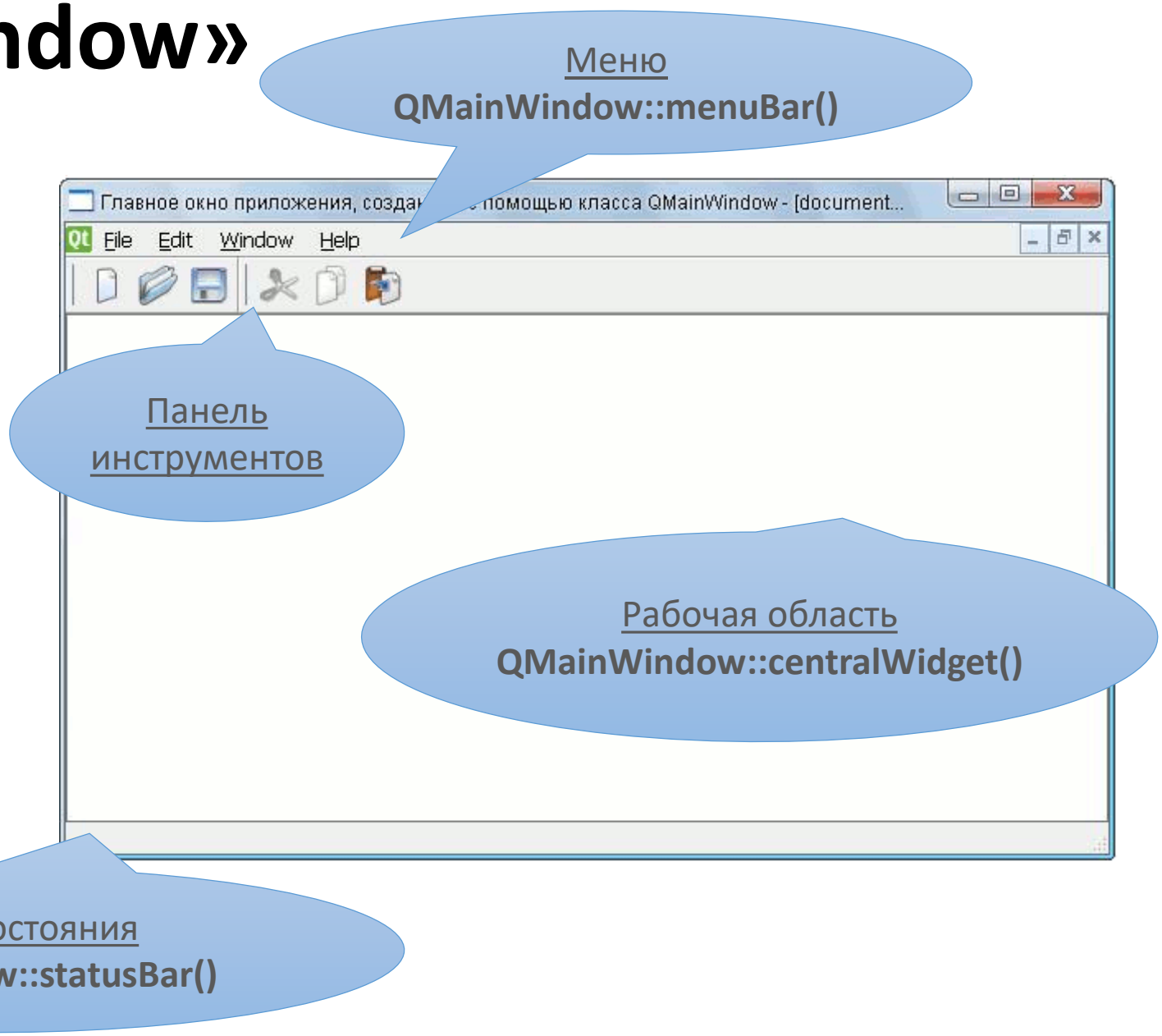
Главное окно. Меню. Диалоговые окна.

