

**Правительство Российской Федерации**

---

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

**Кафедра «Компьютерная безопасность»**

**Отчёт к лабораторной работе №1 (2 год)**  
**по дисциплине**  
**«Языки программирования»**

Работу выполнил

студент группы СКБ221

М. И. Нугманов

---

подпись, дата

Работу проверил

С.А.Булгаков

---

подпись, дата

Москва 2023 г.

# Содержание

Содержание .....	2
Постановка задачи .....	4
1. Алгоритм решения поставленной задачи.....	4
2. Реализация.....	4
2.1. Вкладки органайзера.....	5
2.1.1. myQCheckBox .....	5
2.1.2. myQComboBox.....	5
2.1.3. myQCommandLinkButton.....	7
2.1.4. myQDateEdit.....	7
2.1.5. myQDateTimeEdit .....	7
2.1.6. myQDial.....	8
2.1.7. myQDoubleSpinBox .....	8
2.1.8. myQFocusFrame .....	8
2.1.9. myQFontComboBox .....	9
2.1.10. myQLCDNumber.....	9
2.1.11. myQLabel .....	10
2.1.12. myQLineEdit.....	10
2.1.13. myQMenu .....	10
2.1.14. myQProgressBar .....	11
2.1.15. myQPushButton .....	11
2.1.16. myQRadioButton .....	12
2.1.17. myQScrollArea .....	12
2.1.18. myQScrollBar .....	12
2.1.19. myQSizeGrip .....	13
2.1.20. myQSlider .....	13
2.1.21. myQSpinBox.....	13
2.1.22. myQTabBar.....	14

2.1.23.	myQTabWidget.....	14
2.1.24.	myQTimeEdit .....	14
2.1.25.	myQToolBox .....	15
2.1.26.	myQToolButton .....	15
2.1.27.	myQWidget.....	15
2.1.28.	myQCalendarWidget.....	16
2.1.29.	myQListView.....	16
2.1.30.	myQTableView .....	16
2.1.31.	myQTreeView.....	17
2.1.32.	myQUndoView .....	17
2.1.33.	myQGroupBox.....	17
2.1.34.	myQStackedWidget .....	18
2.2.	Вкладки органайзера.....	19
2.2.1.	Basic .....	19
2.2.2.	Advanced .....	19
2.2.3.	Organizer .....	19
3.	Тестирование программы.....	20
	Приложение А.....	22
	Приложение Б .....	23
	Приложение В.....	24
	Приложение Г .....	25

## Постановка задачи

Разработать программу с использованием библиотеки *Q*, позволяющую ознакомиться с имеющимися элементами пользовательского интерфейса. Окно программы должно содержать 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 поставленной задачи

Для решения задачи создан следующий набор классов: класс, реализующий панель с текстом; класс, реализованный как вкладка органайзера; каждый элемент пользовательского интерфейса имеет свой класс-наследник. Взаимодействие между классами осуществлено с помощью механизма сигнала и слотов.

### 2. Реализация

В подклассах графических элементов разработан открытый конструктор общего вида, выполнена перегрузка метода *mousePressEvent*, а также реализован сигнал с параметром типа *QString*. Все подклассы создавались по единому принципу, единственные отличия заключались в классе прародителя.

Общий вид UML-диаграммы подклассов приведен в *Приложении А*, общий вид объявления подклассов приведен в *Приложении Б*, а общий вид реализации подклассов приведен в *Приложении В*.

Классы, которые отвечают за вкладки органайзера и панель с текстом, размещены на основном слое *QHBoxLayout*.

Класс, который отвечает за вкладки органайзера, создан на основе класса *QWidget*, на котором с помощью *QVBoxLayout* размещены элементы пользовательского интерфейса.

Элементы, отвечающие за панель с текстом, реализованы в виде композиции виджетов *QFrame* и *QLabel*.

## **2.1. Вкладки органайзера**

### **2.1.1. myQCheckBox**

*QCheckBox* - это виджет флажка, который может быть установлен или снят пользователем для обозначения наличия или отсутствия определенной характеристики.

Используется в формах и диалоговых окнах для получения согласия пользователя или выбора опций.

Основные методы:

*setChecked(bool checked):* Устанавливает состояние флажка.

*isChecked():* Проверяет, установлен ли флажок.

*setText(const QString& text):* Устанавливает текст флажка.

### **2.1.2. myQComboBox**

*QComboBox* - это выпадающий список, который позволяет пользователю выбирать один из нескольких вариантов.

Используется для выбора элемента из списка доступных опций.

Основные методы:

*addItem(const QString& text):* Добавляет элемент в выпадающий список.

*currentText():* Возвращает текст текущего выбранного элемента.

*setCurrentIndex(int index):* Устанавливает текущий индекс выбранного элемента.

### 2.1.3. myQCommandLinkButton

*QCommandLinkButton* - это кнопка, предназначенная для выполнения команды или действия. Он обычно содержит большой текст и информативный заголовок.

Используется для создания информативных кнопок, которые привлекают внимание пользователя.

Основные методы:

*setText(const QString& text):* Устанавливает текст кнопки.

*setDescription(const QString& description):* Устанавливает описание кнопки.

### 2.1.4. myQDateEdit

*QDateEdit* - это поле ввода для даты, которое позволяет пользователю выбирать или вводить дату.

Используется для ввода дат в формах и диалогах, где требуется указать дату.

Основные методы:

*setDate(const QDate& date):* Устанавливает выбранную дату.

*date():* Возвращает выбранную дату.

### 2.1.5. myQDateTimeEdit

*QDateTimeEdit* - это поле ввода для даты и времени, которое позволяет пользователю выбирать или вводить дату и время.

Используется для ввода даты и времени в формах и диалогах.

Основные методы:

*setDateTime(const QDateTime& dateTime):* Устанавливает выбранную дату и время.

*dateTime():* Возвращает выбранную дату и время.

### 2.1.6. **myQDial**

*QDial* - это крутильный регулятор, который позволяет пользователю выбирать числовое значение, вращая его.

Используется для выбора числовых значений с помощью вращения регулятора.

Основные методы:

*setValue(int value)*: Устанавливает текущее значение на диске.

*valueChanged(int value)*: Сигнал, срабатывающий при изменении значения.

### 2.1.7. **myQDoubleSpinBox**

*QDoubleSpinBox* - это поле ввода для чисел с плавающей точкой.

Используется для ввода дробных чисел в формах и диалогах.

Основные методы:

*setValue(double value)*: Устанавливает текущее значение.

*value()*: Возвращает текущее значение.

*setDecimals(int decimals)*: Устанавливает количество десятичных знаков.

### 2.1.8. **myQFocusFrame**

*QFocusFrame* - это виджет, который представляет фокусную рамку и используется для отображения активного элемента управления в пользовательском интерфейсе.

Используется для визуализации фокуса в пользовательском интерфейсе.

