Лекция 9

СОЗДАНИЕ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА СРЕДСТВАМИ QT

Лекция 9. Создание графического интерфейса средствами Qt

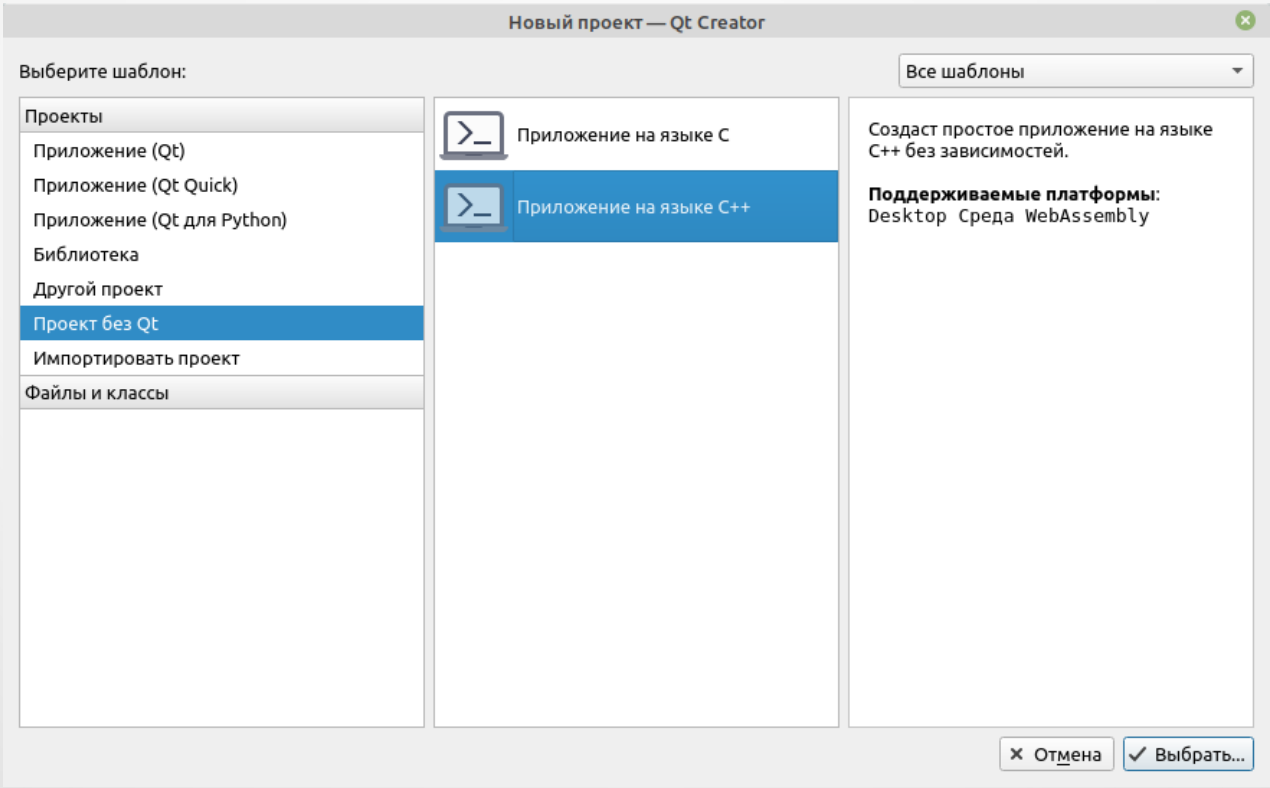
План

1. Простейшее приложение с GUI в Qt.
2. Понятие виджета. Класс QWidget.
3. Управление размещением компонентов виджета.
4. Обработка сигналов.
5. Главное окно и меню.

1. Простейшее приложение с GUI в Qt

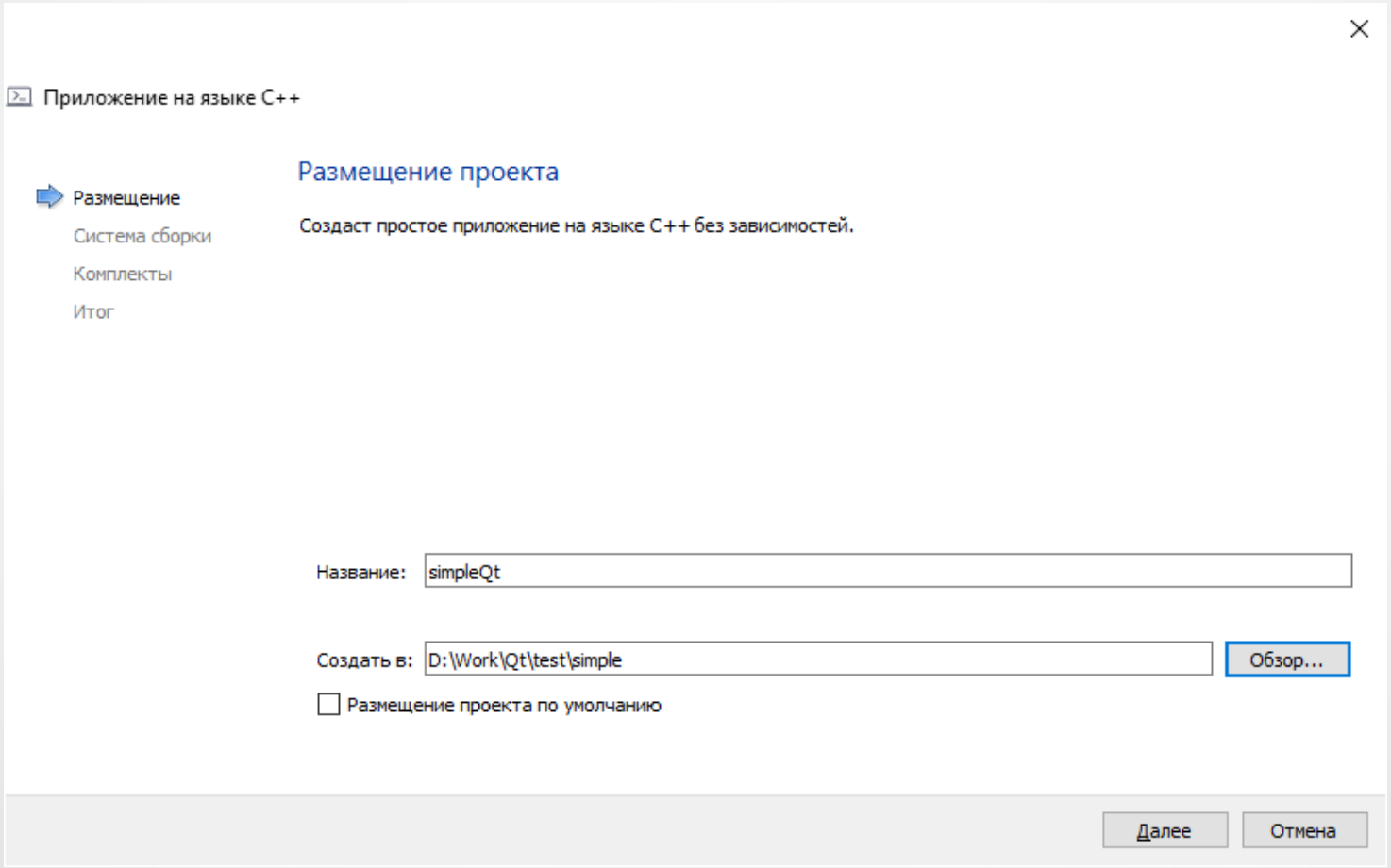
Для создания приложения с графическим интерфейсом пользователя в Qt Creator можно, например, выполнить следующую последовательность действий:

