Элементы управления GUI

Лекция + практика

Виджеты (widgets)

Виджеты (*widgets*)- это «строительный материал» для его создания. Виджет - это не просто область, отображаемая на экране, это компонент, способный выполнять различные действия (реагировать на сигналы и события или отправлять сигналы другим виджетам).
Qt предоставляет богатую библиотеку виджетов: от кнопок меню до диалоговых окон, необходимых для создания профессиональных приложений. Но можно создать свои виджеты, наследуя классы уже существующих.

Класс **QWidget** является фундаментальным для всех классов виджетов. Его интерфейс содержит 254 метода, 53 свойства и массу определений, необходимых каждому из виджетов (изменения размеров, местоположения, обработки событий и др). Класс QWidget, унаследован от класса QObject, а значит, может использовать *механизм сигналов/слотов* и механизм *объектной иерархии*. Благодаря этому виджеты могут иметь потомков, которые отображаются внутри предка. То есть каждый виджет может служить контейнером для других виджетов. Фактически в Qt нет разделения между элементами управления и контейнерами. Виджеты в контейнерах могут выступать в роли контейнеров для других виджетов, и так до бесконечности. Например, диалоговое окно содержит кнопки Ok и Cancel - следовательно, оно является контейнером. Это удобно еще и потому, что если виджет-предок станет недоступным или невидимым, то виджеты-потомки автоматически примут его состояние.

Виджеты (widgets)

- Виджеты без предка называются **виджетами верхнего уровня** (*top-level widgets*) и имеют свое собственное окно. Все виджеты без исключения могут быть виджетами верхнего уровня.
- Позиция виджетов-потомков внутри виджета-предка может изменяться методом **setGeometry( )** вручную или автоматически, с помощью специальных классов компоновки (*layouts*).
- Для отображения виджета на экране вызывается метод **show(),** а для скрытия - метод **hide()** .

Если конструктор вызывается без аргументов, то созданный виджет станет виджетом верхнего уровня. Второй параметр: **Qt::WindowFlags** служит для задания свойств окна, и с его помощью можно управлять внешним видом окна и режимом отображения (чтобы окно не перекрывалось другими окнами и т. д.). Чтобы изменить внешний вид окна, необходимо во втором параметре конструктора передать значения модификаторов, объединенные с типом окна побитовой операцией ИЛИ, обозначенной символом “**|**” . Аналогичного результата можно добиться вызовом метода **setWindowFlags ()**

```
wgt.setWindowFlags(Qt::Window | Qt::WindowTitleHint |
                   Qt::WindowStaysOnTopHint);
```

Конструктор класса QWidget и унаследованных классов

QWidget(QWidget* pwgt = 0, Qt::WindowFlags f = 0)

Размеры и координаты виджета

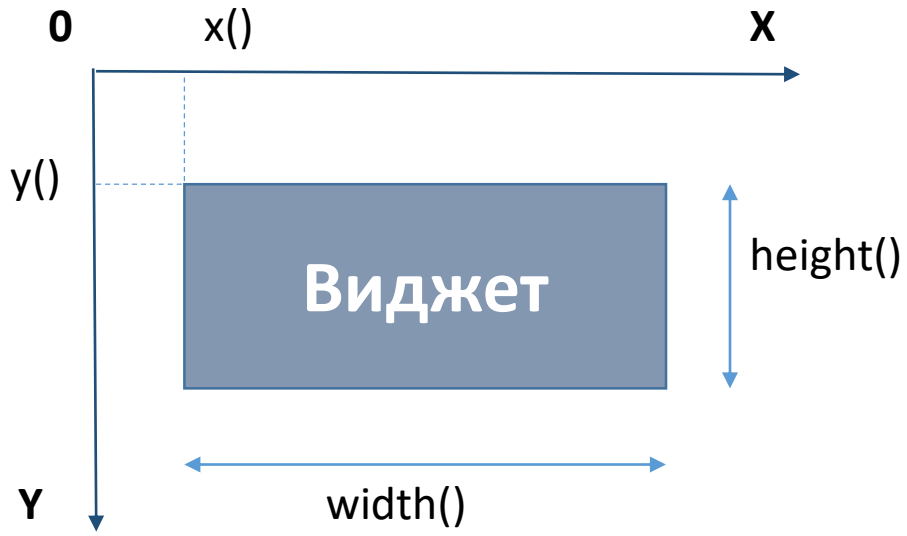
Виджет представляет собой прямоугольную область. Методы **size (), height ()** и **width ()** возвращают размеры виджета. При этом, если вызовы height() и width () вернут значения типа **int**, то вызов метода size () вернет объект класса **QSize**, хранящий ширину и высоту виджета.

