            "QLabel is used for displaying text or an image. No user interaction functionality is provided. "
            "The visual appearance of the label can be configured in various ways, and it can be used for specifying a focus mnemonic key for another widget.<p>");
    QLabel::mouseMoveEvent(e);
}

myLineEdit::myLineEdit(QWidget * parent) : QLineEdit(parent) {
    setMouseTracking(true);
}

void myLineEdit::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QLineEdit</b><p>The QLineEdit widget is a one-line text editor."
            "A line edit allows the user to enter and edit a single line of plain text with a useful collection of editing functions, "
            "including undo and redo, cut and paste, and drag and drop."
            "By changing the echoMode() of a line edit, it can also be used as a write-only field, for inputs such as passwords.<p>");
    QLineEdit::mouseMoveEvent(e);
}

myMenu::myMenu(QWidget * parent) : QMenu(parent) {
    setMouseTracking(true);
}
```

```
void myMenu::mouseMoveEvent(QMouseEvent *e) {
   emit info("<b>QMenu</b><p>The QMenu class provides a menu widget for use in menu bars, context menus, and other popup menus."
        "A menu widget is a selection menu. It can be either a pull-down menu in a menu bar or a standalone context menu"
        "Pull-down menus are shown by the menu bar when the user clicks on the respective item or presses the specified shortcut key. "
        "Use QMenuBar::addMenu() to insert a menu into a menu bar."
        "Context menus are usually invoked by some special keyboard key or by right-clicking. "
        "They can be executed either asynchronously with popup() or synchronously with exec(). "
        "Menus can also be invoked in response to button presses; these are just like context menus except for how they are invoked.<p>");
   QMenu::mouseMoveEvent(e);
}

myProgressBar::myProgressBar(QWidget * parent) : QProgressBar(parent) {
   setMouseTracking(true);
}

void myProgressBar::mouseMoveEvent(QMouseEvent *e) {
   emit info("<b>QProgressBar</b><p>The QProgressBar widget provides a horizontal or vertical progress bar."
        "A progress bar is used to give the user an indication of the progress of an operation and to reassure them that the application is still running."
        "The progress bar uses the concept of steps. You set it up by specifying the minimum and maximum possible step values, "
        "and it will display the percentage of steps that have been completed when you later give it the current step value. "
        "The percentage is calculated by dividing the progress (value() - minimum()) divided by maximum() - minimum().<p>");
   QProgressBar::mouseMoveEvent(e);
}

myPushButton::myPushButton(QWidget * parent) : QPushButton(parent) {
   setMouseTracking(true);
```

```
}

void myPushButton::mouseMoveEvent(QMouseEvent *e) {
   emit info("<b>QPushButton</b><p>The QPushButton widget provides a command button."
         "The push button, or command button, is perhaps the most commonly used widget in any graphical user interface."
         "Push (click) a button to command the computer to perform some action, or to answer a question. Typical buttons are OK, "
         "Apply, Cancel, Close, Yes, No and Help.<p>");
   QPushButton::mouseMoveEvent(e);
}

myRadioButton::myRadioButton(const QString parent) : QRadioButton(parent) {
   setMouseTracking(true);
}

void myRadioButton::mouseMoveEvent(QMouseEvent *e) {
   emit info("<b>QRadioButton</b><p>The QRadioButton widget provides a radio button with a text label."
         "A QRadioButton is an option button that can be switched on (checked) or off (unchecked). "
         "Radio buttons typically present the user with a one of many choice. "
         "In a group of radio buttons only one radio button at a time can be checked; if the user selects another button, "
         "the previously selected button is switched off.<p>");
   QRadioButton::mouseMoveEvent(e);
}

myScrollArea::myScrollArea(QWidget * parent) : QScrollArea(parent) {
   setMouseTracking(true);
}

void myScrollArea::mouseMoveEvent(QMouseEvent *e) {
```

```
    emit info("<b>QScrollArea</b><p>The QScrollArea class provides a scrolling view onto another widget."
            "A scroll area is used to display the contents of a child widget within a frame. If the widget exceeds the size of the frame, "
            "the view can provide scroll bars so that the entire area of the child widget can be viewed. The child widget must be specified with setWidget().<p>");
    QScrollArea::mouseMoveEvent(e);
}

myScrollBar::myScrollBar(QWidget * parent) : QScrollBar(parent) {
    setMouseTracking(true);
}

void myScrollBar::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QScrollBar</b><p>The QScrollBar widget provides a vertical or horizontal scroll bar."
            "A scroll bar is a control that enables the user to access parts of a document that is larger than the widget used to display it. "
            "It provides a visual indication of the user's current position within the document and the amount of the document that is visible. "
            "Scroll bars are usually equipped with other controls that enable more accurate navigation. Qt displays scroll bars in a way that "
            "is appropriate for each platform.<p>");
    QScrollBar::mouseMoveEvent(e);
}

mySizeGrip::mySizeGrip(QWidget * parent) : QSizeGrip(parent) {
    setMouseTracking(true);
}

void mySizeGrip::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QSizeGrip</b><p>The QSizeGrip class provides a resize handle for resizing top-level windows."
            "This widget works like the standard Windows resize handle. In the X11 version this resize handle generally works differently "
```