1) создать новый проект («Файл|Создать файл или проект...»), выбрав «Приложение на языке C++» в шаблоне «Проект без Qt»;



1. Простейшее приложение с GUI в Qt

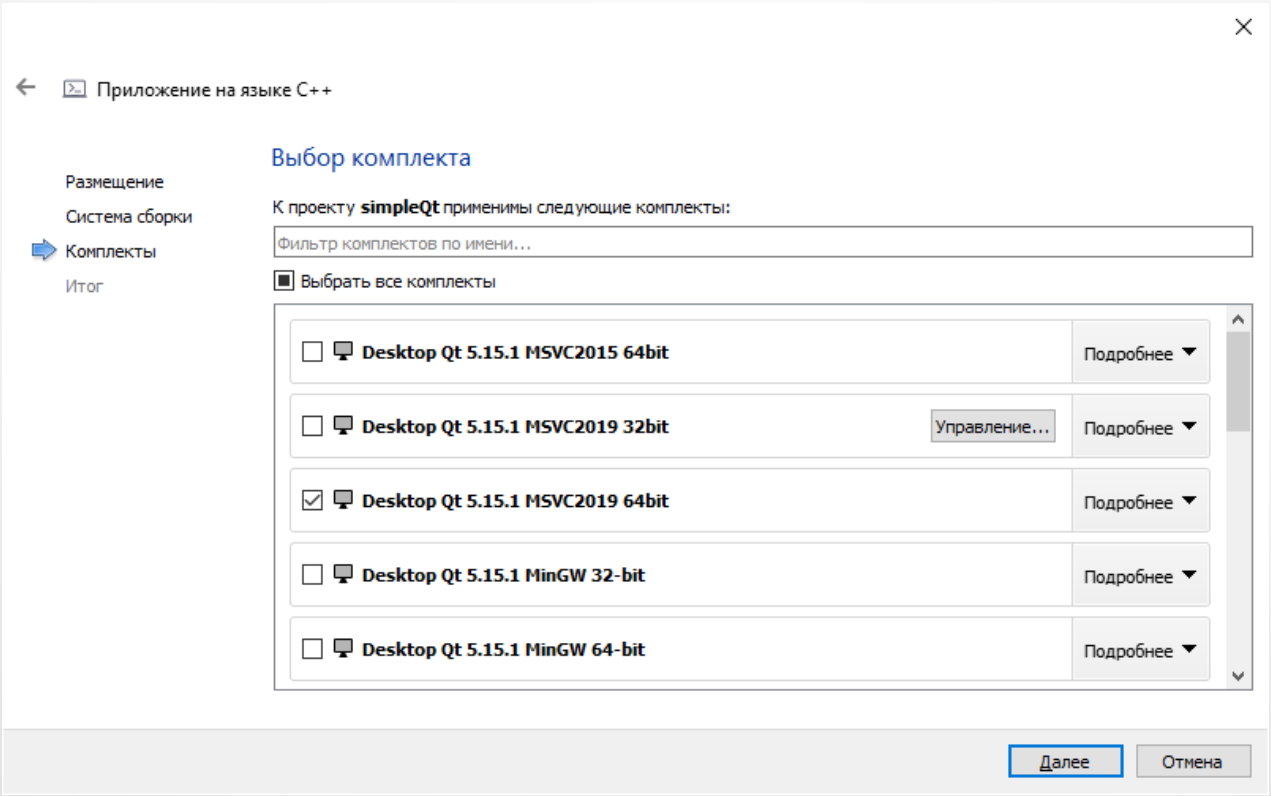
2) указать название проекта и его месторасположение;



1. Простейшее приложение с GUI в Qt

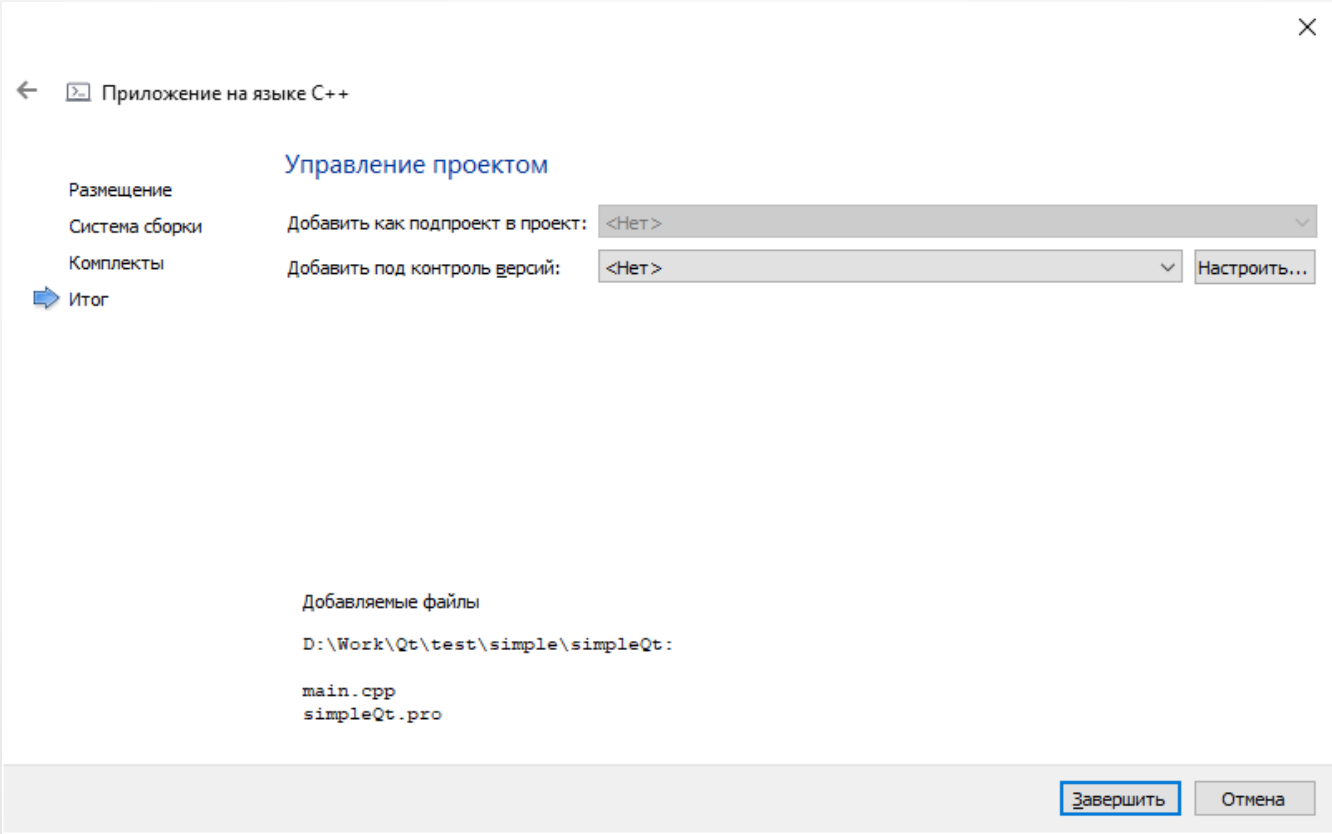
3) выбрать систему сборки (оптимально оставить предлагаемую по умолчанию **qmake**);