Основные методы:

*setWidget(QWidget\* widget)*: Устанавливает виджет для отображения фокусной рамки.

*setStyleOption(QStyleOption\* option)*: Устанавливает опции стиля.



### 2.1.9. myQFontComboBox

*QFontComboBox* - это выпадающий список для выбора шрифта из доступного списка.

Используется для выбора шрифта в текстовых редакторах и других приложениях, где требуется установка шрифта.

Основные методы:

*setCurrentFont(const QFont& font)*: Устанавливает текущий шрифт.

*currentFont()*: Возвращает текущий выбранный шрифт.

### 2.1.10. myQLCDNumber

*QLCDNumber* - это виджет для отображения числовых значений в стиле жидкокристаллического дисплея.

Используется для отображения числовых показателей, таких как время, температура, и другие значения.

Основные методы:

*display(int num)*: Отображает числовое значение.

*intValue()*: Возвращает текущее числовое значение.

*setDigitCount(int count)*: Устанавливает количество цифр для отображения.

### 2.1.11. **myQLabel**

*QLabel* - это виджет для отображения текста или изображений.

Используется для отображения информации или изображений в пользовательском интерфейсе.

Основные методы:

*setText(const QString& text):* Устанавливает текст метки.

*setPixmap(const QPixmap& pixmap):* Устанавливает изображение метки.

### 2.1.12. **myQLineEdit**

*QLineEdit* - это поле для ввода текста, которое позволяет пользователю вводить текст.

Используется для получения текстового ввода от пользователя, например, для ввода имени, пароля или других текстовых данных.

Основные методы:

*setText(const QString& text):* Устанавливает текст в поле ввода.

*text():* Возвращает текст из поля ввода.

*setPlaceholderText(const QString& text):* Устанавливает текст-подсказку.

### 2.1.13. **myQMenu**

*QMenu* - это контекстное меню, которое предоставляет список доступных действий или команд для выбора.

Используется для создания выпадающих меню и контекстных меню в приложениях.

Основные методы:

*addAction(QAction\* action):* Добавляет действие в меню.

*addMenu(QMenu\* menu):* Добавляет подменю.

*popup(const QPoint& pos):* Показывает меню в указанной позиции.

### 2.1.14. **myQProgressBar**

*QProgressBar* - это виджет, который отображает ход выполнения задачи или процесса.

Используется для отслеживания прогресса выполнения операции.

Основные методы:

*setValue(int value)*: Устанавливает текущее значение полосы состояния.

*setRange(int minimum, int maximum)*: Устанавливает диапазон значений полосы состояния.

### 2.1.15. **myQPushButton**

*QPushButton* - это кнопка, которая позволяет пользователю выполнить команду или действие.

Используется для создания кнопок, которые выполняют операции при нажатии.

Основные методы:

*setText(const QString& text)*: Устанавливает текст кнопки.

*clicked()*: Сигнал, срабатывающий при нажатии на кнопку.

### 2.1.16. myQRadioButton

*QRadioButton* - это радиокнопка, которая позволяет пользователю выбрать одну из нескольких взаимоисключающих опций.

Используется для создания групп опций, из которых можно выбрать только одну.

Основные методы:

*setChecked(bool checked)*: Устанавливает состояние радиокнопки.

*isChecked()*: Проверяет, установлена ли радиокнопка.

### 2.1.17. myQScrollArea

*QScrollArea* - это область прокрутки, которая позволяет отображать содержимое, которое не помещается на экране.

Используется для отображения больших или многостраничных контентов, таких как изображения или текст.

Основные методы:

*addWidget(QWidget\* widget)*: Устанавливает виджет для отображения в области прокрутки.

*addWidgetResizable(bool resizable)*: Устанавливает возможность изменения размеров виджета.

### 2.1.18. myQScrollBar

*QScrollBar* - это полоса прокрутки, которая позволяет пользователю изменять значение в пределах заданного диапазона.

Применение: Используется в виджетах с прокруткой, таких как *QScrollArea*, для прокрутки контента.

Основные методы:

*setValue(int value)*: Устанавливает текущее значение полосы прокрутки.

*valueChanged(int value)*: Сигнал, срабатывающий при изменении значения.

### 2.1.19. **myQSizeGrip**

*QSizeGrip* - это виджет, который позволяет пользователю изменять размеры окна, улавливая его угол.

Используется для создания возможности изменения размеров окна приложения.

Основные методы:

*setVisible(bool visible)*: Устанавливает видимость уголка изменения размера.

### 2.1.20. **myQSlider**

*QSlider* - это ползунок, который позволяет пользователю выбирать значение в пределах заданного диапазона.

Используется для выбора числовых значений или настройки параметров.

Основные методы:

*setValue(int value)*: Устанавливает текущее значение ползунка.

*valueChanged(int value)*: Сигнал, срабатывающий при изменении значения.

### 2.1.21. **myQSpinBox**

*QSpinBox* - это поле ввода для целых чисел.

Используется для ввода целых чисел в формах и диалогах.

Основные методы:

*setValue(int value)*: Устанавливает текущее значение.

*value()*: Возвращает текущее значение.

*setRange(int minimum, int maximum)*: Устанавливает диапазон значений.

### 2.1.22. myQTabBar

*QTabBar* - это виджет для организации множества вкладок, которые могут быть переключены пользователем.

Используется для создания интерфейсов с вкладками, где каждая вкладка представляет разные разделы или функциональности.

Основные методы:

*addTab(const QString& text)*: Добавляет вкладку с указанным текстом.

*setCurrentIndex(int index)*: Устанавливает текущую активную вкладку.

### 2.1.23. myQTabWidget

*QTabWidget* - это контейнер, который содержит набор вкладок, и каждая вкладка может содержать свое собственное содержимое.

Используется для создания интерфейсов с несколькими вкладками, где каждая вкладка представляет отдельный раздел приложения.

Основные методы:

*addTab(QWidget\* widget, const QString& label)*: Добавляет вкладку с содержимым и текстом.

*setCurrentIndex(int index)*: Устанавливает текущую активную вкладку.

### 2.1.24. myQTimeEdit

*QTimeEdit* - это поле ввода для времени, которое позволяет пользователю выбирать или вводить время.

Используется для ввода времени в формах и диалогах, где требуется указать время.

Основные методы:

*setTime(const QTime& time)*: Устанавливает выбранное время.

*time()*: Возвращает выбранное время.

### 2.1.25. myQToolBox

*QToolBox* - это контейнер, который содержит набор элементов с заголовками и позволяет пользователю переключаться между ними.

Используется для создания интерфейсов с несколькими разделами, где каждый раздел может содержать различное содержимое.

Основные методы:

*addItem(QWidget\* widget, const QString& text)*: Добавляет элемент с содержимым и текстом.

*setCurrentIndex(int index)*: Устанавливает текущий активный элемент.