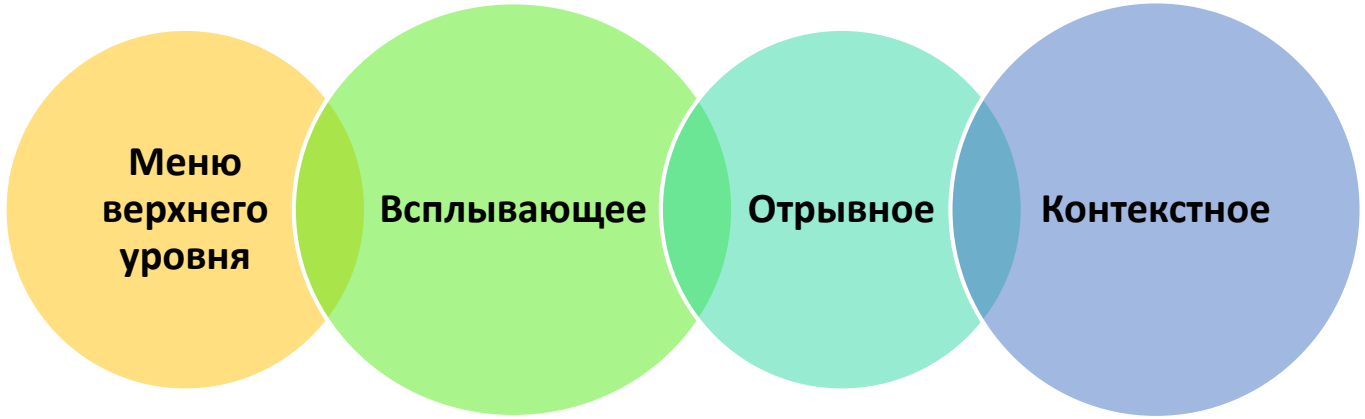
Лекция + практика

Класс «QMainWindow»

QMainWindow — это класс, который реализует главное окно, содержащее в себе типовые виджеты, необходимые большинству приложений, такие как **меню**, **секции для панелей инструментов**, **рабочую область**, **строки состояния**. В этом классе внешний вид уже подготовлен и его виджеты не нуждаются в дополнительном размещении, так как они уже находятся в нужных местах.



Виды меню



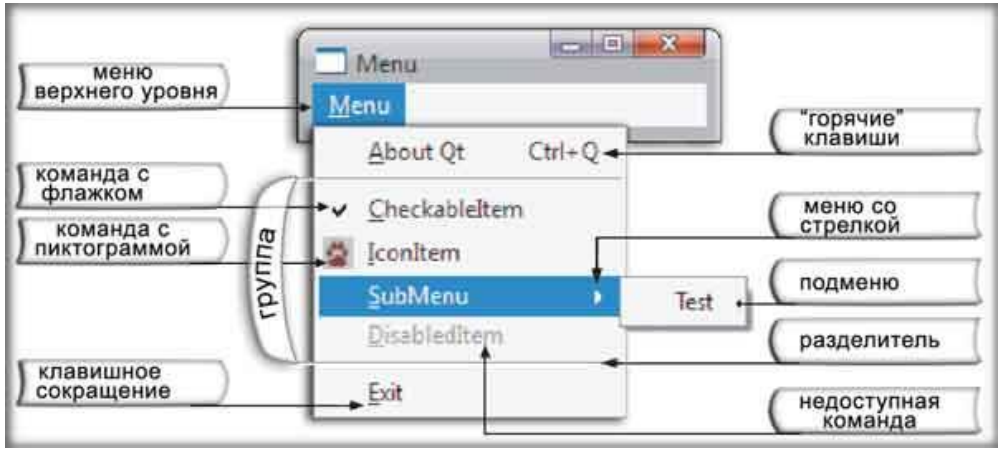
Главное меню, находится в верхней части главного окна приложения и представляет собой секцию для расположения большого количества команд, из которых пользователь может выбирать нужную ему.

В библиотеке Qt меню реализуется классом *QMenu*. Этот класс определен в заголовочном файле *QMenu*. Основное назначение класса - это размещение команд в меню. Каждая команда представляет объект действия - класс *QAction*

Все действия или сами команды меню могут быть соединены со слотами для исполнения соответствующего кода при выборе команды пользователем.

Анатомия меню

Меню верхнего уровня. Оно представляет собой меню с постоянно **видимым набором команд**, которые, в свою очередь, могут быть выбраны с помощью указателя мыши или клавиш клавиатуры (<**Alt**> и клавиши курсора). Команды *меню верхнего уровня* предназначены для отображения **выпадающих меню**, поэтому их не следует использовать для других целей, так как это может изрядно озадачить пользователя. Старайтесь логически группировать команды и объединять их в одном выпадающем меню, которое, в свою очередь, будет вызываться при выборе соответствующей команды верхнего меню. Класс ***QMenuBar*** отвечает за меню верхнего уровня и определен в заголовочном файле *QMenuBar*.