4) выбрать комплект компилятора и отладчика (зависит от используемой ОС, установленных в ней программ и требований к приложению);



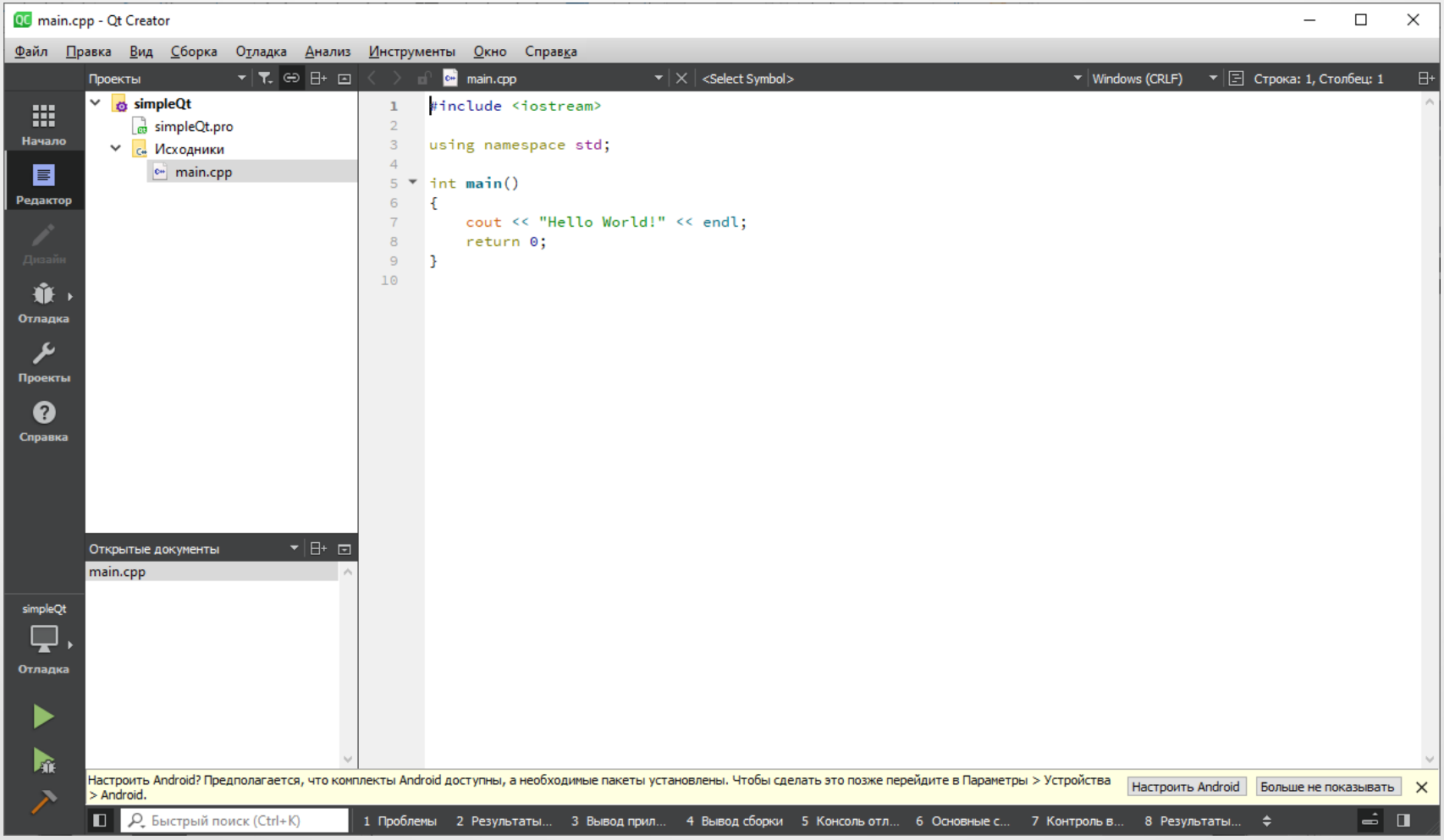
1. Простейшее приложение с GUI в Qt

5) выбрать способ управления проектом (в данном случае оптимальным вариантом будет оставить предложенный по умолчанию вариант);