```
        "from the one provided by the system if the X11 window manager does not support necessary modern post-ICCCM specifications.<p>");
    QSizeGrip::mouseMoveEvent(e);
}

mySlider::mySlider(QWidget * parent) : QSlider(parent) {
    setMouseTracking(true);
}

void mySlider::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QSlider</b><p>The QSlider widget provides a vertical or horizontal slider."
        "The slider is the classic widget for controlling a bounded value. It lets the user move a slider handle along a horizontal "
        "or vertical groove and translates the handle's position into an integer value within the legal range.<p>");
    QSlider::mouseMoveEvent(e);
}

mySpinBox::mySpinBox(QWidget * parent) : QSpinBox(parent) {
    setMouseTracking(true);
}

void mySpinBox::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QSpinBox</b><p>The QSpinBox class provides a spin box widget."
        "QSpinBox is designed to handle integers and discrete sets of values (e.g., month names); use QDoubleSpinBox for floating point values."
        "QSpinBox allows the user to choose a value by clicking the up/down buttons or pressing up/down on the keyboard to increase/decrease the value currently displayed."
        "The user can also type the value in manually. The spin box supports integer values but can be extended to use different strings with validate(), textFromValue() and valueFromText().<p>");
    QSpinBox::mouseMoveEvent(e);
}
```

```
myTabBar::myTabBar(QWidget * parent) : QTabBar(parent) {
   setMouseTracking(true);
}

void myTabBar::mouseMoveEvent(QMouseEvent *e) {
   emit info("<b>QTabBar</b><p>The QTabBar class provides a tab bar, e.g. for use in tabbed dialogs."
         "QTabBar is straightforward to use; it draws the tabs using one of the predefined shapes, and emits a signal when a tab is selected."
         "It can be subclassed to tailor the look and feel. Qt also provides a ready-made QTabWidget.<p>");
   QTabBar::mouseMoveEvent(e);
}

myTabWidget::myTabWidget(QWidget * parent) : QTabWidget(parent) {
   setMouseTracking(true);
}

void myTabWidget::mouseMoveEvent(QMouseEvent *e) {
   emit info("<b>QTabWidget</b><p>The QTabWidget class provides a stack of tabbed widgets."
         "A tab widget provides a tab bar (see QTabBar) and a page area that is used to display pages related to each tab. By default, the tab bar is shown above the page area,"
         "but different configurations are available (see TabPosition). Each tab is associated with a different widget (called a page)."
         "Only the current page is shown in the page area; all the other pages are hidden. The user can show a different page by clicking on its tab or by pressing its Alt+letter shortcut if it has one.<p>");
   QTabWidget::mouseMoveEvent(e);
}

myTimeEdit::myTimeEdit(QWidget * parent) : QTimeEdit(parent) {
   setMouseTracking(true);
}

void myTimeEdit::mouseMoveEvent(QMouseEvent *e) {
```

```
    emit info("<b>QTimeEdit</b><p>The QTimeEdit class provides a widget for editing times based on the QDateTimeEdit widget."
        "Many of the properties and functions provided by QTimeEdit are implemented in QDateTimeEdit. <p>");
    QTimeEdit::mouseMoveEvent(e);
}

myToolBox::myToolBox(QWidget * parent, Qt::WindowFlags f) : QToolBox(parent, f) {
    setMouseTracking(true);
}

void myToolBox::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QToolBox</b><p>The QToolBox class provides a column of tabbed widget items."
        "A toolbox is a widget that displays a column of tabs one above the other, with the current item displayed below the current tab."
        "Every tab has an index position within the column of tabs. A tab's item is a QWidget.<p>");
    QToolBox::mouseMoveEvent(e);
}

myWidget::myWidget(QWidget * parent) : QWidget(parent) {
    setMouseTracking(true);
}

void myWidget::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QWidget</b><p>The QWidget class is the base class of all user interface objects."
        "The widget is the atom of the user interface: it receives mouse, keyboard and other events from the window system,"
        "and paints a representation of itself on the screen. Every widget is rectangular, and they are sorted in a Z-order."
        "A widget is clipped by its parent and by the widgets in front of it.<p>");
    QWidget::mouseMoveEvent(e);
}
```

Б.3 Исходный код tab2.h