Методы **x()** , **y()** и **pos()** служат для определения координат виджета. Первые два метода возвращают целые значения координат по осям X и У, а метод pos() - объект класса **QPoint**, хранящий обе координаты.
Метод **geometry()** возвращает объект класса **QRect,** описывающий положение и размеры виджета.
Положение виджета можно изменить методом **move()**, а его размеры - методом **resize()**:

```
QWidget wgt;
QWidget* pwgt = new Qwidget(&wgt);
pwgt->move(5, 5);
pwgt->resize(260, 330);
```

Одновременно изменить и положение, и размеры виджета можно, вызвав метод **setGeometry ()** .
Первый параметр этого метода задает координату левого верхнего угла
виджета по оси X, второй - по оси У, третий задает ширину, а четвертый – высоту:

```
pwgt->setGeometry(5, 5, 260, 330) ;
```



Установка фона виджета

Создадим виджет верхнего уровня wgt, и передадим двум другим виджетам (указатели pwgt1 и pwgt2) в качестве предка.

Виджету можно **задать фон**. Это может быть цвет или растровое изображение. Для заполнения сплошным цветом или растровым изображением необходимо сначала создать объект палитры, а затем вызовом метода **setPalette()** установить его в виджете. Виджет имеет важное свойство **autoFillBackground,** которое должно быть установлено = true, тем самым вызаставите виджет заполнять фон автоматически, что сделает его видимым. Например:

wgt.setAutoFillBackground(true);


```
#include <QtWidgets>
 int main(int argc, char ** argv) {
        QApplication app (argc, argv);
        QWidget wgt;
        QWidget* pwgt1 = new QWidget (&wgt);
        QPalette pal1;
        pal1.setColor(pwgt1->backgroundRole() ,
        Qt::blue); pwgt1->setPalette(pal1);
        pwgt1->resize(100, 100);
        pwgt1->move(25, 25);
        pwgt1->setAutoFillBackground (true) ;
        QWidget* pwgt2 = new QWidget (&wgt) ;
        QPalette pal2 ;
        pal2. setBrush (pwgt2->backgroundRole(),
        QBrush (QPixmap ("aaa. jpg"))) ;
        pwgt2->setPalette(pal2);
        pwgt2->resize(100, 100);
        pwgt2->move(75, 75);
        pwgt2->setAutoFillBackground(true);
        wgt.resize(200, 200);
        wgt . show() ;
return app.exec();
}
```

Виджет видовой прокрутки QScrollArea

Базовый класс для видовой прокрутки **QAbstractScrollArea** унаследован от класса **QFrame** и представляет собой окно для просмотра только части информации. Сам виджет видовой прокрутки реализует класс **QScrollArea**.
Этот виджет может размещать виджеты потомков, а если хотя бы один из них выйдет за границы окна просмотра, то автоматически появляются вертикальная и/или горизонтальная полосы прокрутки. С их помощью можно перемещать части виджета в область просмотра. Если вы хотите, чтобы полосы прокрутки были видны всегда, то нужно передать в методы управления поведением полос значение Qt::ScrollBarAlwaysOn. Например:

```
QScrollArea sa;
sa.setHorizontalScrollBarPolicy(Qt: :ScrollBarAlwaysOn);
sa.setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOn);
```

Виджет видовой прокрутки является совокупностью сразу нескольких виджетов, работающих вместе.

Осуществить доступ к виджету области просмотра можно посредством метода QAbstractScrollArea ::viewport ().