### 2.1.26. myQPushButton

*QPushButton* - это кнопка, которая используется для выполнения команд или действий, а также может отображать значки.

Используется для создания кнопок с иконками или текстом для выполнения операций.

Основные методы:

*setText(const QString& text)*: Устанавливает текст кнопки.

*clicked()*: Сигнал, срабатывающий при нажатии на кнопку.

### 2.1.27. myQWidget

*QWidget* - это базовый класс для всех виджетов в Qt, который представляет собой окно или элемент пользовательского интерфейса.

Используется для создания пользовательских виджетов и приложений.

### 2.1.28. myQCalendarWidget

*QCalendarWidget* - это виджет календаря, который позволяет пользователю выбирать даты.

Применение: Используется для выбора даты в приложениях, где требуется работа с календарем.

Основные методы:

*setSelectedDate(const QDate & date)*: Устанавливает выбранную дату.

*selectedDate()*: Возвращает выбранную дату.

### 2.1.29. myQListView

*QListView* - это виджет, который отображает данные из модели в виде списка элементов.

Используется для отображения данных в списке, где каждый элемент представляет собой одну запись.

Основные методы:

*setModel(QAbstractItemModel\* model)*: Устанавливает модель данных.

*setViewMode(QListView::ViewMode mode)*: Устанавливает режим отображения.

### 2.1.30. myQTableView

*QTableView* - это виджет, который отображает данные из модели в виде таблицы.

Используется для отображения данных в виде таблицы, где каждая строка представляет собой запись, а столбцы - поля данных.

Основные методы:

*setModel(QAbstractItemModel\* model)*: Устанавливает модель данных.

*setSortingEnabled(bool enable)*: Включает или отключает сортировку данных.



### 2.1.31. myQTreeView

*QTreeView* - это виджет, который отображает иерархические данные из модели.

Используется для отображения иерархических данных, таких как файловая система или иерархия каталогов.

Основные методы:

*setModel(QAbstractItemModel\* model)*: Устанавливает модель данных.

*expandAll()*: Разворачивает все узлы в дереве.

### 2.1.32. myQUndoView

*QUndoView* - это виджет, который отображает и управляет списком действий, доступных для отмены и повтора.

Используется для отображения истории действий в приложениях с поддержкой отмены и повтора.

Основные методы:

*setStack(QUndoStack\* stack)*: Устанавливает стек отмены/повтора.

*setCleanIcon(const QIcon& icon)*: Устанавливает значок для "чистого" состояния.

### 2.1.33. myQGroupBox

*QGroupBox* - это контейнер, который используется для создания группировки виджетов с заголовком.

Используется для создания группировки и организации виджетов в логические разделы.

Основные методы:

*setTitle(const QString& title)*: Устанавливает заголовок группы.

### 2.1.34. **myQStackedWidget**

*QStackedWidget* - это контейнер, который отображает только один из своих дочерних виджетов одновременно, переключаясь между ними.

Используется для создания интерфейсов с переключаемыми разделами, где каждый раздел представляет собой отдельный виджет.

Основные методы:

*addWidget(QWidget\* widget)*: Добавляет виджет в контейнер.

*setCurrentIndex(int index)*: Устанавливает текущий активный виджет.

## **2.2. Вкладки органайзера**

Вкладки органайзера предназначены для распределения элементов интерфейса по различным типам. Всего выделяется 3 типа виджетов: Basic, Advanced и Organizer.

### **2.2.1. Basic**

Это основные элементы пользовательского интерфейса. Они представляют собой простые компоненты, такие как кнопки, текстовые поля, метки и другие, которые можно использовать для создания базовых элементов интерфейса.

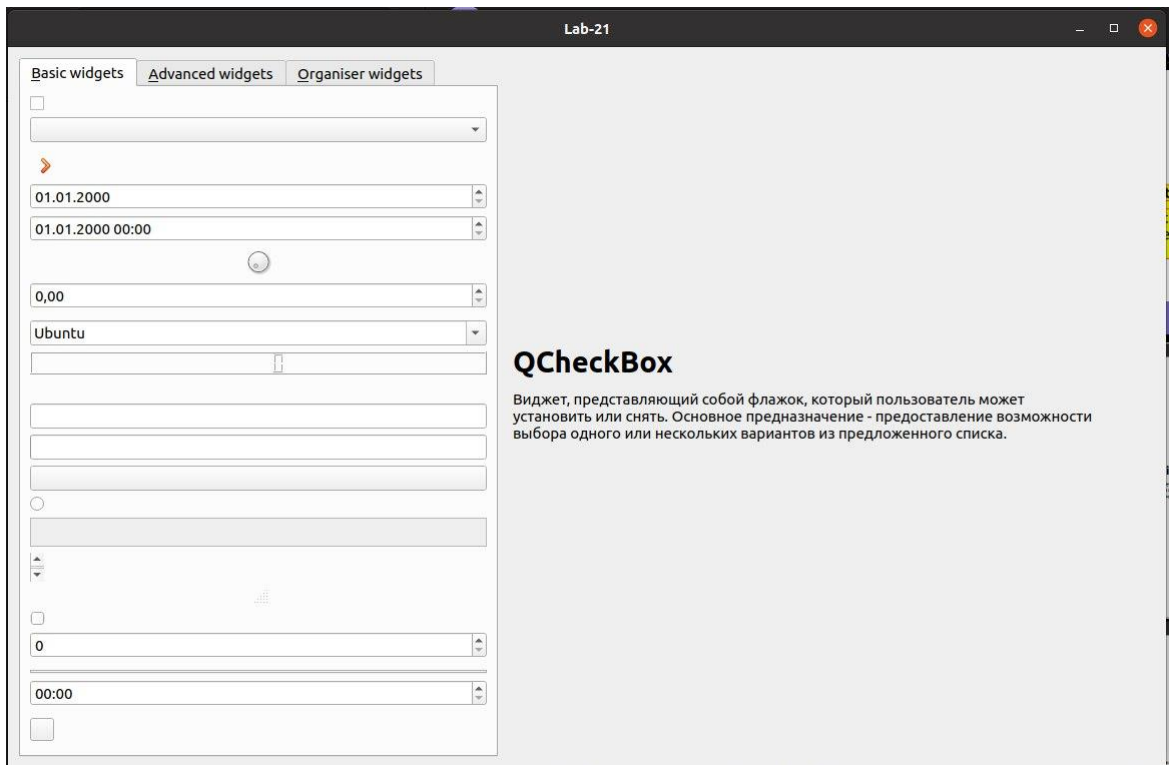
### **2.2.2. Advanced**

Это более сложные и функциональные элементы пользовательского интерфейса. Они предоставляют расширенные возможности и часто используются для создания более сложных приложений с богатым функционалом.