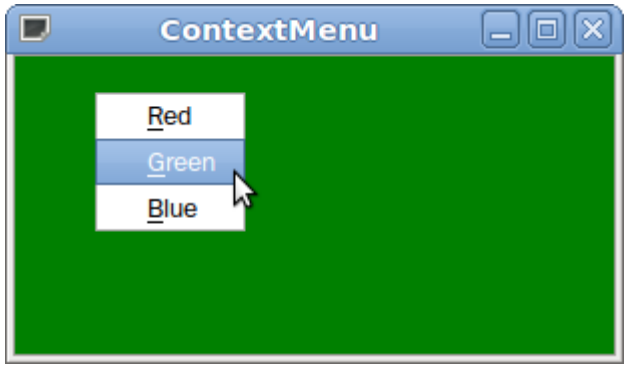


Стандартные комбинации «горячих» клавиш

Клавиша	Описание	Клавиша	Описание
<Ctrl>+<F4>	Закрыть	<Ctrl>+<Z>	Отменить
<F1>	Помощь	<Ctrl>+<X>	Вырезать
<Ctrl>+<P>	Печать	<Ctrl>+<C>	Копировать
<Ctrl>+<S>	Сохранить	<Ctrl>+<V>	Вставить

Контекстное меню

Контекстное меню — это меню, которое открывается при нажатии правой кнопки мыши. Для его реализации, также как и в случае всплывающего меню, используется класс ***QMenu***. Отличие состоит лишь в том, что это меню не присоединяется к виджету ***QMenuBar***.



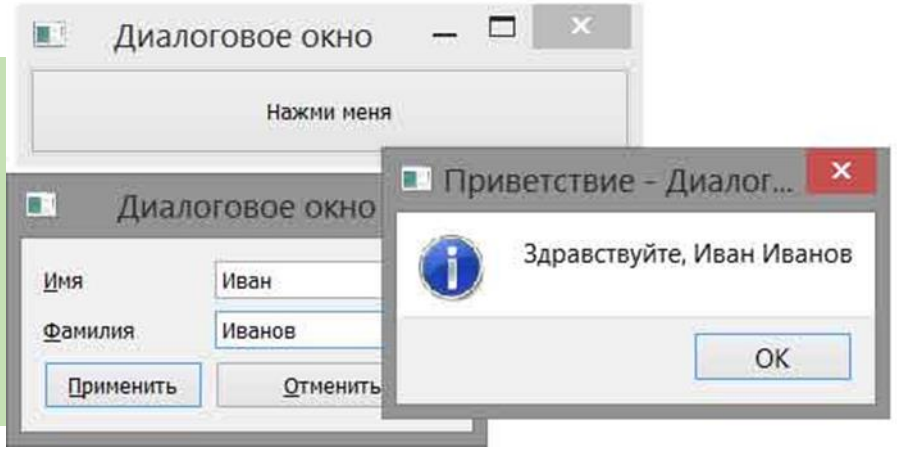
В конструкторе класса *ContextMenu* создается виджет контекстного меню — указатель mpmnu. С помощью метода *addAction()* добавляются команды меню. Метод *connect()* соединяет сигнал *меню triggered(QAction*)* со слотом *slotActivated(QAction*)*. Сигнал высылается каждый раз при выборе пользователем одной из команд меню. Этот слот получает указатель на объект действия. Благодаря тому, что наш класс *ContextMenu* унаследован от класса *QTextEdit*, мы можем устанавливать цвет фона при помощи строки в формате HTML — нужно только вызвать метод *QTetxEdit::setHtml().* Сам цвет устанавливается в соответствии с именем выбранной команды, но без символа &, для чего вызывается метод *QString::remove().* Строка с цветом записывается в переменную *strColor.*

Показ контекстного меню выполняется из метода обработки события контекстного меню *QWidget::contextMenuEvent()* и должен производиться на месте (координатах) указателя мыши, при нажатии правой кнопки. Для этого нужно передать в метод *exec()* значение объекта события контекстного меню, возвращаемое методом *globalPos().* Этот метод возвращает объекты класса *QPoint,* содержащие координаты указателя мыши относительно верхнего левого угла экрана.

Диалоговые окна. QDialog

Диалоговые окна всегда являются виджетами верхнего уровня и имеют свой заголовок. Их можно разбить на три основные категории:

- собственные;
- стандартные;
- окна сообщений.



Класс ***QDialog*** является базовым для всех диалоговых окон, представленных в классовой иерархии Qt . Для создания диалогового окна удобнее воспользоваться классом ***QDialog***, который предоставляет ряд возможностей, необходимых всем диалоговым окнам. Диалоговые окна подразделяются на две группы:

- модальные;
- немодальные.

Режим модальности и немодальности можно установить, а также определить при помощи методов ***QDialog::setModal () и QDialog::isModal ()*** соответственно. Значение *true* означает модальный режим, а *false* - немодальный.