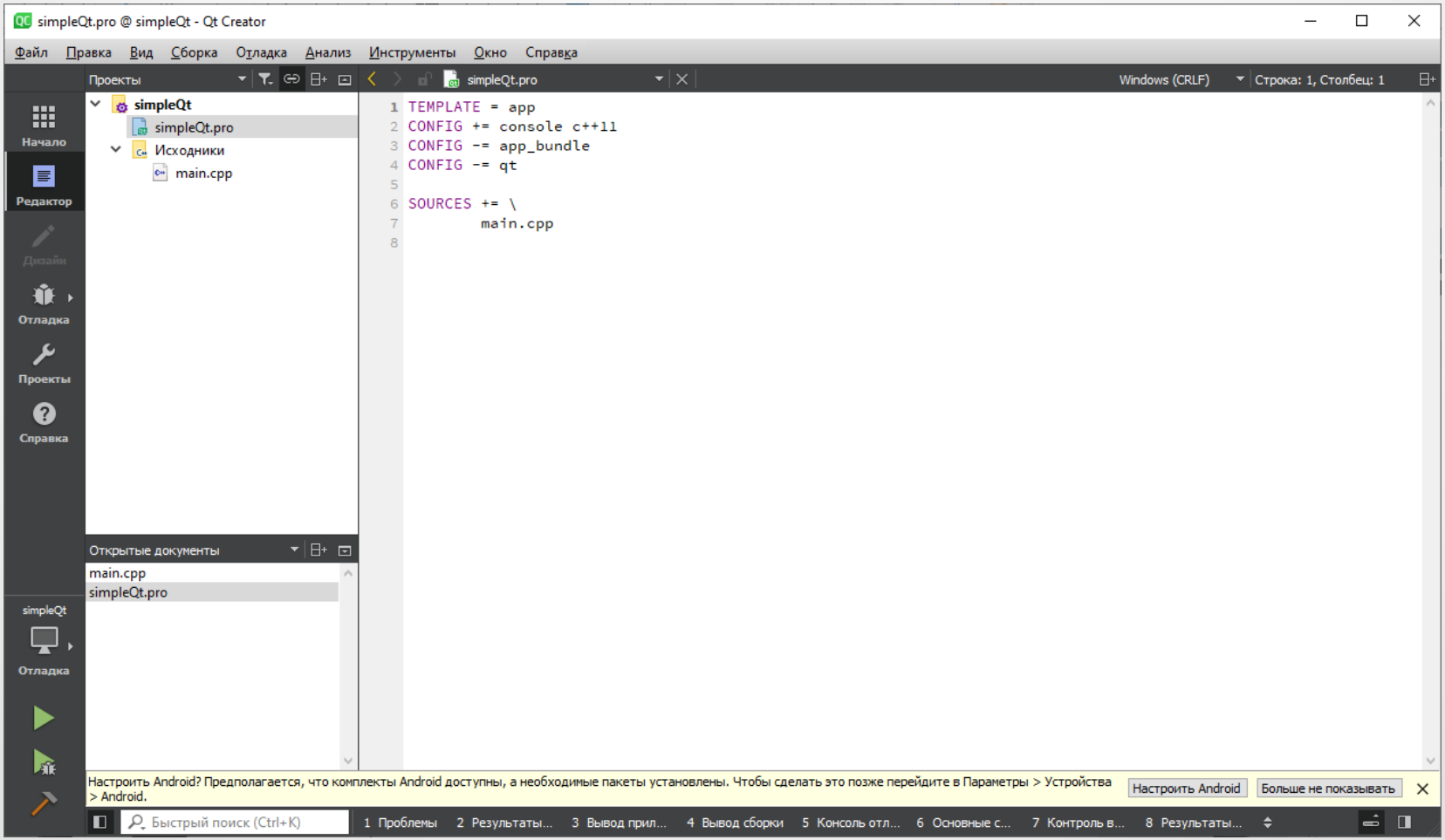
1. Простейшее приложение с GUI в Qt

После завершения выбора необходимых настроек проект будет создан и отображен в Qt Creator.



1. Простейшее приложение с GUI в Qt

Для перенастройки проекта с целью создания приложения с GUI необходимо внести соответствующие изменения в файл проекта (имеет расширение .pro)



1. Простейшее приложение с GUI в Qt

Для создания проекта программы с GUI средствами Qt необходимо, чтобы файл настроек содержал следующие параметры:

```
# Целью проекта является приложение (не библиотека)
TEMPLATE = app
# Используемый стандарт языка C++17 (не все компиляторы поддерживают)
CONFIG += c++17
# Использовать библиотеку Qt и ее соответствующие модули
QT += gui widgets

# Перечень исходных файлов проекта
SOURCES += \
        main.cpp
```

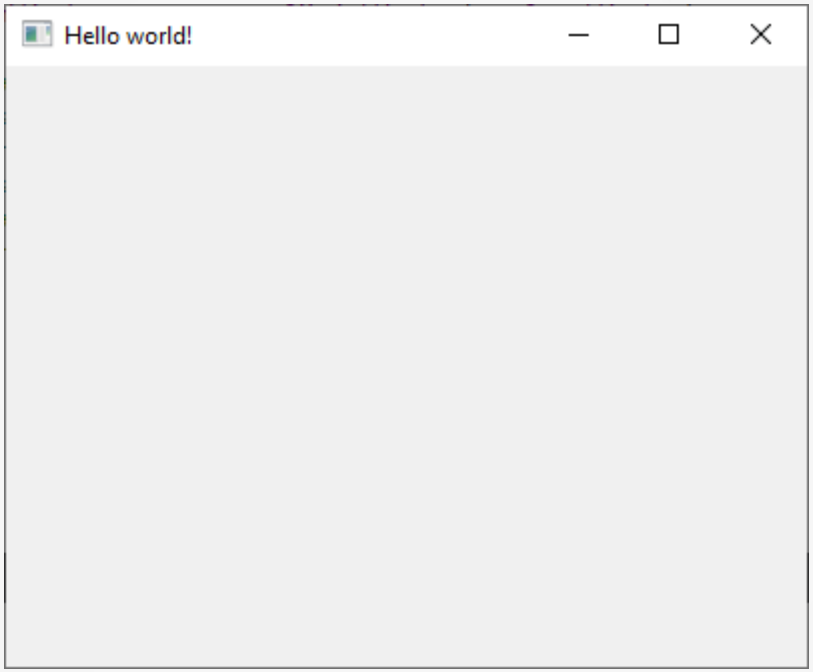
1. Простейшее приложение с GUI в Qt

После чего исходный текст программы (в нашем случае main.cpp) поменяем следующим образом:

```
#include <QApplication>
#include <QMainWindow>

int main(int argc, char *argv[])
{
    // Создание экземпляра приложения
    QApplication app(argc, argv);
    // Создание главного окна
    QMainWindow *mw = new QMainWindow(0, Qt::Window);

    // Настройка главного окна
    mw->setWindowTitle("Hello world!");
    mw->resize(400, 300);
    mw->show();
    // Запуск цикла обработки GUI
    return app.exec();
}
```

1. Простейшее приложение с GUI в Qt

Компиляция и запуск данной программы должен привести к следующему результату:



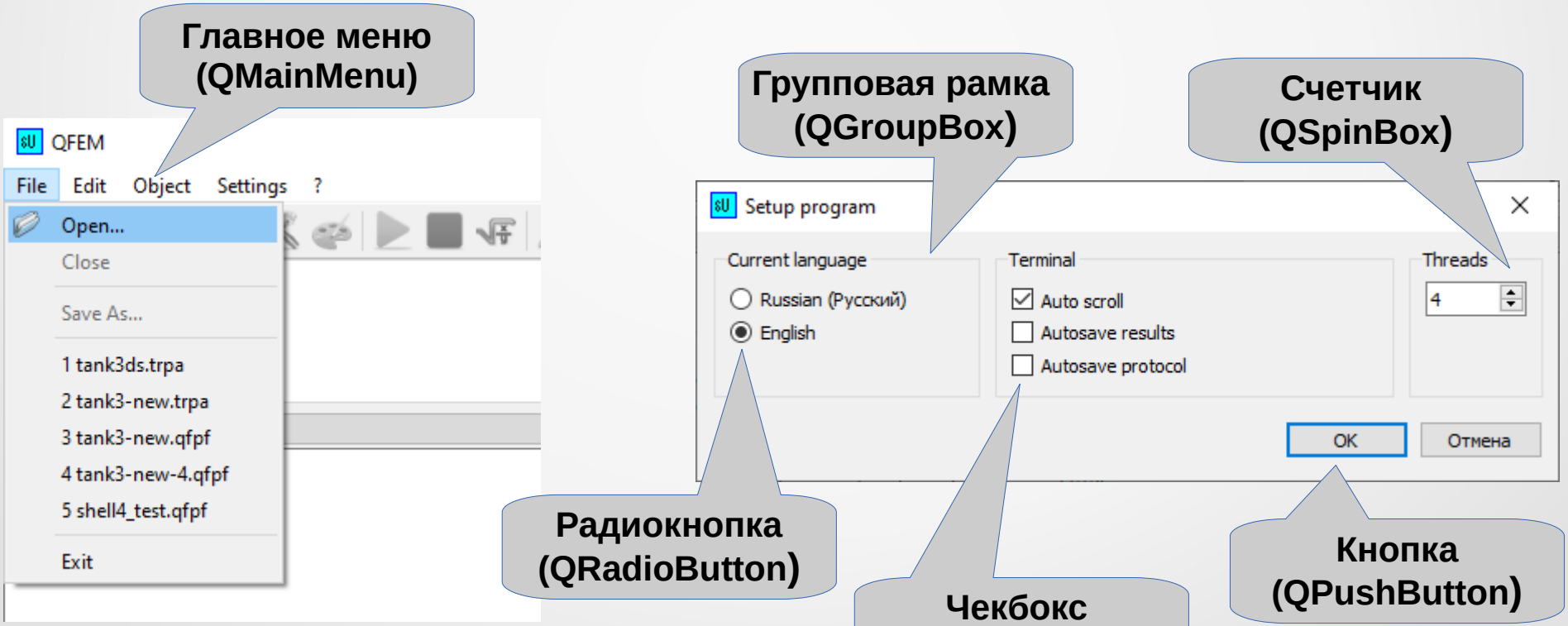
Простейшая Qt-программа с GUI

Примечание. Если проект не компилируется, можно попробовать выполнить команду главного меню «Сборка|Запустить qmake».

2. Понятие виджета

Центральным элементом графического интерфейса пользователя является **виджет** (**widget**) – примитив, имеющий некоторый стандартный вид и функционал.

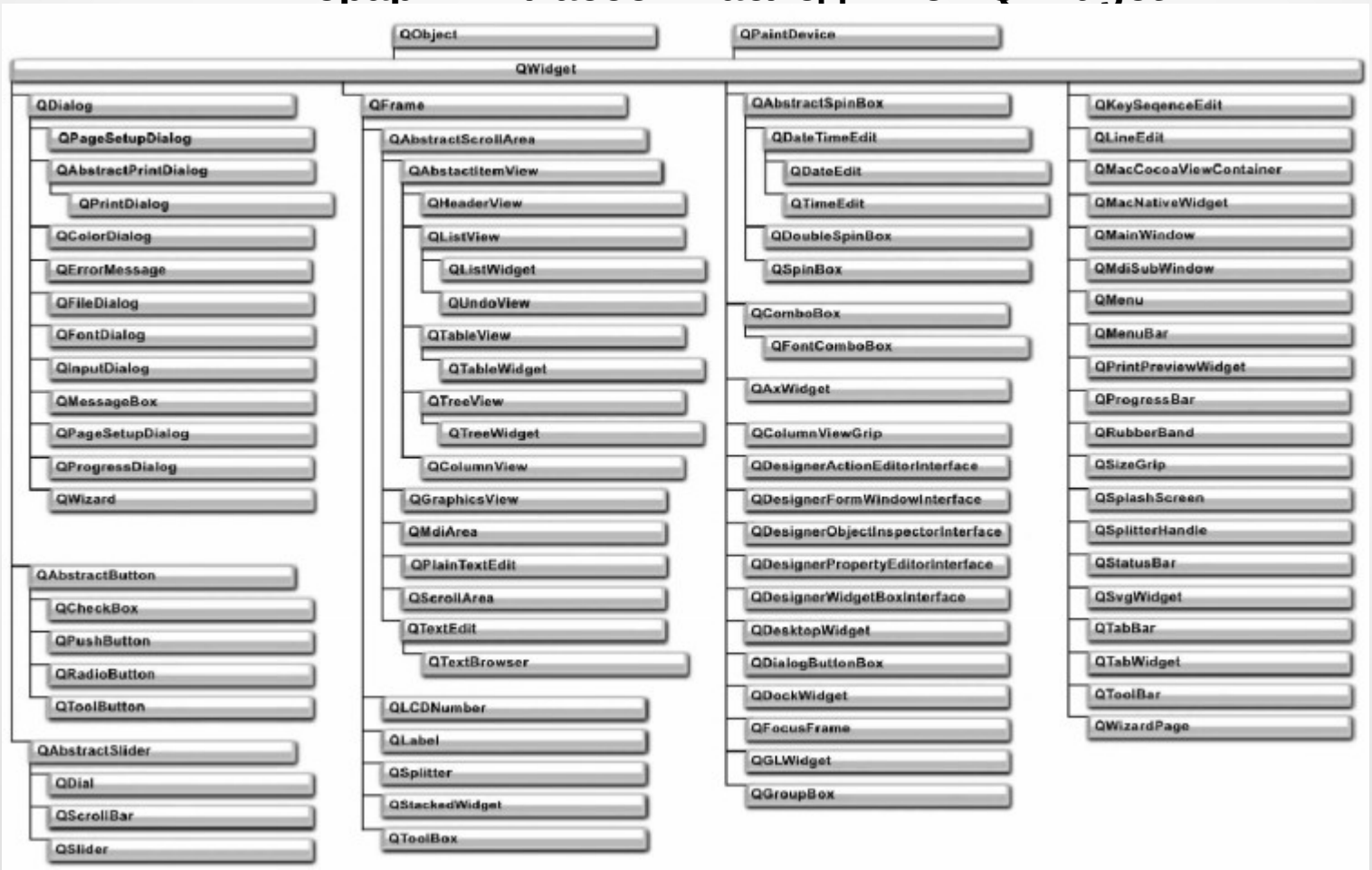
Примеры виджетов



2. Понятие виджета. Класс QWidget

В Qt реализованы все необходимые для создания GUI виджеты: окна, меню, кнопки, полосы прокрутки, чекбоксы и т. п. Все они реализуются как наследники класса **QWidget**.