### **2.2.3. Organizer**

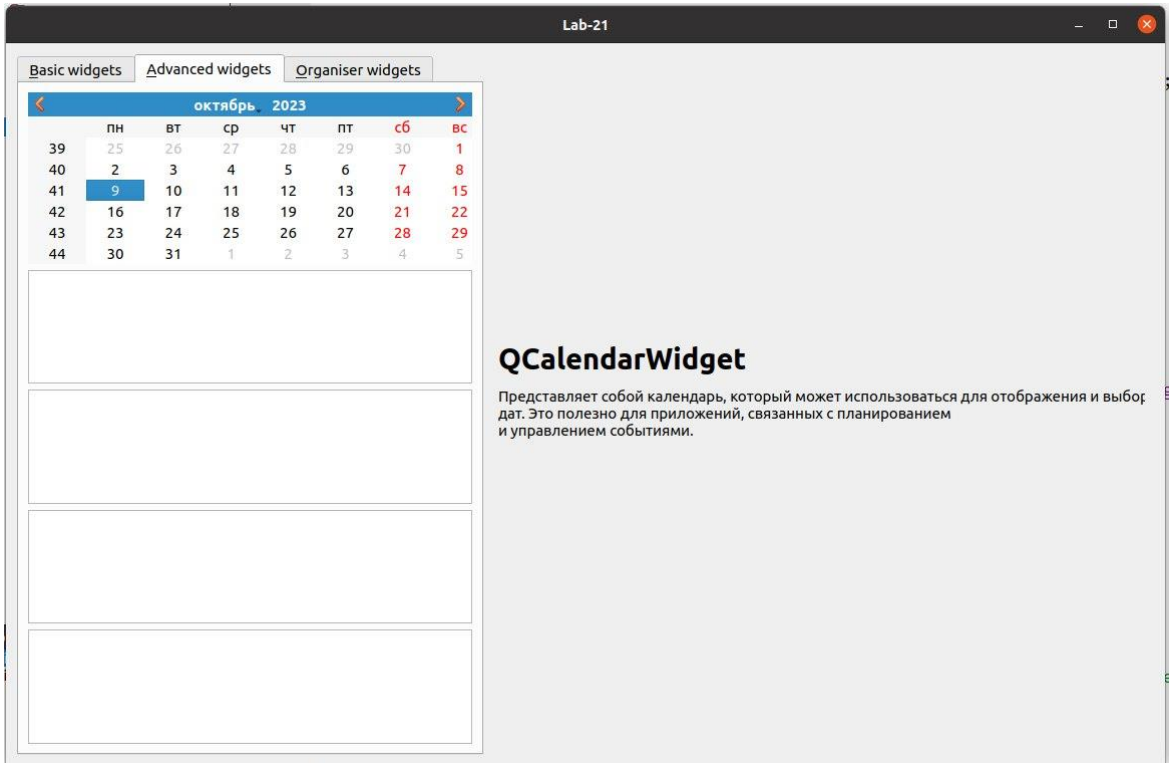
Это виджеты-контейнеры, предназначенные для управления расположением и организацией других виджетов в пользовательском интерфейсе. Они используются для создания сложных макетов и структуры приложения, облегчая размещение и выравнивание виджетов.

### 3. Тестирование программы



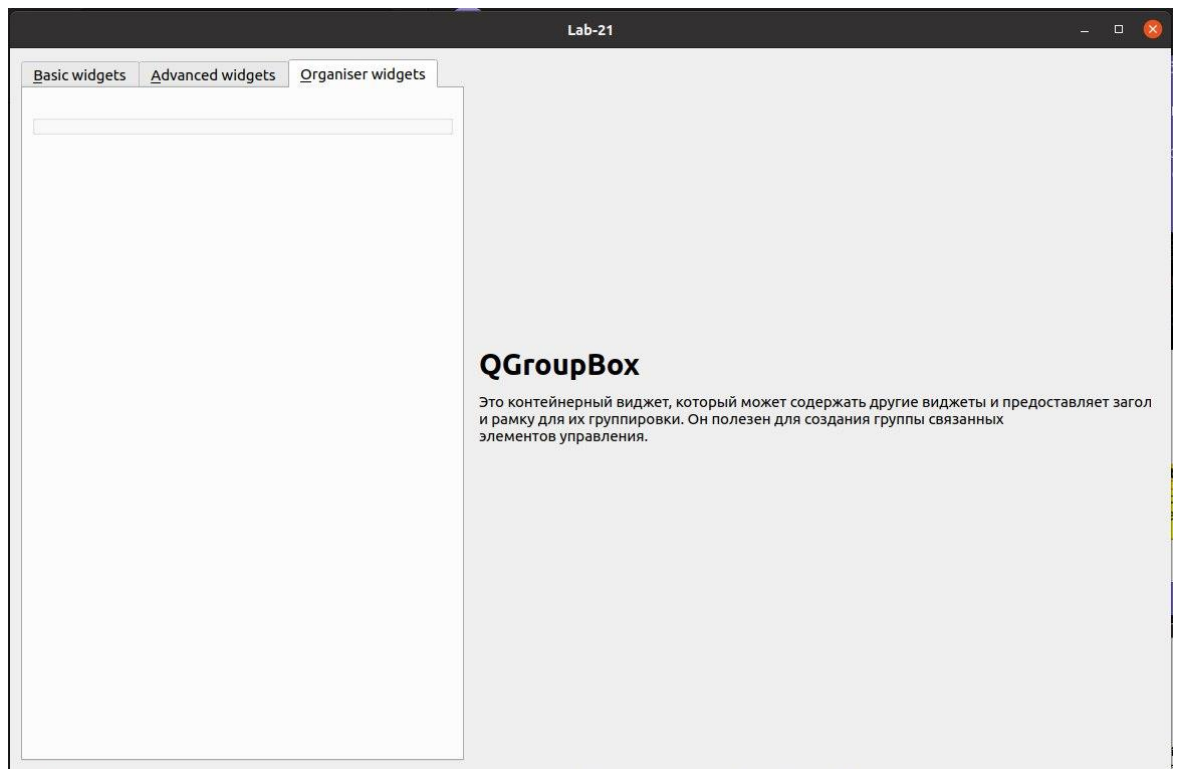
**QCheckBox**

Виджет, представляющий собой флажок, который пользователь может установить или снять. Основное предназначение - предоставление возможности выбора одного или нескольких вариантов из предложенного списка.