Ничего лишнего

Не использовать прокрутку

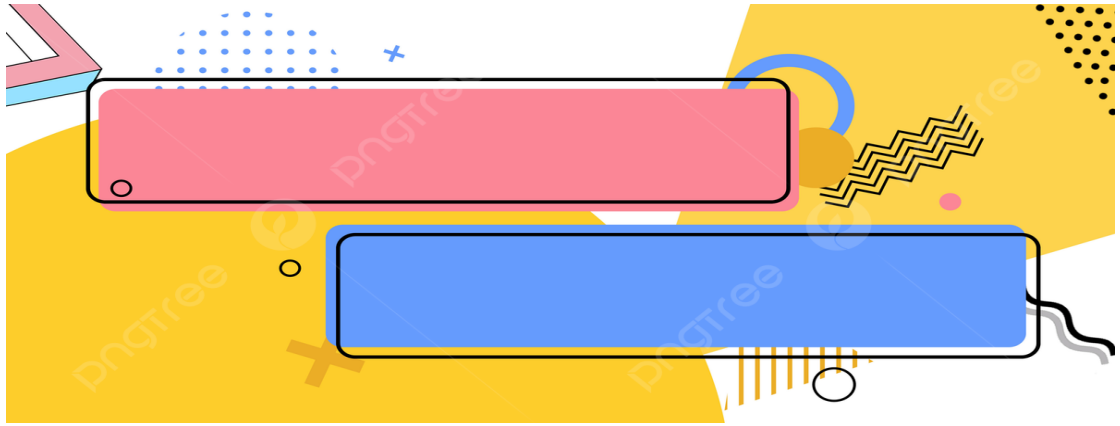
Команды меню, с диалоговыми окнами имеют надпись + «...»

Клавиатура используется с диалоговым окном наравне с мышью

Модальные и немодальные

Модальные диалоговые окна обычно используются для вывода важных сообщений. Например, иногда возникают ошибки, на которые пользователь должен отреагировать, прежде чем продолжить работать с приложением. Модальные окна прерывают работу приложения, и для продолжения его работы такое окно должно быть закрыто.

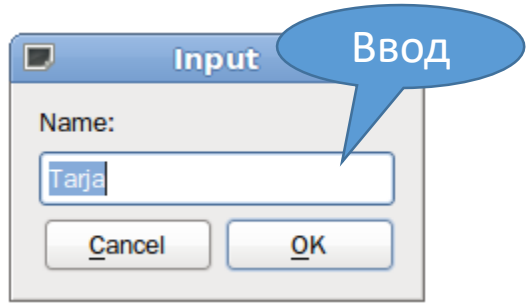
```
MyDialog* pdlg = new MyDialog(&data);
if (pdlg->exec() == QDialog::Accepted) // QDialog::Rejected
//Пользователь выбрал Accepted
// Получить данные для дальнейшего анализа и обработки
Data data = pdlg->getData();
delete pdlg;
```

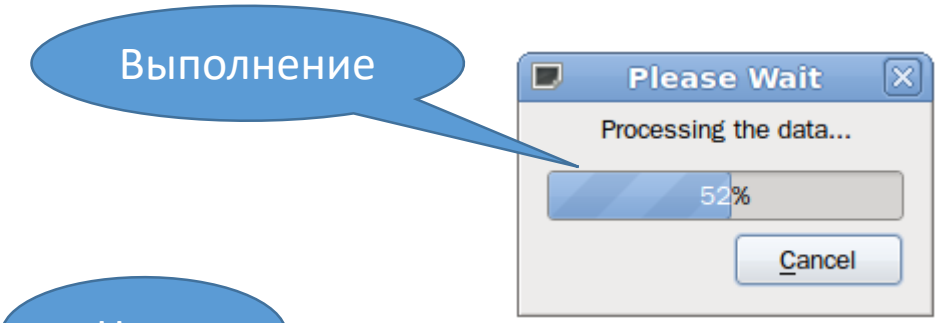


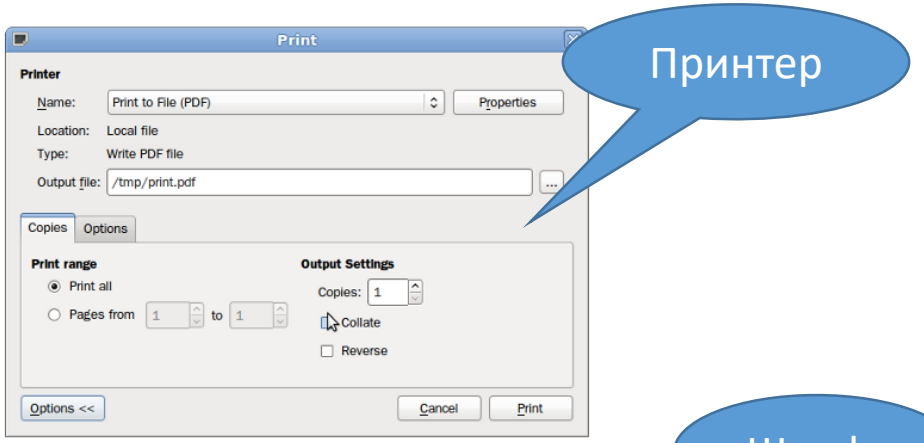
Немодальные диалоговые окна ведут себя как нормальные виджеты, не прерывая при своем появлении работу приложения. На первый взгляд может показаться, что применение немодальных диалоговых окон имеет больше смысла, чем модальных, так как в этом случае пользователь обладает большей свободой в своих действиях. Немодальное окно может быть отображено с помощью метода *show ()*, как и обычный виджет. Метод *show ()* не возвращает никаких значений и не останавливает выполнение всей программы. Метод *hide ()* позволяет сделать окно невидимым.