Иерархия классов-наследников QWidget



2. Понятие виджета. Класс QWidget

Класс **QWidget** является наследником от **QObject** (может использовать **механизм сигналов-слотов**) и имеет более 250 методов и более 50 свойств. Каждый виджет может содержать внутри себя другие виджеты (т. е. являться их контейнером).

Для создания виджета используется конструктор, имеющий следующий шаблон:

QWidget(QWidget* parent = nullptr, Qt::WindowFlags flag = 0);

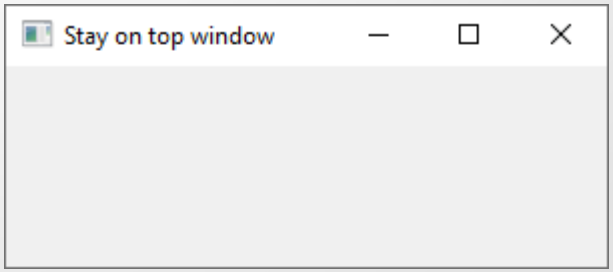
Параметр **parent** содержит указатель на родительский виджет (если этот параметр равен **nullptr**, то вновь созданный объект станет виджетом верхнего уровня). Параметр **flag** определяет свойства, внешний вид и режимом отображения (это может быть логическая комбинация флагов, например, Qt::Window | Qt::WindowTitleHint | Qt::WindowStaysOnTopHint).

2. Понятие виджета. Класс QWidget

Для изменения атрибутов виджета класс **QWidget** содержит метод **setWindowFlags()**. Для изменения заголовка окон верхнего уровня используется метод **setWindowTitle()**. Метод **show()** делает окно видимым. Метод **resize()** изменяет текущий размер окна.

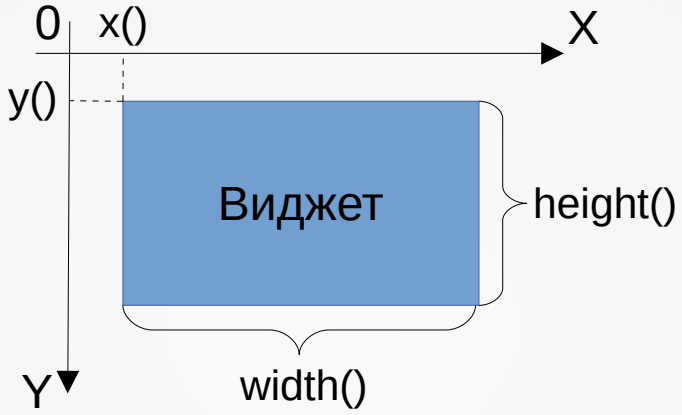
Например, следующий код создает и отображает окно верхнего уровня заданного размера, размещаемое поверх остальных окон, предварительно задав его заголовок:

```
// …
QWidget *widget = new QWidget();

widget->setWindowFlags(Qt::Window | Qt::WindowStaysOnTopHint);
widget->setWindowTitle("Stay on top window");
widget->resize(200, 100);
widget->show();
// …
```



2. Понятие виджета. Класс QWidget

Виджет занимает на экране прямоугольную область определенных габаритов.



Координаты верхнего левого угла виджета определяются методами **x()** и **y()**, а ширина и высота – **width()** и **height()**.

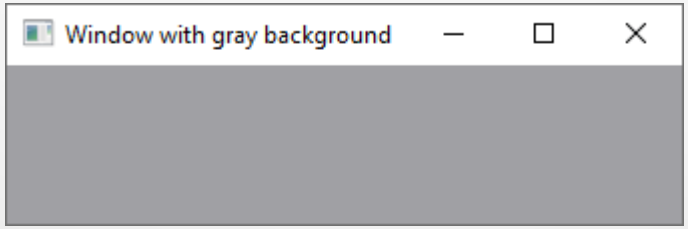
Положение виджета можно изменить методом **move()**. Одновременно изменить положение, и размеры виджета можно, вызвав метод **setGeometry()**. Например, переместить виджет в точку (10, 10) и задать его габариты 200x300 можно таким образом:

```
widget->setGeometry (10, 10, 200, 300);
```

2. Понятие виджета. Класс QWidget

Изменить цвет фона виджета можно с помощью метода **setPallete()**. Например:

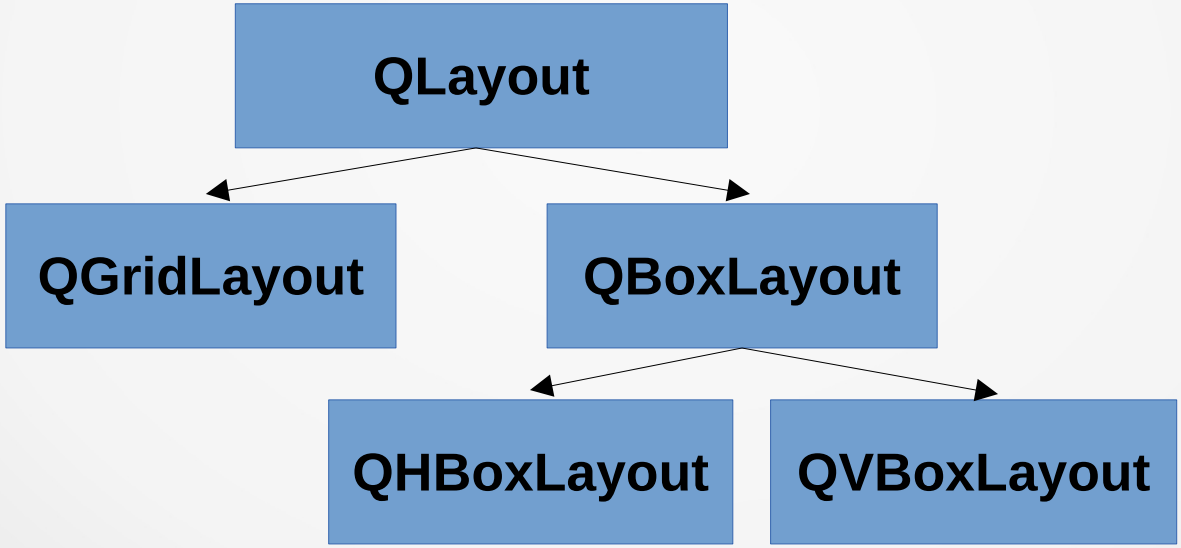
```
// …
#include <QPalette>

// …
// Создание палитры серого цвета
QPalette palette (Qt::gray);

// …
widget->setPalette(palette);
// …
```



3. Управление размещением компонентов виджета

Для управления **компоновкой** (размещением) элементов виджета в Qt используются специальный класс **QLayout** и его наследники.

Компоновка определяет расположение различных виджетов относительно друг друга.