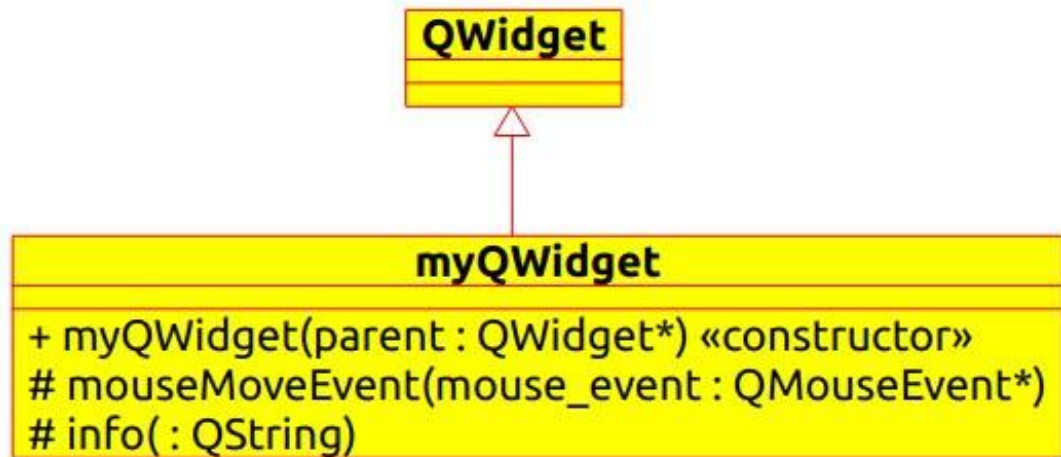
**QCalendarWidget**

Представляет собой календарь, который может использоваться для отображения и выбора дат. Это полезно для приложений, связанных с планированием и управлением событиями.



## Приложение А

Общий вид UML-схемы подклассов на примере QWidget:



## Приложение Б

### Общий вид объявления подклассов на примере **myQRadioButton:**

```
class myQRadioButton: public QRadioButton {
    Q_OBJECT
public:
    myQRadioButton(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};
```

## Приложение В

### Общий вид реализации подклассов на примере `myQRadioButton`:

```
myQRadioButton::myQRadioButton(QWidget
*parent):QRadioButton(parent){setMouseTracking(true);}

void myQRadioButton::mouseMoveEvent(QMouseEvent *e)
{
    emit info("<h1>QRadioButton</h1><p>Позволяет пользователю
выбирать один из нескольких вариантов, как и QCheckBox, <br>но
обычно используется для взаимоисключающих выборов.<p>");
    QRadioButton::mouseMoveEvent(e);
}
```



## Приложение Г

### Исходный код widget.h:

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QFrame>

#include <QCheckBox>
#include <QComboBox>
#include <QCommandLinkButton>
#include <QDateEdit>
#include <QDateTimeEdit>
#include <QDial>
#include <QDoubleSpinBox>
#include <QFocusFrame>
#include <QFontComboBox>
#include <QLCDNumber>
#include <QLabel>
#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 <QToolButton>
#include <QWidget>
```

```

#include <QCalendarWidget>
#include <QListView>
#include <QTableView>
#include <QTreeView>
#include <QUndoView>

#include <QGroupBox>
#include <QStackedWidget>

class Widget : public QWidget {
    Q_OBJECT
public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

public slots:
    void changeInfo(QString);
private:
    QFrame *frame;
    QLabel *frame_description_text;
};

class myQCheckBox: public QCheckBox {
    Q_OBJECT
public:
    myQCheckBox(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQComboBox: public QComboBox {
    Q_OBJECT
public:
    myQComboBox(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

class myQCommandLinkButton: public QCommandLinkButton {
    Q_OBJECT
public:
    myQCommandLinkButton(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

class myQDateEdit: public QDateEdit {
    Q_OBJECT
public:
    myQDateEdit(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

class myQDateTimeEdit: public QDateTimeEdit {
    Q_OBJECT
public:
    myQDateTimeEdit(QWidget* parent = nullptr);
protected:

```

```

        void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
        void info(QString);
};

class myQDial: public QDial {
        Q_OBJECT
public:
        myQDial(QWidget* parent = nullptr);
protected:
        void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
        void info(QString);
};

class myQDoubleSpinBox: public QDoubleSpinBox {
        Q_OBJECT
public:
        myQDoubleSpinBox(QWidget* parent = nullptr);
protected:
        void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
        void info(QString);
};

class myQFocusFrame: public QFocusFrame {
        Q_OBJECT
public:
        myQFocusFrame(QWidget* parent = nullptr);
protected:
        void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
        void info(QString);
};

```

```

class myQFontComboBox: public QFontComboBox {
    Q_OBJECT
public:
    myQFontComboBox(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQLCDNumber: public QLCDNumber {
    Q_OBJECT
public:
    myQLCDNumber(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQLabel: public QLabel {
    Q_OBJECT
public:
    myQLabel(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQLineEdit: public QLineEdit {
    Q_OBJECT
public:
    myQLineEdit(QWidget* parent = nullptr);
protected:

```

```

        void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
        void info(QString);
};

class myQMenu: public QMenu {
        Q_OBJECT
public:
        myQMenu(QWidget* parent = nullptr);
protected:
        void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
        void info(QString);
};

class myQProgressBar: public QProgressBar {
        Q_OBJECT
public:
        myQProgressBar(QWidget* parent = nullptr);
protected:
        void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
        void info(QString);
};

class myQPushButton: public QPushButton {
        Q_OBJECT
public:
        myQPushButton(QWidget* parent = nullptr);
protected:
        void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
        void info(QString);
};

```

```

class myQRadioButton: public QRadioButton {
    Q_OBJECT
public:
    myQRadioButton(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQScrollArea: public QScrollArea {
    Q_OBJECT
public:
    myQScrollArea(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQScrollBar: public QScrollBar {
    Q_OBJECT
public:
    myQScrollBar(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQSizeGrip: public QSizeGrip {
    Q_OBJECT
public:
    myQSizeGrip(QWidget* parent = nullptr);

```

```

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

class myQSlider: public QSlider {
    Q_OBJECT
public:
    myQSlider(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

class myQSpinBox: public QSpinBox {
    Q_OBJECT
public:
    myQSpinBox(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

class myQTabBar: public QTabBar {
    Q_OBJECT
public:
    myQTabBar(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```



```

class myQTabWidget: public QTabWidget {
    Q_OBJECT
public:
    myQTabWidget(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQTimeEdit: public QTimeEdit {
    Q_OBJECT
public:
    myQTimeEdit(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQToolBox: public QToolBox {
    Q_OBJECT
public:
    myQToolBox(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQToolButton: public QToolButton {
    Q_OBJECT
public:
    myQToolButton(QWidget* parent = nullptr);

```

```

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

class myQWidget: public QWidget {
    Q_OBJECT
public:
    myQWidget(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

class myQCalendarWidget : public QCalendarWidget {
    Q_OBJECT
public:
    myQCalendarWidget(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

class myQListView: public QListView {
    Q_OBJECT
public:
    myQListView(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQTableView : public QTableView {
    Q_OBJECT
public:
    myQTableView(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQTreeView : public QTreeView {
    Q_OBJECT
public:
    myQTreeView(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQUndoView : public QUndoView {
    Q_OBJECT
public:
    myQUndoView(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

```

```

class myQGroupBox : public QGroupBox {
    Q_OBJECT
public:
    myQGroupBox(QWidget* parent = nullptr);