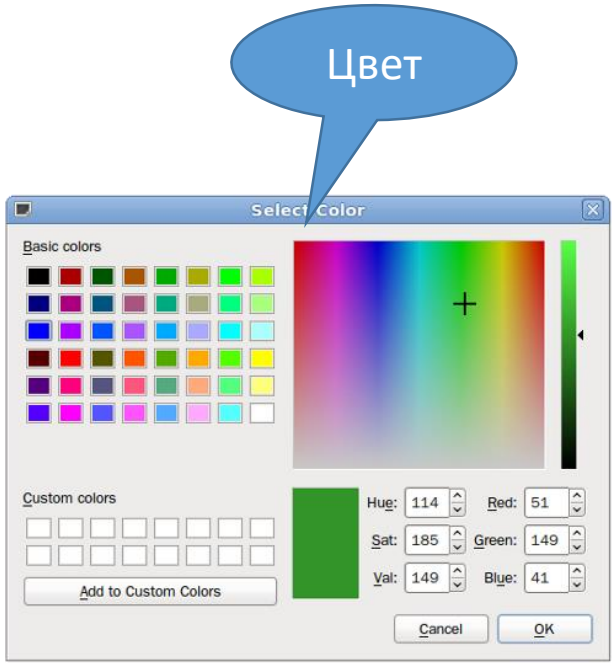
Стандартное диалоговое окно

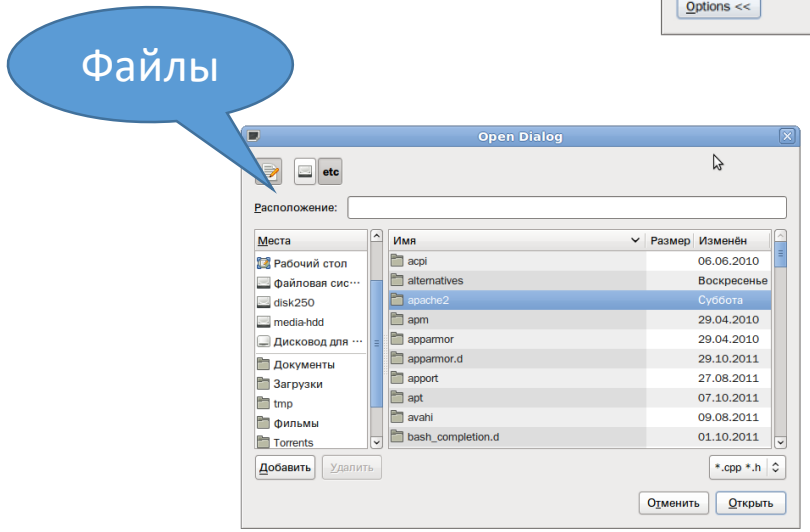
Использование стандартных окон значительно ускоряет разработку тех приложений, в которых необходимо использовать стандартные диалоговые окна выбора файлов, шрифта, цвета и т. д.