Класс **QGridLayout** управляет табличной компоновкой, **QHBoxLayout** – горизонтальным, а **QVBoxLayout** – вертикальным размещением элементов.

3. Управление размещением компонентов виджета

Для управления горизонтальным или вертикальным размещением компонентов используется класс **QBoxLayout** или его специализированные наследники: **QHBoxLayout** и **QVBoxLayout**.

Для задания способа размещения первым параметром конструктора **QBoxLayout** должно быть одно из следующих значений:

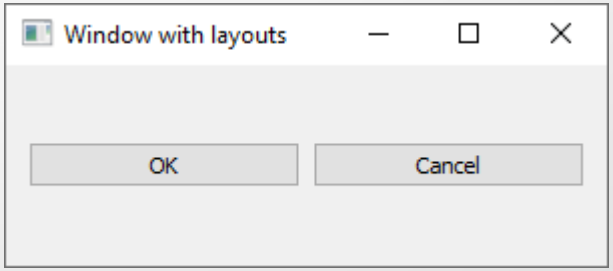
- **LeftToRight** – горизонтальное размещение, заполнение осуществляется слева направо;
- **RightToLeft** – горизонтальное размещение, заполнение выполняется справа налево;
- **TopToBottom** – вертикальное размещение, заполнение осуществляется сверху вниз;
- **BottomToTop** – вертикальное размещение, заполнение выполняется снизу вверх.

Для вставки в заданную позицию виджета используется метод **insertWidget()**, встроенной компоновки – **insertLayout()**, расстояния между виджетами – **insertSpacing()**, фактора растяжения – **insertStretch()** и заданного расстояния между двумя виджетами – **addSpacing()**.

3. Управление размещением компонентов виджета

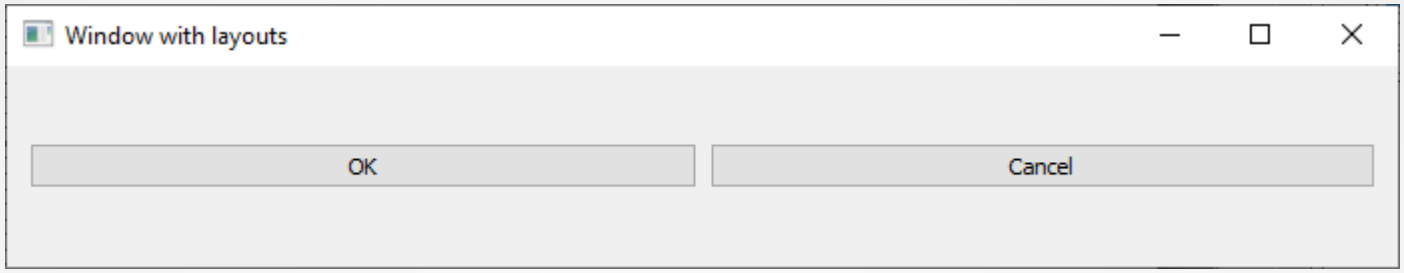
Простой пример использования компоновщика:

```
#include <QtWidgets>
#include <QLayout>
#include <QPushButton>

int main(int argc, char **argv)
{
    QApplication app(argc, argv);
    QWidget *widget = new QWidget();
    QBoxLayout *layout = new QBoxLayout(QBoxLayout::Direction::LeftToRight);

    layout->addWidget(new QPushButton("OK"));
    layout->addWidget(new QPushButton("Cancel"));

    widget->setWindowTitle("Window with layouts");
    widget->setLayout(layout);
    widget->resize(300, 100);

    widget->show();
    return app.exec();
}
```



3. Управление размещением компонентов виджета

В предыдущем примере компоненты виджета автоматически растягивались при изменении его размера:



Если нужно, чтобы кнопки оставались фиксированного размера, в компоновщик следует добавить **фактор растяжения**:


```
// …
layout->insertStretch(0);
// …
```

3. Управление размещением компонентов виджета

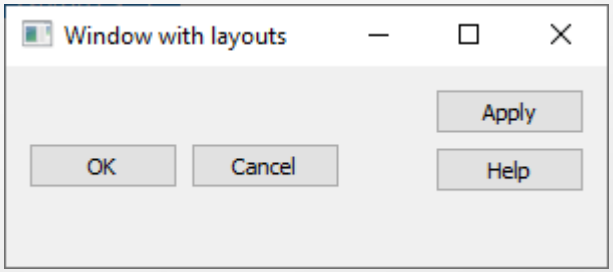
```
// …
layout->insertStretch(1);
// …
```


```
// …
layout->insertStretch(2);
// …
```

3. Управление размещением компонентов виджета

Компоновщики могут быть вложенными. Например:

```
// …
QBoxLayout *h_layout = new QBoxLayout(QBoxLayout::Direction::LeftToRight),
                   *v_layout = new QBoxLayout(QBoxLayout::Direction::BottomToTop);

v_layout->addWidget(new QPushButton("Help"));
v_layout->addWidget(new QPushButton("Apply"));
v_layout->insertStretch(0);

h_layout->addWidget(new QPushButton("OK"));
h_layout->addWidget(new QPushButton("Cancel"));
h_layout->addLayout(v_layout);
h_layout->insertStretch(2);

widget->setLayout(h_layout);
// …
```

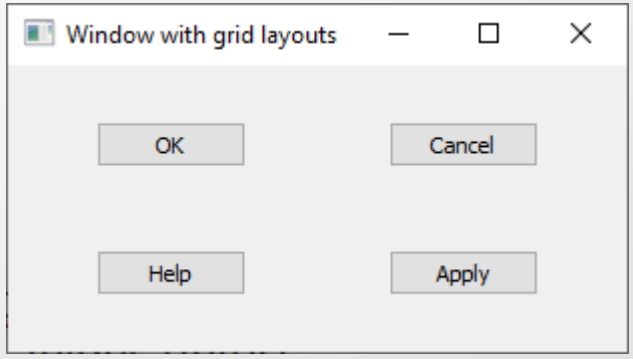


3. Управление размещением компонентов виджета

Для табличного размещения компонентов на виджете используется класс **QGridLayout**, с помощью которого можно быстро создавать сложные по структуре размещения.