```

```

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

class myQStackedWidget: public QStackedWidget {
    Q_OBJECT
public:
    myQStackedWidget(QWidget* parent = nullptr);
protected:
    void mouseMoveEvent(QMouseEvent* mouse_event);
signals:
    void info(QString);
};

#endif // WIDGET_H

```

## Исходный код widget.cpp:

```
#include "widget.h"
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QTabWidget>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    QHBoxLayout *mainLayout = new QHBoxLayout(this);
    QTabWidget *tab = new QTabWidget(this);
    tab->setMinimumSize(400, 350);
    mainLayout->addWidget(tab);

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

    QWidget *tab2 = new QWidget(tab);
    tab->addTab(tab2, "&Advanced widgets");

    QWidget *tab3 = new QWidget(tab);
    tab->addTab(tab3, "&Organiser widgets");

    QVBoxLayout *layout_tab1 = new QVBoxLayout(tab1);

    myQCheckBox * thisQCheckBox= new myQCheckBox(tab1);
    connect(thisQCheckBox, SIGNAL(info(QString)), this,
    SLOT(changeInfo(QString)));
    layout_tab1->addWidget(thisQCheckBox);
    myQComboBox * thisQComboBox= new myQComboBox(tab1);
    connect(thisQComboBox, SIGNAL(info(QString)), this,
    SLOT(changeInfo(QString)));
    layout_tab1->addWidget(thisQComboBox);
```

```

        myQCommandLinkButton * thisQCommandLinkButton= new
myQCommandLinkButton(tab1);

        connect(thisQCommandLinkButton, SIGNAL(info(QString)),
this, SLOT(changeInfo(QString)));

        layout_tab1->addWidget(thisQCommandLinkButton);

        myQDateEdit * thisQDateEdit= new myQDateEdit(tab1);

        connect(thisQDateEdit, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab1->addWidget(thisQDateEdit);

        myQDateTimeEdit * thisQDateTimeEdit= new
myQDateTimeEdit(tab1);

        connect(thisQDateTimeEdit, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab1->addWidget(thisQDateTimeEdit);

        myQDial * thisQDial= new myQDial(tab1);

        connect(thisQDial, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab1->addWidget(thisQDial);

        myQDoubleSpinBox * thisQDoubleSpinBox= new
myQDoubleSpinBox(tab1);

        connect(thisQDoubleSpinBox, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab1->addWidget(thisQDoubleSpinBox);

        myQFocusFrame * thisQFocusFrame= new myQFocusFrame(tab1);

        connect(thisQFocusFrame, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab1->addWidget(thisQFocusFrame);

        myQFontComboBox * thisQFontComboBox= new
myQFontComboBox(tab1);

        connect(thisQFontComboBox, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab1->addWidget(thisQFontComboBox);

        myQLCDNumber * thisQLCDNumber= new myQLCDNumber(tab1);

        connect(thisQLCDNumber, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

```

```

        layout_tab1->addWidget(thisQLCDNumber);
        myQLabel * thisQLabel= new myQLabel(tab1);
        connect(thisQLabel,      SIGNAL(info(QString)),      this,
        SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQLabel);
        myQLineEdit * thisQLineEdit= new myQLineEdit(tab1);
        connect(thisQLineEdit,      SIGNAL(info(QString)),      this,
        SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQLineEdit);
        myQMenu * thisQMenu= new myQMenu(tab1);
        connect(thisQMenu,      SIGNAL(info(QString)),      this,
        SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQMenu);
        myQProgressBar      *      thisQProgressBar=      new
myQProgressBar(tab1);
        connect(thisQProgressBar,      SIGNAL(info(QString)),      this,
        SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQProgressBar);
        myQPushButton * thisQPushButton= new myQPushButton(tab1);
        connect(thisQPushButton,      SIGNAL(info(QString)),      this,
        SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQPushButton);
        myQRadioButton      *      thisQRadioButton=      new
myQRadioButton(tab1);
        connect(thisQRadioButton,      SIGNAL(info(QString)),      this,
        SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQRadioButton);
        myQScrollArea * thisQScrollArea= new myQScrollArea(tab1);
        connect(thisQScrollArea,      SIGNAL(info(QString)),      this,
        SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQScrollArea);
        myQScrollBar * thisQScrollBar= new myQScrollBar(tab1);
        connect(thisQScrollBar,      SIGNAL(info(QString)),      this,
        SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQScrollBar);

```

```

        myQSizeGrip * thisQSizeGrip= new myQSizeGrip(tab1);
        connect(thisQSizeGrip,    SIGNAL(info(QString)),    this,
SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQSizeGrip);
        myQSlider * thisQSlider= new myQSlider(tab1);
        connect(thisQSlider,    SIGNAL(info(QString)),    this,
SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQSlider);
        myQSpinBox * thisQSpinBox= new myQSpinBox(tab1);
        connect(thisQSpinBox,    SIGNAL(info(QString)),    this,
SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQSpinBox);
        myQTabBar * thisQTabBar= new myQTabBar(tab1);
        connect(thisQTabBar,    SIGNAL(info(QString)),    this,
SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQTabBar);
        myQTabWidget * thisQTabWidget= new myQTabWidget(tab1);
        connect(thisQTabWidget,    SIGNAL(info(QString)),    this,
SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQTabWidget);
        myQTimeEdit * thisQTimeEdit= new myQTimeEdit(tab1);
        connect(thisQTimeEdit,    SIGNAL(info(QString)),    this,
SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQTimeEdit);
        myQToolBox * thisQToolBox= new myQToolBox(tab1);
        connect(thisQToolBox,    SIGNAL(info(QString)),    this,
SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQToolBox);
        myQToolButton * thisQToolButton= new myQToolButton(tab1);
        connect(thisQToolButton,    SIGNAL(info(QString)),    this,
SLOT(changeInfo(QString)));
        layout_tab1->addWidget(thisQToolButton);
        myQWidget * thisQWidget= new myQWidget(tab1);
        connect(thisQWidget,    SIGNAL(info(QString)),    this,
SLOT(changeInfo(QString)));

```



```

        layout_tab1->addWidget(thisQWidget);

        QVBoxLayout * layout_tab2 = new QVBoxLayout(tab2);
        myQCalendarWidget * thisQCalendarWidget = new
myQCalendarWidget(tab2);

        connect(thisQCalendarWidget, SIGNAL(info(QString)),
this, SLOT(changeInfo(QString)));

        layout_tab2->addWidget(thisQCalendarWidget);
        myQListView * thisQListView = new myQListView(tab2);
        connect(thisQListView, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab2->addWidget(thisQListView);
        myQTableView * thisQTableView = new myQTableView(tab2);
        connect(thisQTableView, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab2->addWidget(thisQTableView);
        myQTreeView * thisQTreeView = new myQTreeView(tab2);
        connect(thisQTreeView, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab2->addWidget(thisQTreeView);
        myQUndoView * thisQUndoView = new myQUndoView(tab2);
        connect(thisQUndoView, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab2->addWidget(thisQUndoView);


        QVBoxLayout * layout_tab3 = new QVBoxLayout(tab3);
        myQGroupBox * thisQGroupBox = new myQGroupBox(tab3);
        connect(thisQGroupBox, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab3->addWidget(thisQGroupBox);
        myQStackedWidget * thisQStackedWidget = new
myQStackedWidget(tab3);

        connect(thisQStackedWidget, SIGNAL(info(QString)), this,
SLOT(changeInfo(QString)));

        layout_tab3->addWidget(thisQStackedWidget);