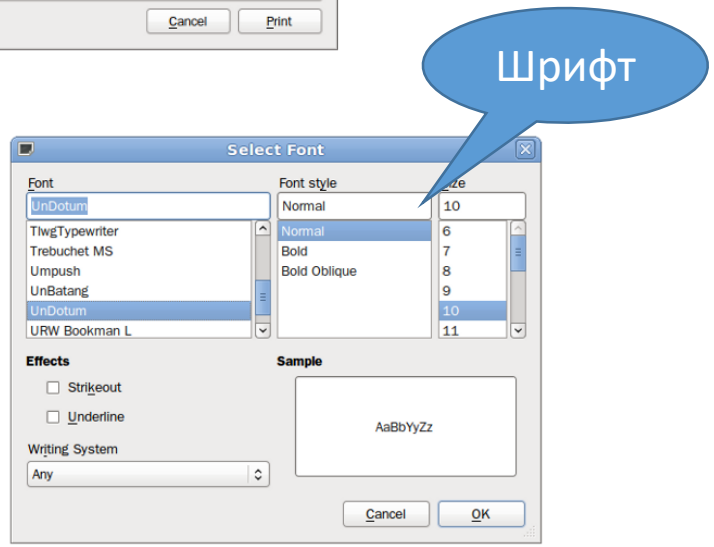












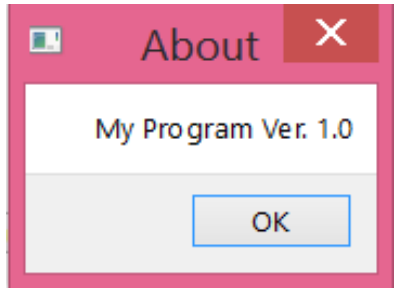
Диалоговое окно сообщений

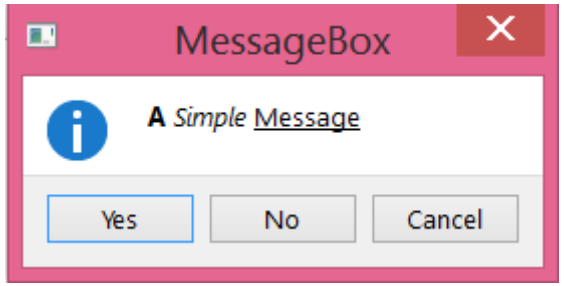
Для вывода сообщений на экран, можно воспользоваться уже готовыми окнами сообщений, предоставляемыми классом *QMessageBox*.

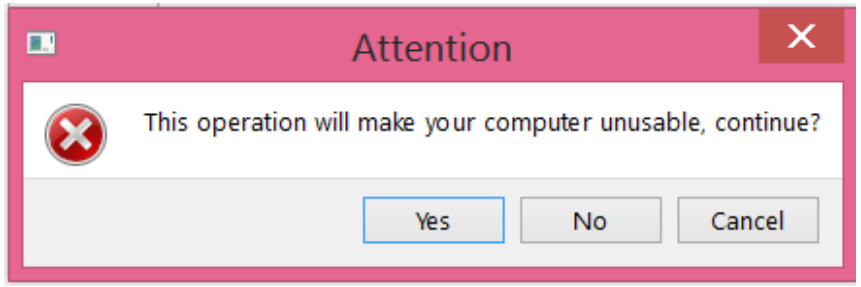
Окно сообщения отображает текстовое сообщение и ожидает реакции со стороны пользователя. Его основное назначение состоит в информировании о совершении определенного события. Все окна, предоставляемые классом *QMessageBox*, — модальные. Они могут содержать кнопки, заголовок и текст сообщения.

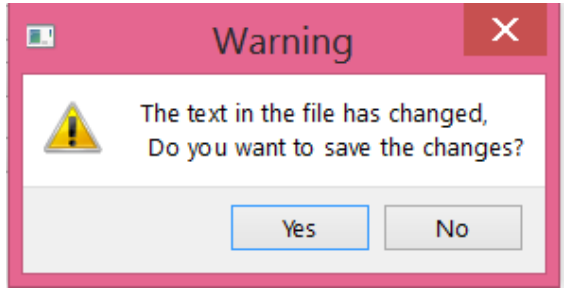
Тип окна выбирается в зависимости от обстоятельств. Окна могут содержать до трех кнопок. Тип может быть:

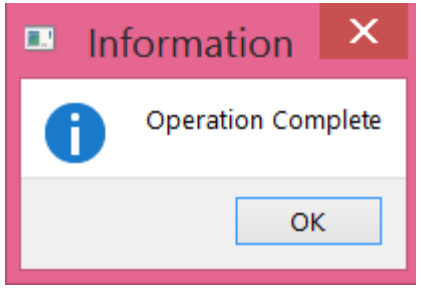
информационного,
предупреждающего,
критического.











Всплывающая подсказка

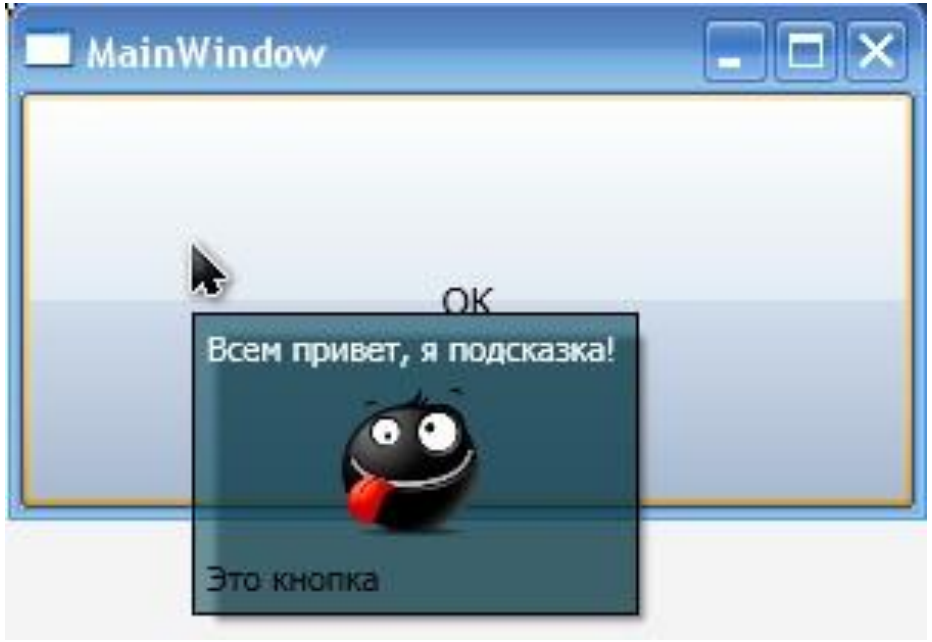
В некоторых приложениях при задержке указателя мыши над виджетом рядом с ним автоматически появляется небольшое текстовое окошко, поясняющее назначение виджета. Такое окно называется **всплывающей подсказкой** (*tooltip*) и, как правило, содержит только одну строку текста.