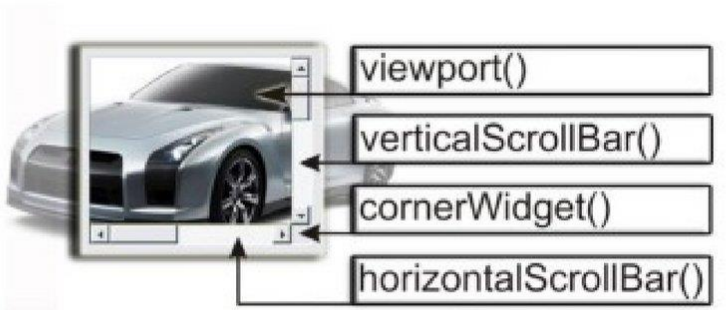
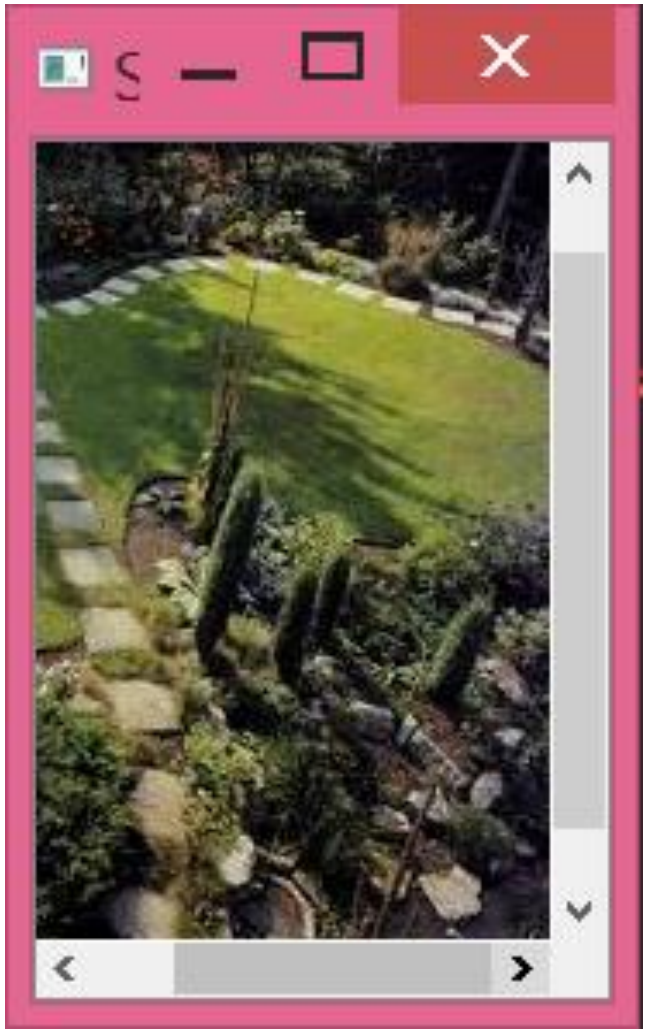


Рис. 5.6. Структура виджета видовой прокрутки

П. Виджет видовой прокрутки QScrollArea

```
#include <QtWidgets>

int main(int argc, char** argv) {
        QApplication a(argc, argv);
        QScrollArea sa;
        QWidget* pwgt = new QWidget;
        QPixmap pix (“bbb.jpg") ;
        QPalette pal;
        pal.setBrush(pwgt->backgroundRole(), QBrush(pix) );
        pwgt->setPalette(pal);
        pwgt->setAutoFillBackground(true);
        pwgt->setFixedSize(pix.width(), pix.height() ) ;
        sa.setWidget(pwgt);
        sa.resize(350, 150);
        sa.show();
return a.exec();
}
```



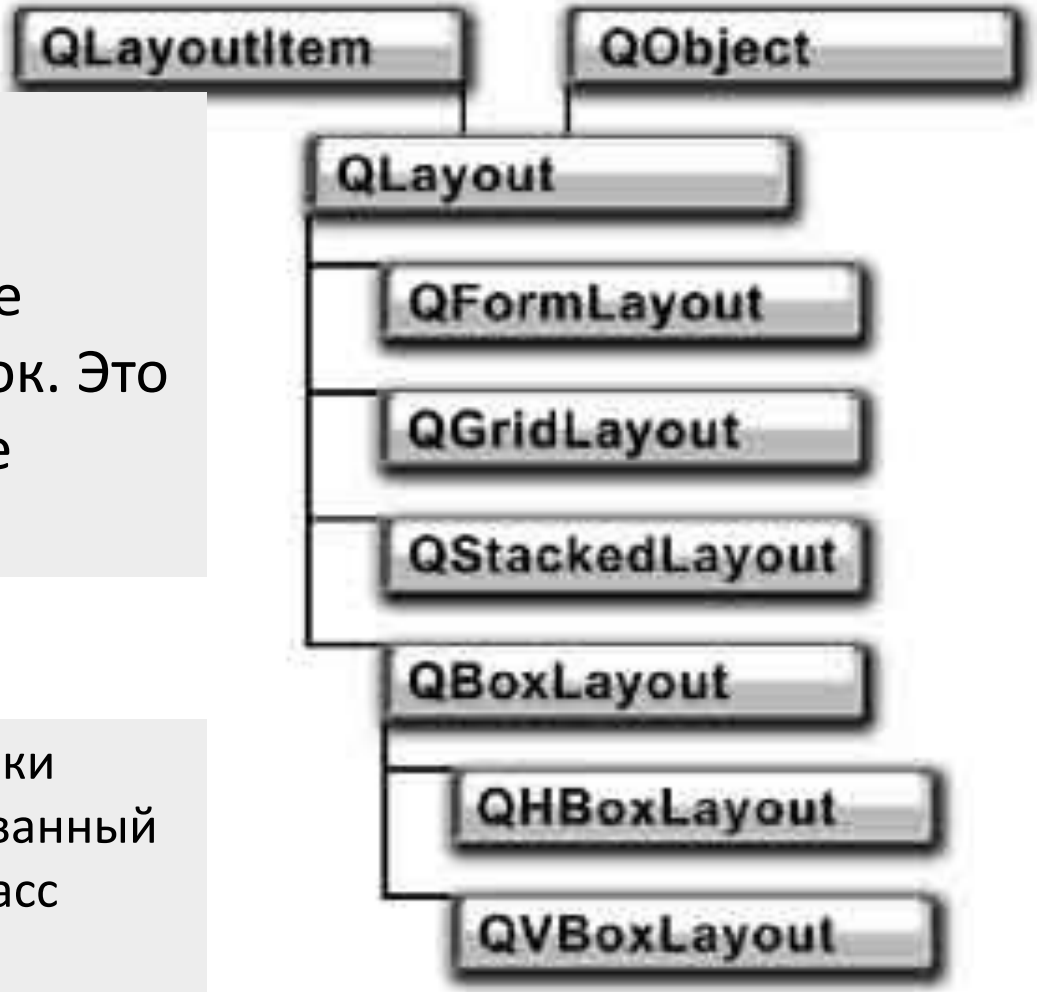
Управление автоматическим размещением элементов