```
#ifndef TAB2_H
```

```
#define TAB2_H

#include <QWidget>
#include <QMouseEvent>
#include <QCalendarWidget>
#include <QColumnView>
#include <QListView>
#include <QTableView>
#include <QTreeView>
#include <QUndoView>


class myCalendarWidget : public QCalendarWidget
{
   Q_OBJECT
public:
   myCalendarWidget(QWidget * parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class myColumnView : public QColumnView
{
   Q_OBJECT
public:
   myColumnView(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
```

```
};

class myListView : public QListView
{
   Q_OBJECT
public:
   myListView(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class myTableView : public QTableView
{
   Q_OBJECT
public:
   myTableView(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class myTreeView : public QTreeView
{
   Q_OBJECT
public:
   myTreeView(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
```

```
    void info(QString);
};

class myUndoView : public QUndoView
{
    Q_OBJECT
public:
    myUndoView(QWidget *parent = 0);
protected:
    void mouseMoveEvent(QMouseEvent *e);
signals:
    void info(QString);
};

#endif // TAB2_H
```

Б.4 Исходный код tab2.cpp

```
#include "tab2.h"

myCalendarWidget::myCalendarWidget(QWidget * parent) : QCalendarWidget(parent) {
    setMouseTracking(true);
}

void myCalendarWidget::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QCalendarWidget</b><p>The widget is initialized with the current month and year, but
"
        "QCalendarWidget provides several public slots to change the year and month that is shown."
        "By default, today's date is selected, and the user can select a date using both mouse and
keyboard."
        "The currently selected date can be retrieved using the selectedDate() function. It is possible to
constrain "
        "the user selection to a given date range by setting the minimumDate and maximumDate
properties. Alternatively, "
        "both properties can be set in one go using the setDateRange() convenience slot. Set the
selectionMode property "
```

```
        "to NoSelection to prohibit the user from selecting at all. Note that a date also can be selected
programmatically using the setSelectedDate() slot.<p>");
    QCalendarWidget::mouseMoveEvent(e);
}

myColumnView::myColumnView(QWidget * parent) : QColumnView(parent) {
    setMouseTracking(true);
}

void myColumnView::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QColumnView</b><p>QColumnView displays a model in a number of QListViews, "
        "one for each hierarchy in the tree. This is sometimes referred to as a cascading list."
        "The QColumnView class is one of the Model/View Classes and is part of Qt's model/view
framework."
        "QColumnView implements the interfaces defined by the QAbstractItemView class to allow it to
display "
        "data provided by models derived from the QAbstractItemModel class.<p>");
    QColumnView::mouseMoveEvent(e);
}

myListView::myListView(QWidget * parent) : QListView(parent) {
    setMouseTracking(true);
}

void myListView::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QListView</b><p>A QListView presents items stored in a model, either as a simple
non-hierarchical list, "
        "or as a collection of icons. This class is used to provide lists and icon views that were previously
provided "
        "by the QListBox and QIconView classes, but using the more flexible approach provided by Qt's
model/view architecture."
        "The QListView class is one of the Model/View Classes and is part of Qt's model/view
framework."
```

```
        "This view does not display horizontal or vertical headers; to display a list of items with a
horizontal header, use QTreeView instead."

        "QListView implements the interfaces defined by the QAbstractItemView class to allow it to
display data provided by models derived "

        "from the QAbstractItemModel class.<p>");

   QListView::mouseMoveEvent(e);

}


myTableView::myTableView(QWidget * parent) : QTableView(parent) {

   setMouseTracking(true);

}


void myTableView::mouseMoveEvent(QMouseEvent *e) {

   emit info("<b>QTableView</b><p>A QTableView implements a table view that displays items from a
model."

        "This class is used to provide standard tables that were previously provided by the QTable class,
but using the "

        "more flexible approach provided by Qt's model/view architecture."

        "The QTableView class is one of the Model/View Classes and is part of Qt's model/view
framework."

        "QTableView implements the interfaces defined by the QAbstractItemView class to allow it to
display "

        "data provided by models derived from the QAbstractItemModel class.<p>");

   QTableView::mouseMoveEvent(e);

}


myTreeView::myTreeView(QWidget * parent) : QTreeView(parent) {

   setMouseTracking(true);

}


void myTreeView::mouseMoveEvent(QMouseEvent *e) {

   emit info("<b>QTreeView</b><p>A QTreeView implements a tree representation of items from a
model. "

        "This class is used to provide standard hierarchical lists that were previously provided"
```