Чтобы установить подсказку в виджете, нужно вызвать метод *setToolTip():*

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

Чтобы удалить всплывающую подсказку, просто передайте в метод *setToolTip()* пустую строку.

Если нужно вместо окна всплывающей подсказки показать свое собственное окно, (см. проект “ToolTip”). В этой программе при задержке курсора мыши на виджете окна мы отображаем виджет надписи *QLabel* с текстом, установленным методом *setToolTip().*



Строка состояния. QStatusBar

Этот виджет располагается в нижней части главного окна и отображает, текстовые сообщения, содержащие информацию о состоянии приложения или короткую справку о командах меню или кнопках панелей инструментов. Различают следующие типы сообщений строки состояния:

- **промежуточный** - вызывается методом *showMessage ()*;
- **нормальный** - служит для отображения часто изменяющейся информации (например, для отображения позиции указателя мыши);
- **постоянный** - отображает информацию, необходимую для работы с приложением. Например, для отображения состояния клавиатуры в строку состояния могут быть внесены виджеты надписей, отображающие состояние клавиш <Caps Lock> и т.п.;

```
#pragma once
#include <QtWidgets>
class MainWindow : public QMainWindow {
Q_OBJECT
private:
	QLabel* m_plblX;
	QLabel* m_plblY;
protected:
virtual void mouseMoveEvent(QMouseEvent* pe){
	m_plblX->setText("X=" + QString().setNum(pe->x()));
	m_plblY->setText("У=" + QString().setNum(pe->y())) ;
}
```

```
public:
	MainWindow(QWidget* pwgt = 0)
		:QMainWindow(pwgt){
		setMouseTracking(true);
		m_plblX = new QLabel (this);
		m_plblY = new QLabel (this);
		statusBar()->addWidget(m_plblY);
		statusBar()->addWidget(m_plblX) ;
	}
};
```

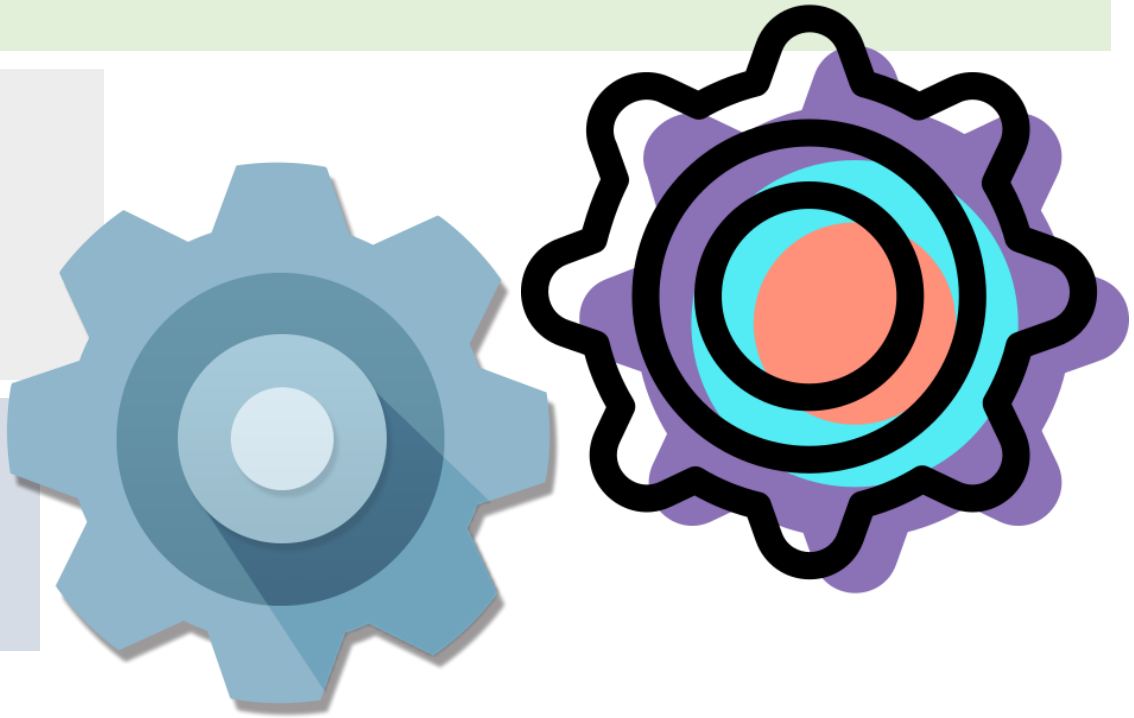
Сохранение настроек приложения.