Классы **компоновки** виджетов (*Layouts*) по сути, являются контейнерами, которые после изменения размеров окна автоматически приводят в соответствие размеры и координаты виджетов, находящихся в нем. Хотя они ничего не добавляют к функциональной части самой программы, тем не менее, они очень важны для внешнего вида окон приложения. Компоновка определяет расположение различных виджетов относительно друг друга.

Qt предоставляет так называемые **менеджеры компоновки**, позволяющие организовать размещение виджетов на поверхности другого виджета. Основу их работы определяет возможность каждого виджета сообщать о том, сколько ему необходимо места, может ли он быть растянут по вертикали и/или горизонтали и т. п.

Менеджеры компоновки

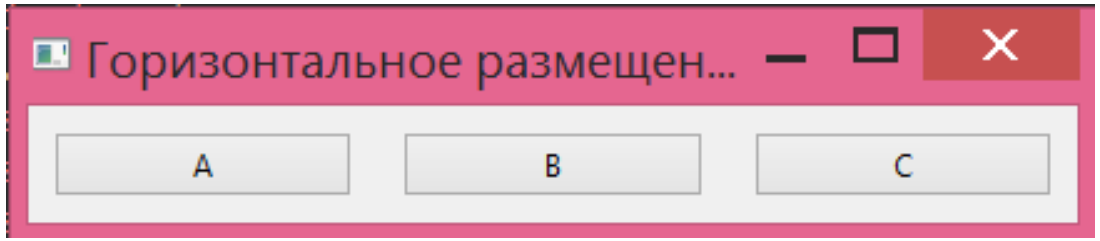
Менеджеры компоновки предоставляют возможности для *горизонтального, вертикального и табличного размещения* не только виджетов, но и встроенных компоновок. Это позволяет конструировать довольно сложные размещения.



Фундаментом для всей группы менеджеров компоновки является класс **QLayout** - абстрактный класс, унаследованный сразу от двух классов: **QObject** и **QLayoutItem** . Этот класс определен в заголовочном файле **QLayout**.