```

```

        frame = new QFrame(this); //basic, advanced widgets;
        QVBoxLayout * frame_layout = new QVBoxLayout(frame);
        frame->setLayout(frame_layout);
        frame->setMinimumSize(650,500);
        mainLayout->addWidget(frame);

        frame_description_text = new QLabel(frame);
        frame_layout->addWidget(frame_description_text);
        frame_description_text->setText("<h1>This          is
application</h1><h1>where    u    can    see</h1><h1>all    widgets
and</h1><h1>get some information</h1><h1>about them</h1>");
        frame_description_text->setMinimumSize(500, 500);

    }

```

```

Widget::~Widget()

```

```

{
}

```

```

void Widget::changeInfo(QString s)

```

```

{
    frame_description_text->setText(s);
}

```

```

myQCheckBox::myQCheckBox(QWidget *parent):QCheckBox(parent)
{setMouseTracking(true);}

```

```

void myQCheckBox::mouseMoveEvent(QMouseEvent *e)

```

```

{emit    info("<h1>QCheckBox</h1><p>Виджет,    представляющий
собой флажок, который пользователь может<br> установить или снять.

```

Основное предназначение - предоставление возможности выбора одного или нескольких вариантов из предложенного списка.

```
QCheckBox::mouseMoveEvent(e);}  
myQComboBox::myQComboBox(QWidget *parent):QComboBox(parent)  
{setMouseTracking(true);}  
void myQComboBox::mouseMoveEvent(QMouseEvent *e)  
{emit info("<h1>QComboBox</h1><p>Элемент интерфейса,  
представляющий собой выпадающий список с выбором одного  
элемента из списка опций. Очень полезен для выбора одного  
варианта из набора.");}  
QComboBox::mouseMoveEvent(e);}  
myQCommandLinkButton::myQCommandLinkButton(QWidget  
*parent):QCommandLinkButton(parent)  
{setMouseTracking(true);}  
void myQCommandLinkButton::mouseMoveEvent(QMouseEvent *e)  
{emit info("<h1>QCommandLinkButton</h1><p>Элемент  
интерфейса, который предоставляет текстовую команду или  
действие, которое пользователь может выполнить. Часто используется  
для создания диалоговых окон с явными действиями.");}  
QCommandLinkButton::mouseMoveEvent(e);}  
myQDateEdit::myQDateEdit(QWidget *parent):QDateEdit(parent)  
{setMouseTracking(true);}  
void myQDateEdit::mouseMoveEvent(QMouseEvent *e)  
{emit info("<h1>QDateEdit</h1><p>Виджет для редактирования  
даты. Позволяет пользователю вводить и редактировать даты в  
удобной форме.");}  
QDateEdit::mouseMoveEvent(e);}  
myQDateTimeEdit::myQDateTimeEdit(QWidget  
*parent):QDateTimeEdit(parent)  
{setMouseTracking(true);}  
void myQDateTimeEdit::mouseMoveEvent(QMouseEvent *e)  
{emit info("<h1>QDateTimeEdit</h1><p>Аналогичен QDateEdit,  
но предназначен для редактирования даты и времени в одном  
виджете.");}  
QDateTimeEdit::mouseMoveEvent(e);}
```

```

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

void myQDial::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QDial</h1><p>Компонент интерфейса,
представляющий собой круглый регулятор, который может<br> быть
использован для выбора числовых значений в диапазоне.<p>");
QDial::mouseMoveEvent(e);}

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

void myQDoubleSpinBox::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QDoubleSpinBox</h1><p>Виджет для ввода
числовых значений с плавающей точкой. Позволяет <br>пользователю
выбирать числа с десятичной частью.<p>");
QDoubleSpinBox::mouseMoveEvent(e);}

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

void myQFocusFrame::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QFocusFrame</h1><p>Элемент, используемый для
отображения текущего активного элемента <br>управления в
интерфейсе. Обычно используется в качестве визуального индикатора
<br>фокуса.<p>");
QFocusFrame::mouseMoveEvent(e);}

myQFontComboBox::myQFontComboBox(QWidget
*parent):QFontComboBox(parent)
{setMouseTracking(true);}

void myQFontComboBox::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QFontComboBox</h1><p>Виджет, предоставляющий
пользователю выбор шрифта из доступных на <br>системе. Полезен для
настройки внешнего вида текста.<p>");
QFontComboBox::mouseMoveEvent(e);}

myQLCDNumber::myQLCDNumber(QWidget
*parent):QLCDNumber(parent)
{setMouseTracking(true);}

```

```

void myQLCDNumber::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QLCDNumber</h1><p>Отображает цифровое
значение на экране, часто используется для отображения
<br>числовых данных, таких как значения счетчиков.<p>");
QLCDNumber::mouseMoveEvent(e);}

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

void myQLabel::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QLabel</h1><p>Простой виджет для отображения
текста или изображений. Используется для вывода <br>информации и
меток в интерфейсе.<p>");
QLabel::mouseMoveEvent(e);}

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

void myQLineEdit::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QLineEdit</h1><p>Поле ввода текста, которое
позволяет пользователю вводить и редактировать текст.<p>");
QLineEdit::mouseMoveEvent(e);}

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

void myQMenu::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QMenu</h1><p>Виджет, представляющий
контекстное меню с вариантами действий. Обычно появляется при
<br>правом клике мыши.<p>");
QMenu::mouseMoveEvent(e);}

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

void myQProgressBar::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QProgressBar</h1><p>Отображает прогресс
выполнения задачи или операции. Полезен для визуализации <br>хода
выполнения длительных задач.<p>");
QProgressBar::mouseMoveEvent(e);}

myQPushButton::myQPushButton(QWidget
*parent):QPushButton(parent)