Таблица состоит из ячеек, позиции которых задаются строками и столбцами. Например:

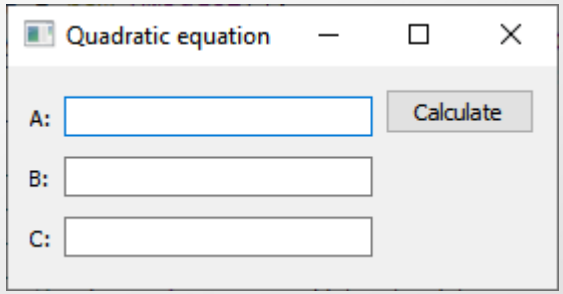
```
// …
QGridLayout *layout = new QGridLayout();

layout->addWidget(new QPushButton("OK"), 0, 0, Qt::AlignCenter);
layout->addWidget(new QPushButton("Cancel"), 0, 1, Qt::AlignCenter);
layout->addWidget(new QPushButton("Help"), 1, 0, Qt::AlignCenter);
layout->addWidget(new QPushButton("Apply"), 1, 1, Qt::AlignCenter);

widget->setWindowTitle("Window with grid layouts");
widget->setLayout(layout);
// …
```



3. Управление размещением компонентов виджета

Например, компоновку виджетов для программы решения квадратного уравнения можно выполнить следующим образом:

```
#include <QLabel>
#include <QLineEdit>
// ...
QBoxLayout *layout = new QBoxLayout(QBoxLayout::Direction::LeftToRight),
                   *v_layout = new QBoxLayout(QBoxLayout::Direction::TopToBottom);
QGridLayout *g_layout = new QGridLayout();

g_layout->addWidget(new QLabel("A:"), 0, 0);
g_layout->addWidget(new QLabel("B:"), 1, 0);
g_layout->addWidget(new QLabel("C:"), 2, 0);
g_layout->addWidget(new QLineEdit(), 0, 1);
g_layout->addWidget(new QLineEdit(), 1, 1);
g_layout->addWidget(new QLineEdit(), 2, 1);

v_layout->addWidget(new QPushButton("Calculate"));
v_layout->addStretch(0);

layout->addLayout(g_layout);
layout->addLayout(v_layout);

widget->setLayout(layout);
// ...
```



4. Обработка сигналов

Как уже отмечалось, для **обмена сообщениями** между виджетами в Qt используются **сигналы** и **слоты**. Однако, поскольку это объектно-ориентированный механизм, для его использования графический интерфейс программы придется описывать с помощью классов.

Рассмотрим пример создания создания окна, информирующего пользователя о возникшей ошибке. Элементами интерфейса данного окна будут:

- виджет, содержащий строку сообщения;
- кнопка «ОК», закрывающая окно.

Для создания такого компонента добавим к проекту заголовочный файл, который, например, назовем message.h.

После чего в данном файле реализуем, например, класс **TMessage**, являющийся наследником от класса **TWidget**.

4. Обработка сигналов

Содержимое файла **message.h**:

```
#ifndef MESSAGE_H
#define MESSAGE_H

#include <QWidget>
#include <QPushButton>
#include <QBoxLayout>
#include <QLabel>

class TMessage : public QWidget
{
    Q_OBJECT

private:
    QWidget *window = nullptr;
    QPushButton *ok = nullptr;

private slots:
    void slotClose(void)
    {
       window->close();
    }
```

4. Обработка сигналов

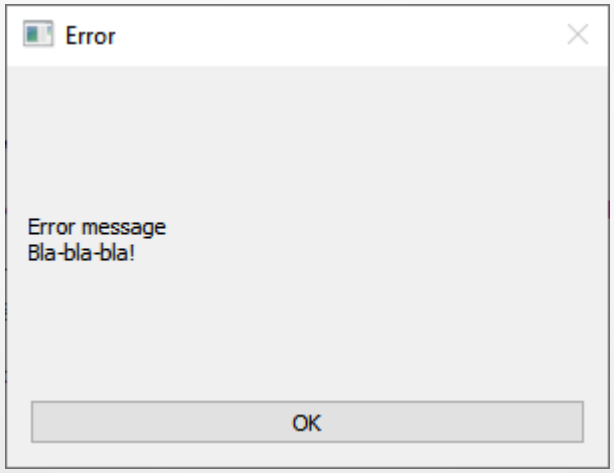
Содержимое файла **message.h** (продолжение):

```
public:
    TMessage(QString title, QString msg, QWidget *parent = nullptr) : QWidget(parent)
    {
        QBoxLayout *layout = new QBoxLayout(QBoxLayout::Direction::BottomToTop);

        layout->addWidget(ok = new QPushButton("OK", this));
        layout->addWidget(new QLabel(msg));

        window = new QWidget(parent,  Qt::WindowSystemMenuHint);
        window->setLayout(layout);
        window->setWindowTitle(title);
        window->resize(300, 200);
        QWidget::connect(ok, SIGNAL(clicked()), this, SLOT(slotClose()));
    }
    ~TMessage() = default;
    void show(void)
    {
       window->show();
    }
};
#endif // MESSAGE_H
```

4. Обработка сигналов

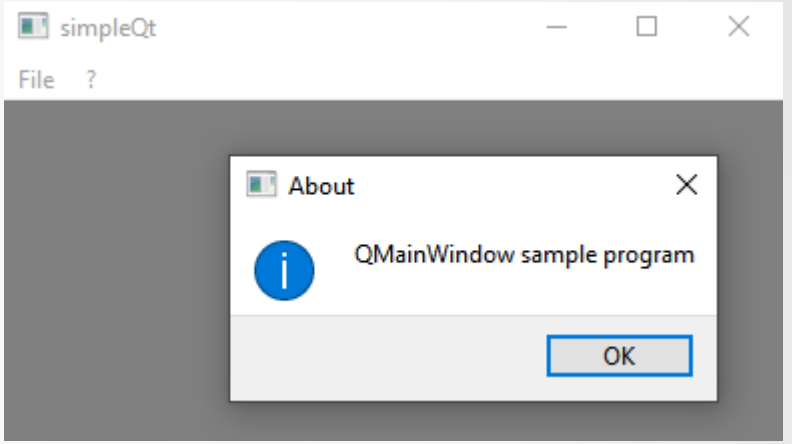
Тогда использование этого класса можно выполнить таким образом:

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

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    TMessage msg("Error", "Error message\nBla-bla-bla!");

    msg.show();
    return app.exec();
}
```



5. Главное окно и меню

Класс **QMainWindow** – реализует **главное окно программы**, содержащее такие типовые виджеты, как рабочую область, меню, панели инструментов, полосы прокрутки, строки состояния и т. п. Например:




```
#include <QApplication>
#include <QMainWindow>
#include <QMessageBox>
#include <QMenuBar>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QMainWindow *window = new QMainWindow();
    QMenu *menuFile = new QMenu("&File"),
          *menuHelp = new QMenu("&?");
    QMessageBox *mbox = new QMessageBox(QMessageBox::Information, "About",
                                        "QMainWindow sample program");

    menuFile->addAction("E&xit", [window]() -> void{ window->close(); });
    menuHelp->addAction("&About", [mbox]() { mbox->exec(); });
    window->menuBar()->addMenu(menuFile);
    window->menuBar()→addMenu(menuHelp);
    window->setStyleSheet("QMainWindow {background: 'gray';}");
    window->show();
    return app.exec();
}
```