П. Горизонтальное размещение QHBoxLayout

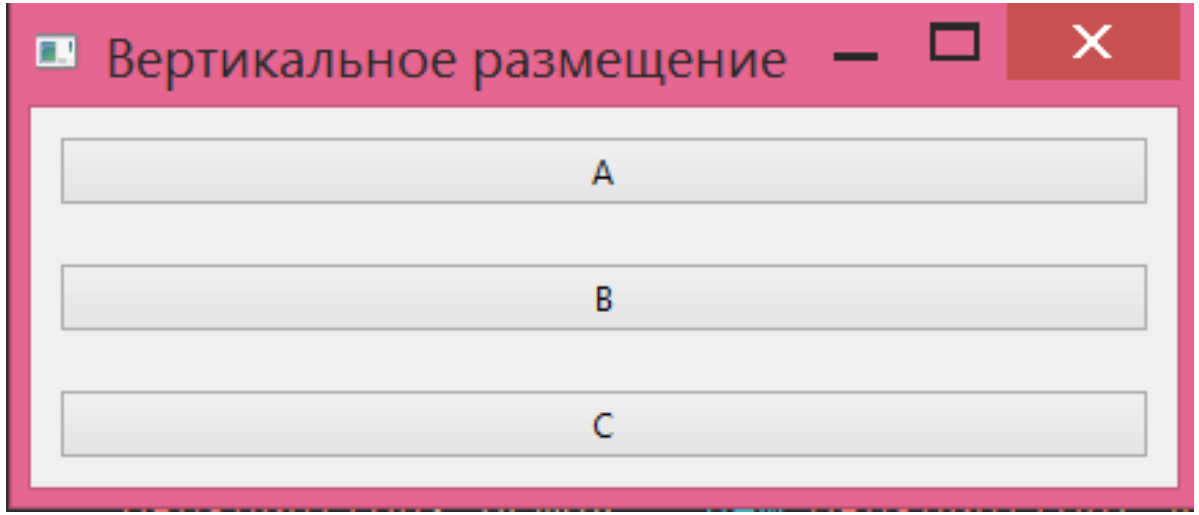
Объекты класса **QHBoxLayout** упорядочивают все виджеты только в горизонтальном порядке - слева направо. Его применение аналогично использованию класса **QBoxLayout**, но передавать в конструктор дополнительный параметр, задающий горизонтальный порядок размещения, не нужно. Окно программы, которая упорядочивает виджеты при помощи объекта класса **QHBoxLayout**.




```
#include <QtWidgets>
int main(int argc, char** argv){
        QApplication a(argc, argv);
        QWidget wgt;
        QPushButton* pcmdA = new QPushButton("A");
        QPushButton* pcmdB = new QPushButton("B");
        QPushButton* pcmdC = new QPushButton("C");
        //Layout setup
        QHBoxLayout* phbxLayout = new QHBoxLayout;
        phbxLayout ->setContentsMargins(10, 10, 10, 10);
        phbxLayout ->setSpacing(20);
        phbxLayout ->addWidget(pcmdA);
        phbxLayout ->addWidget(pcmdB);
        phbxLayout ->addWidget(pcmdC);
        wgt.setLayout(phbxLayout);
        wgt.show();
return a.exec();
}
```

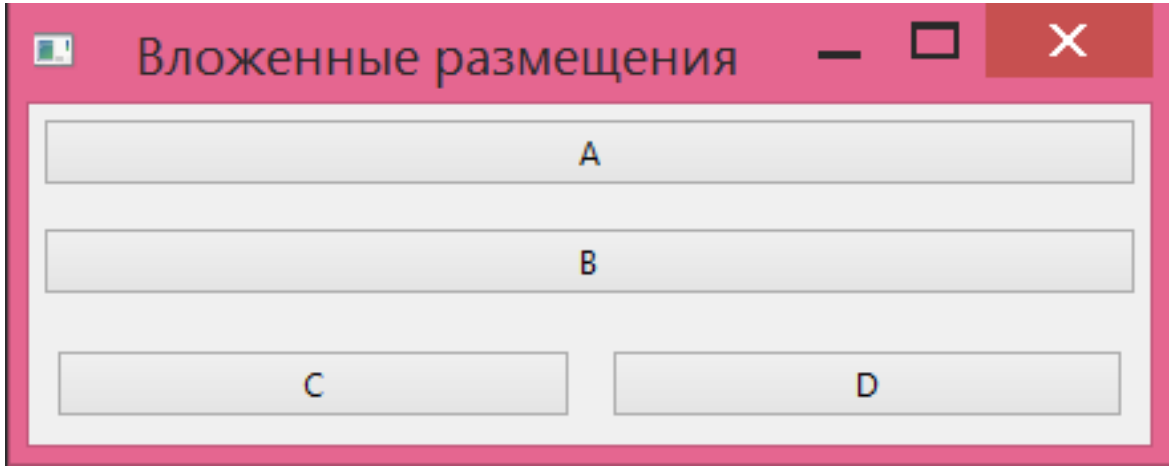
П. Вертикальное размещение QVBoxLayout

Компоновка **QVBoxLayout** унаследована от **QBoxLayout** и упорядочивает все виджеты только по вертикали - сверху вниз. В остальном она ничем не отличается от классов QBoxLayout и QHBoxLayout.



П. Вложенные размещения

Размещая одну компоновку внутри другой, можно создавать размещения практически любой сложности. Для организации вложенных размещений существует метод **addLayout()**, в который вторым параметром передается фактор растяжения для добавляемой компоновки.