```
        " by the QListView class, but using the more flexible approach provided by Qt's model/view architecture."
        "The QTreeView class is one of the Model/View Classes and is part of Qt's model/view framework."
        "QTreeView implements the interfaces defined by the QAbstractItemView class to allow it to display "
        "data provided by models derived from the QAbstractItemModel class.<p>");
    QTreeView::mouseMoveEvent(e);
}

myUndoView::myUndoView(QWidget * parent) : QUndoView(parent) {
    setMouseTracking(true);
}

void myUndoView::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QUndoView</b><p>QUndoView is a QListView which displays the list of commands pushed on "
        "an undo stack. The most recently executed command is always selected. Selecting a different command "
        "results in a call to QUndoStack::setIndex(), rolling the state of the document backwards or forward to the new command."
        "The stack can be set explicitly with setStack(). Alternatively, a QUndoGroup object can be set with setGroup()."
        "The view will then update itself automatically whenever the active stack of the group changes.<p>");
    QUndoView::mouseMoveEvent(e);
}
```

Б.5 Исходный код tab3.h

```
#ifndef TAB3_H
#define TAB3_H

#include <QWidget>
#include <QButtonGroup>
#include <QMouseEvent>
#include <QGroupBox>
```

```
#include <QSplitter>
#include <QSplitterHandle>
#include <QStackedWidget>
#include <QTabWidget>

class myGroupBox : public QGroupBox
{
   Q_OBJECT
public:
   myGroupBox(const QString parent);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class mySplitter : public QSplitter
{
   Q_OBJECT
public:
   mySplitter(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class mySplitterHandle : public QSplitterHandle
{
   Q_OBJECT
public:
   mySplitterHandle(Qt::Orientation, QSplitter *);
```

```
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};

class myStackedWidget : public QStackedWidget
{
   Q_OBJECT
public:
   myStackedWidget(QWidget *parent = 0);
protected:
   void mouseMoveEvent(QMouseEvent *e);
signals:
   void info(QString);
};
#endif // TAB3_H
```

Б.6 Исходный код tab3.cpp

```
#include "tab3.h"

myGroupBox::myGroupBox(const QString parent) : QGroupBox(parent) {
   setMouseTracking(true);
}

void myGroupBox::mouseMoveEvent(QMouseEvent *e) {
   emit info("<b>QGroupBox</b><p>A group box provides a frame, a title on top, a keyboard shortcut, and displays various other widgets inside itself."
         "The keyboard shortcut moves keyboard focus to one of the group box's child widgets."
         "QGroupBox also lets you set the title (normally set in the constructor) and the title's alignment.
"
         "Group boxes can be checkable. Child widgets in checkable group boxes are enabled or disabled depending on whether or not the group box is checked.<p>");
   QGroupBox::mouseMoveEvent(e);
```

```
}

mySplitter::mySplitter(QWidget * parent) : QSplitter(parent) {
   setMouseTracking(true);
}

void mySplitter::mouseMoveEvent(QMouseEvent *e) {
   emit info("<b>QSplitter</b><p>A splitter lets the user control the size of child widgets by dragging the boundary"
        "between them. Any number of widgets may be controlled by a single splitter. The typical use of a QSplitter"
        "is to create several widgets and add them using insertWidget() or addWidget()."
        "If a widget is already inside a QSplitter when insertWidget() or addWidget() is called, it will move to the new position."
        "This can be used to reorder widgets in the splitter later. You can use indexOf(), widget(), and count() to get access to the widgets inside the splitter."
        "A default QSplitter lays out its children horizontally (side by side); you can use setOrientation(Qt::Vertical) to lay its children out vertically.<p>");
   QSplitter::mouseMoveEvent(e);
}

mySplitterHandle::mySplitterHandle(Qt::Orientation f, QSplitter * e) : QSplitterHandle(f, e) {
   setMouseTracking(true);
}

void mySplitterHandle::mouseMoveEvent(QMouseEvent *e) {
   emit info("<b>QSplitterHandle</b><p>QSplitterHandle is typically what people think about when they think about a splitter."
        "It is the handle that is used to resize the widgets."
        "A typical developer using QSplitter will never have to worry about QSplitterHandle. "
        "It is provided for developers who want splitter handles that provide extra features, such as popup menus."
        "The typical way one would create splitter handles is to subclass QSplitter and then reimplement QSplitter::createHandle()"
```

```
        "to instantiate the custom splitter handle. <p>");
    QSplitterHandle::mouseMoveEvent(e);
}

myStackedWidget::myStackedWidget(QWidget * parent) : QStackedWidget(parent) {
    setMouseTracking(true);
}

void myStackedWidget::mouseMoveEvent(QMouseEvent *e) {
    emit info("<b>QStackedWidget</b><p>QStackedWidget can be used to create a user interface similar to the one provided by QTabWidget. "
        "It is a convenience layout widget built on top of the QStackedLayout class."
        "Like QStackedLayout, QStackedWidget can be constructed and populated with a number of child widgets"
        "QStackedWidget provides no intrinsic means for the user to switch page. This is typically done through a "
        "QComboBox or a QListWidget that stores the titles of the QStackedWidget's pages. <p>");
    QStackedWidget::mouseMoveEvent(e);
}
```

Приложение B. UML диаграмма