Возможность изменения и сохранения настроек приложения очень удобна для «приспособления» интерфейса программы под конкретного пользователя. На самом деле, пользователю будет очень приятно, если при запуске приложения будут восстановлены настройки, сделанные им во время предыдущего сеанса работы: приложение будет находиться на том же месте, иметь те же размеры и выглядеть так, как удобно этому пользователю. Необходимо также сохранять названия и пути последних документов, чтобы пользователь мог быстро их выбрать.

Данные настроек приложения организованы в совокупность ключей и значений. *Ключ* (*key*) представляет собой строковое значение - имя, с помощью которого можно программно получать и устанавливать адресуемое ключом *значение* (*key value*).

Приложение **ОС Windows**, сохраняет данные в системном реестре, например в ветках реестра
HKEY _ LOCAL _ MACHINE\Software
или
HKEY _ CURRENT_ USER\Software.

В **Linux** для хранения данных задействованы каталоги
$HOME/.qt
или
$QTDIR/etc.



Сохранение настроек приложения.

В Qt для работы с настройками служит класс *QSettings*. При его создании в конструктор можно передать имя компании и название программы. Например:
QSettings settings("BHV","MyPrograrn");

Для записи настроек приложения служит метод *setValue ():*

```
settings.setValue("/Settings/StringKey", "String Value");
settings.setValue("/Settings/IntegerKey", 213);
settings.setValue("/Settings/BooleanKey", true);
```

Для получения данных нужно воспользоваться методом *value (),* который возвращает значения типа *QVariant*. Благодаря этому класс *QSettings* предоставляет методы для чтения разных типов - полученные значения типа *QVariant* необходимо лишь привести к нужному вам типу.

```
QString str = settings.value("/Settings/StringKey", "") .toString();
int n       = settings.value("/Settings/IntegerKey", 0) .toInt();
bool b      = settings.value("/Settings/BooleanKey", false) .toBool();
```

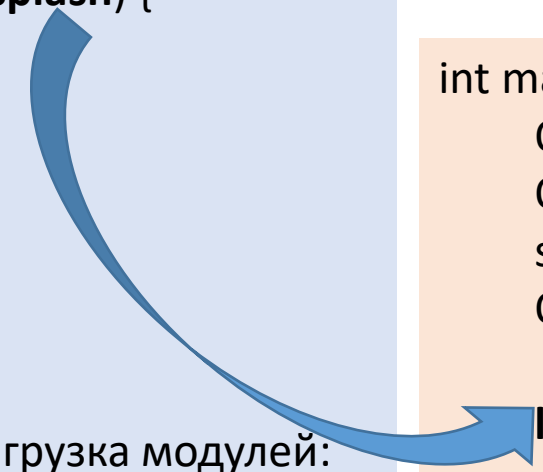
Для удаления ключей и их значений нужно передать имя ключа в метод *remove()* класса *Qsettings.*

```
settings.remove("/Settings/StringKey");
```

Окно заставки

При запуске многие приложения показывают так называемое окно заставки (**Splash Screen**). Это окно отображается на время, необходимое для инициализации приложения, и информирует о ходе его запуска . Зачастую такое окно используют для маскировки длительного процесса старта программы .

```
#include <QtWidgets>
void loadModules(QSplashScreen* psplash) {
        QTime time;
        time.start();
        for (int i = 0; i < 100; ) {
                if (time.elapsed() > 40) { .
                        time.start();
                        ++i;
                }
                psplash->showMessage(“Загрузка модулей:
                        + QString: :number(i) + "%",
                        Qt::AlignHCenter | Qt::AlignBottom,
                        Qt::black );
        qApp->processEvents();
        }
}
```



```
int main (int argc, char** argv) {
        QApplication app(argc, argv);
        QSplashScreen splash(QPixmap(":/splash.png"));
        splash. show ( ) ;
        QLabel lbl("<H1><CENTER>Моё приложение<BR>"
                        «Готово!</CENTER></H1>" );
        loadModules(&splash);
        splash.finish(&lbl);
        lbl.resize(250, 250);
        lbl.show();
        return app.exec();
}
```

Практика

- Проект «Menu_Dialog_App»
- Проект «ToolTip»;