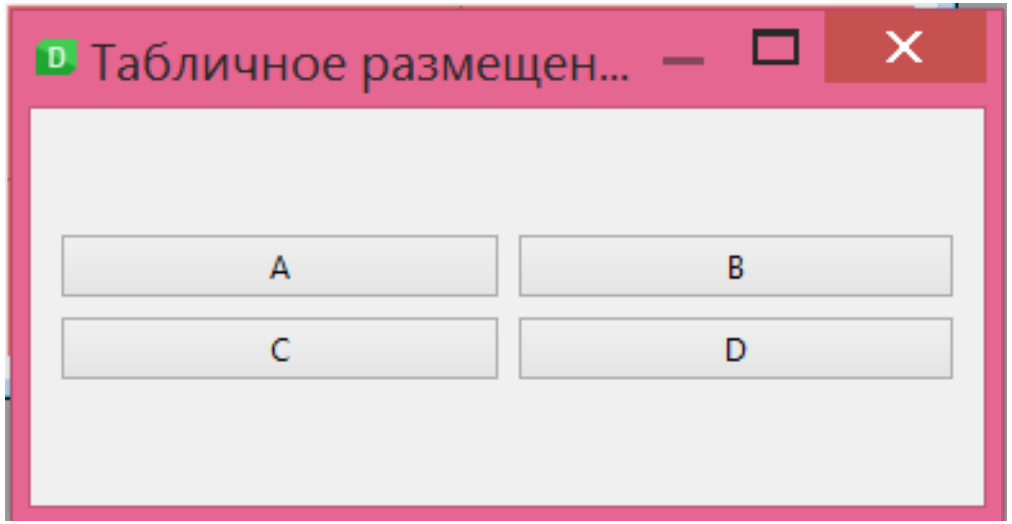



```
        #include <QtWidgets>
int main(int argc, char** argv){
…….
        QVBoxLayout* pvbxLayout = new QVBoxLayout;
        QHBoxLayout* phbxLayout = new QHBoxLayout;
        phbxLayout->setContensMargins(5, 5, 5, 5);
        phbxLayout->setSpacing(15);
        phbxLayout->addWidget(pcmdC);
        phbxLayout->addWidget(pcmdD);
        pvbxLayout->setContentsMargins(5, 5, 5, 5);
        pvbxLayout->setSpacing(15);
        pvbxLayout->addWidget(pcmdA);
        pvbxLayout->addWidget(pcmdB);
        pvbxLayout->addLayout(phbxLayout);
        wgt.setLayout(pvbxLayout);
        wgt.show() ;
return app.exec();
}
```

П. Табличное размещение QGridLayout

```
#include <Qt Widgets>
int main(int argc, char** argv)
        QApplication app(argc, argv);
        QWidget wgt;
        QPushButton* pcmdA= new QPushButton("A");
        QPushButton* pcmdB= new QPushButton("B");
        QPushButton* pcmdC =new QPushButton("C") ;
        QPushButton* pcmdD= new QPushButton("D") ;
        QGridLayout* pgrdLayout = new QGridLayout;
        pgrdLayout->setContentsMargins(5, 5, 5, 5);
        pgrdLayout->setSpacing(15);
        pgrdLayout->addWidget(pcmdA, 0, 0);
        pgrdLayout->addWidget(pcmdB, 0, 1);
        pgrdLayout->addWidget(pcmdC, 1, 0);
        pgrdLayout->addWidget(pcmdD, 1, 1) ;
        wgt.setLayout(pgrdLayout);
        wgt.show();
return app.exec();
}
```

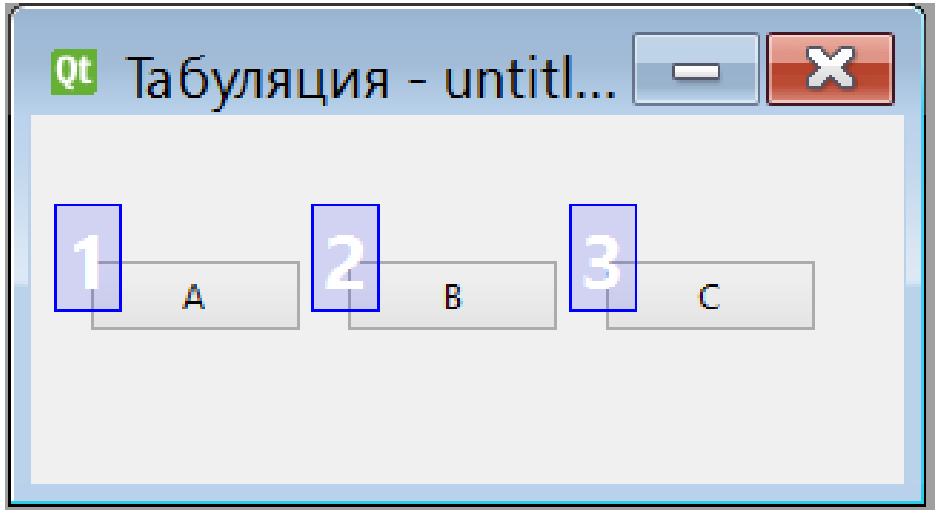
Для табличного размещения служит класс **QGridLayout**, с помощью которого можно быстро создавать сложные по структуре размещения. Таблица состоит из ячеек, позиции которых задаются строками и столбцами.