```

```

        {setMouseTracking(true);}

void myQPushButton::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QPushButton</h1><p>Кнопка, которая может быть
нажата пользователем для выполнения определенного
<br>действия.<p>");
    QPushButton::mouseMoveEvent(e);}

myQRadioButton::myQRadioButton(QWidget
*parent):QRadioButton(parent)
{setMouseTracking(true);}

void myQRadioButton::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QRadioButton</h1><p>Позволяет пользователю
выбирать один из нескольких вариантов, как и QCheckBox, <br>но
обычно используется для взаимоисключающих выборов.<p>");
    QRadioButton::mouseMoveEvent(e);}

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

void myQScrollArea::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QScrollArea</h1><p>Контейнер с полосами
прокрутки, который позволяет отображать содержимое, размер
<br>которого больше размеров виджета.<p>");
    QScrollArea::mouseMoveEvent(e);}

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

void myQScrollBar::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QScrollBar</h1><p>Предоставляет полосы
прокрутки для прокручивания содержимого внутри других виджетов,
<br>таких как QScrollArea.<p>");
    QScrollBar::mouseMoveEvent(e);}

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

void myQSizeGrip::mouseMoveEvent(QMouseEvent *e)

```

```

        {emit    info("<h1>QSizeGrip</h1><p>Виджет,    предоставляющий
пользователю возможность изменять размер окна, перетаскивая его
<br>границы.<p>");
        QSizeGrip::mouseMoveEvent(e);}
        myQSlider::myQSlider(QWidget *parent):QSlider(parent)
        {setMouseTracking(true);}
        void myQSlider::mouseMoveEvent(QMouseEvent *e)
        {emit        info("<h1>QSlider</h1><p>Ползунок,            который
пользователь    может    перетаскивать    для    выбора    значения    в
определенном    <br>диапазоне.<p>");
        QSlider::mouseMoveEvent(e);}
        myQSpinBox::myQSpinBox(QWidget *parent):QSpinBox(parent)
        {setMouseTracking(true);}
        void myQSpinBox::mouseMoveEvent(QMouseEvent *e)
        {emit    info("<h1>QSpinBox</h1><p>Позволяет    пользователю
выбирать    целочисленные    значения    с    помощью    кнопок    вверх    и
вниз.<p>");
        QSpinBox::mouseMoveEvent(e);}
        myQTabBar::myQTabBar(QWidget *parent):QTabBar(parent)
        {setMouseTracking(true);}
        void myQTabBar::mouseMoveEvent(QMouseEvent *e)
        {emit        info("<h1>QTabBar</h1><p>Элемент            интерфейса,
представляющий    собой    вкладки,    которые    позволяют    пользователю
<br>переключаться    между    разными    разделами    интерфейса.<p>");
        QTabBar::mouseMoveEvent(e);}
        myQTabWidget::myQTabWidget(QWidget
*parent):QTabWidget(parent)
        {setMouseTracking(true);}
        void myQTabWidget::mouseMoveEvent(QMouseEvent *e)
        {emit    info("<h1>QTabWidget</h1><p>Контейнер    для    управления
несколькими    страницами    с    помощью    вкладок.    Позволяет
<br>организовать    информацию    в    разделы.<p>");
        QTabWidget::mouseMoveEvent(e);}
        myQTimeEdit::myQTimeEdit(QWidget *parent):QTimeEdit(parent)
        {setMouseTracking(true);}

```

```

void myQTimeEdit::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QTimeEdit</h1><p>Аналогично QDateEdit, но
предназначен для редактирования времени.<p>");
QTimeEdit::mouseMoveEvent(e);}

myQToolBox::myQToolBox(QWidget *parent):QToolBox(parent)
{setMouseTracking(true);}

void myQToolBox::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QToolBox</h1><p>Контейнер для организации
групп элементов в виде раскрывающегося списка. Полезен <br>для
компактной организации информации.<p>");
QToolBox::mouseMoveEvent(e);}

myQToolButton::myQToolButton(QWidget
*parent):QToolButton(parent)
{setMouseTracking(true);}

void myQToolButton::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QToolButton</h1><p>Виджет-кнопка, который
может использоваться для выполнения действий в приложении.
<br>Этот виджет особенно полезен в панелях инструментов, где
<br>пользователь может быстро получить доступ к часто используемым
командам.<p>");
QToolButton::mouseMoveEvent(e);}

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

void myQWidget::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QWidget</h1><p>Базовый класс для всех
виджетов в Qt. Он предоставляет основу для создания
пользовательских <br>виджетов и обеспечивает базовую
функциональность, <br>такую как обработка событий и управление
графикой.<p>");
QWidget::mouseMoveEvent(e);}

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

void myQCalendarWidget::mouseMoveEvent(QMouseEvent *e)

```



```

    {emit info("<h1>QCalendarWidget</h1><p>Представляет собой
календарь, который может использоваться для отображения и выбора
<br>дат. Это полезно для приложений, связанных с планированием
<br>и управлением событиями.<p>");
    QCalendarWidget::mouseMoveEvent(e);}

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

void myQListView::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QListView</h1><p>Этот виджет представляет
собой список, используется для отображения и редактирования
<br>структурированных данных. QListView подходит для простых
списков.<p>");
    QListView::mouseMoveEvent(e);}

myQTableView::myQTableView(QWidget
*parent):QTableView(parent)
{setMouseTracking(true);}

void myQTableView::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QTableView</h1><p>Этот виджет представляет
собой таблицу, используется для отображения и редактирования
<br>структурированных данных. QTableView подходит для табличных
данных.<p>");
    QTableView::mouseMoveEvent(e);}

myQTreeView::myQTreeView(QWidget *parent):QTreeView(parent)
{setMouseTracking(true);}

void myQTreeView::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QTreeView</h1><p>Этот виджет представляет
собой дерево, используется для отображения и редактирования
<br>структурированных данных. QTableVQTreeView подходит для
иерархических данных.<p>");
    QTreeView::mouseMoveEvent(e);}

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

void myQUndoView::mouseMoveEvent(QMouseEvent *e)
{emit info("<h1>QUndoView</h1><p>Представляет собой виджет
для отображения и управления стеком отмены и повтора действий <br>в

```

приложении. Он полезен для создания интерфейса отмены и повтора  
<br>действий.<p>");

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

```
myQGroupBox::myQGroupBox(QWidget *parent):QGroupBox(parent)
{setMouseTracking(true);}
```

```
void myQGroupBox::mouseMoveEvent(QMouseEvent *e)
```

```
{emit info("<h1>QGroupBox</h1><p>Это контейнерный виджет,
который может содержать другие виджеты и предоставляет заголовок
<br>и рамку для их группировки. Он полезен для создания группы
связанных <br>элементов управления.<p>");
```

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

```
myQStackedWidget::myQStackedWidget(QWidget
*parent):QStackedWidget(parent)
```

```
{setMouseTracking(true);}
```

```
void myQStackedWidget::mouseMoveEvent(QMouseEvent *e)
```

```
{emit info("<h1>QStackedWidget</h1><p>Это виджет, который
позволяет отображать только один из своих дочерних виджетов
одновременно,<br> переключаясь между ними. Это полезно для
создания многократных <br>\"страниц \" в интерфейсе, где только
одна страница видна в данный момент времени.<p>");
```

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

## Исходный код main.cpp:

```
#include "widget.h"

#include <QApplication>

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