Порядок следования табулятора

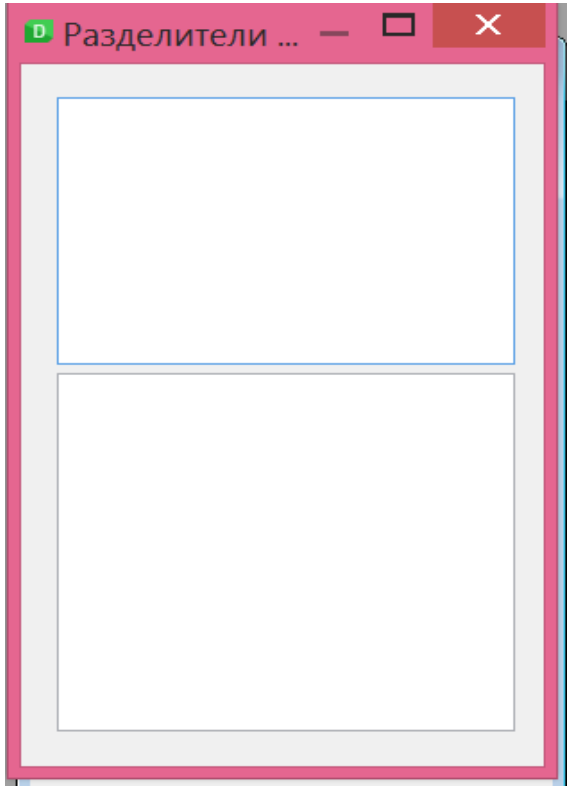
Пользователь может взаимодействовать с виджетами при помощи мыши и клавиатуры .
В последнем случае для выбора нужного виджета используется клавиша табуляции - <Tab>, при нажатии которой происходит переход фокуса согласно установленному порядку от одного виджета к другому. Иногда возникает необходимость в изменении этого порядка, который по умолчанию соответствует очередности установки дочерних виджетов в виджете предка. На рис цифрами изображен порядок смены фокуса с помощью табулятора: при появлении диалогового окна с тремя кнопками фокус будет установлен на кнопке **A** и после нажатия на клавишу табуляции он перейдет на кнопку **B**, а затем - на кнопку **C**.

QWidget::setTabOrder(A, B);
QWidget::setTabOrder(B, C);



П. Разделители QSplitter

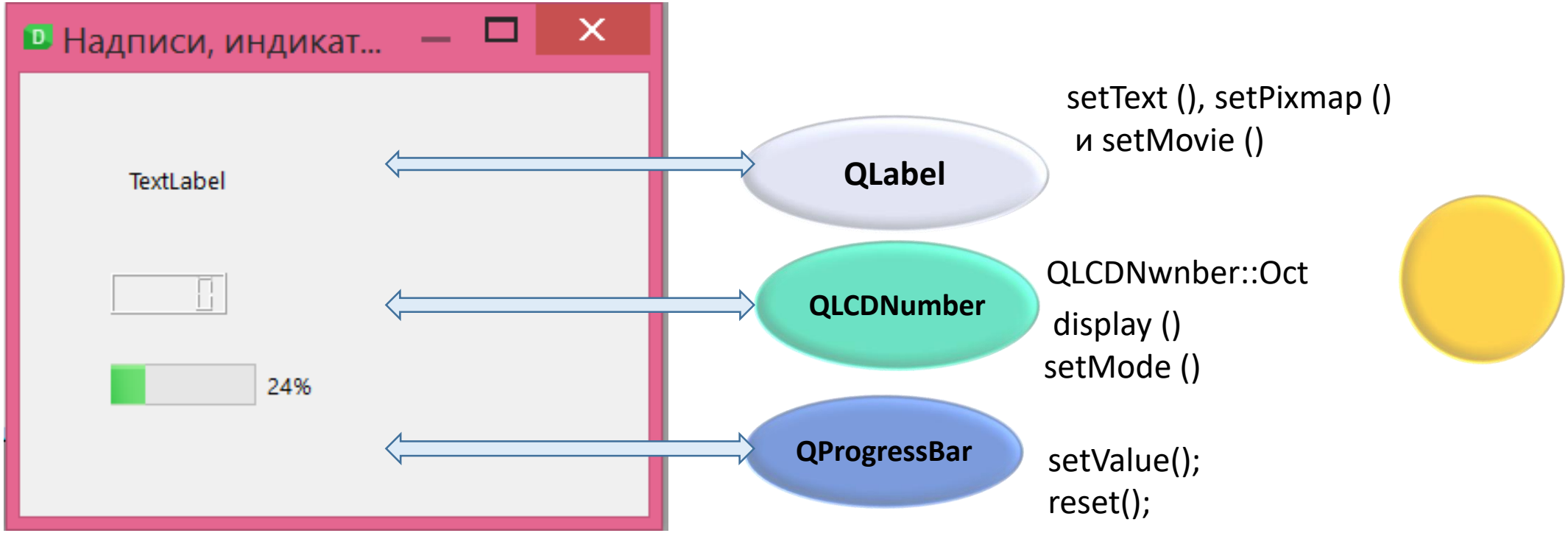
Разделители придуманы для одновременного просмотра различных частей текстовых или графических объектов. В некоторых случаях применение разделителя более предпочтительно, чем размещение с помощью классов компоновки, так как появляется возможность изменения размеров виджетов.



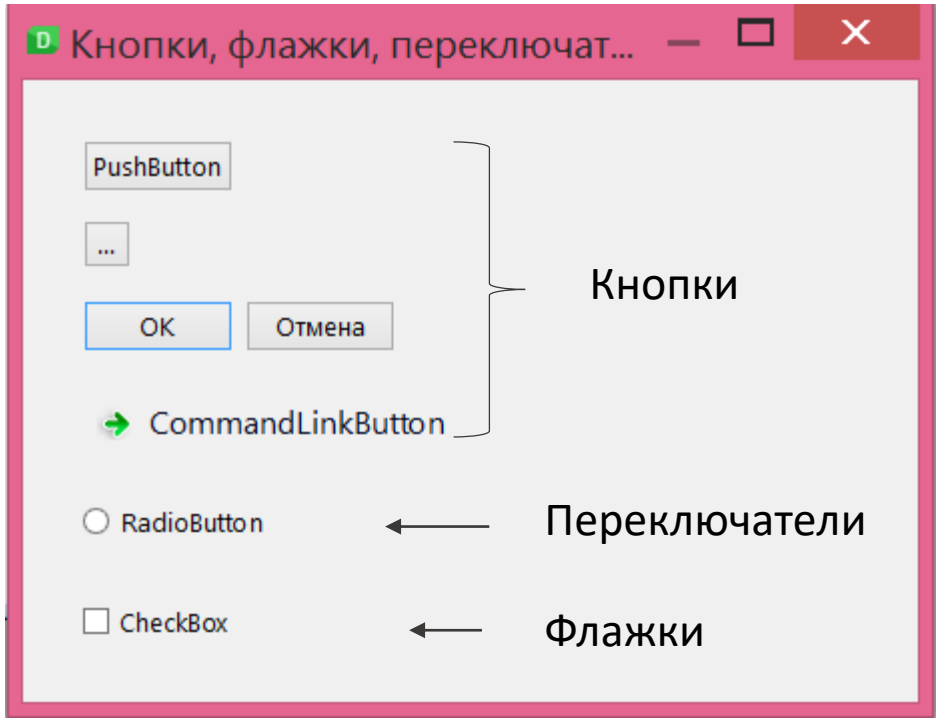

```
#include <QtWidgets>
int main(int argc, char** argv){
        QApplication app(argc, argv) ;
        QSplitter spl (Qt::Vertical);
        QTextEdit* ptxt1 = new QTextEdit;
        QTextEdit* ptxt2 = new QTextEdit;
        spl.addWidget(ptxt1);
        spl.addWidget(ptxt2);
        ptxtl->setPlainText("Line1\n"
        "Line2\n"
        "Line3\n") ;
        ptxt2->setPlainText(ptxt1-
        >toPlainText());
        spl.resize(200, 220);
        spl.show();
return app.exec();
}
```

Надписи, индикаторы

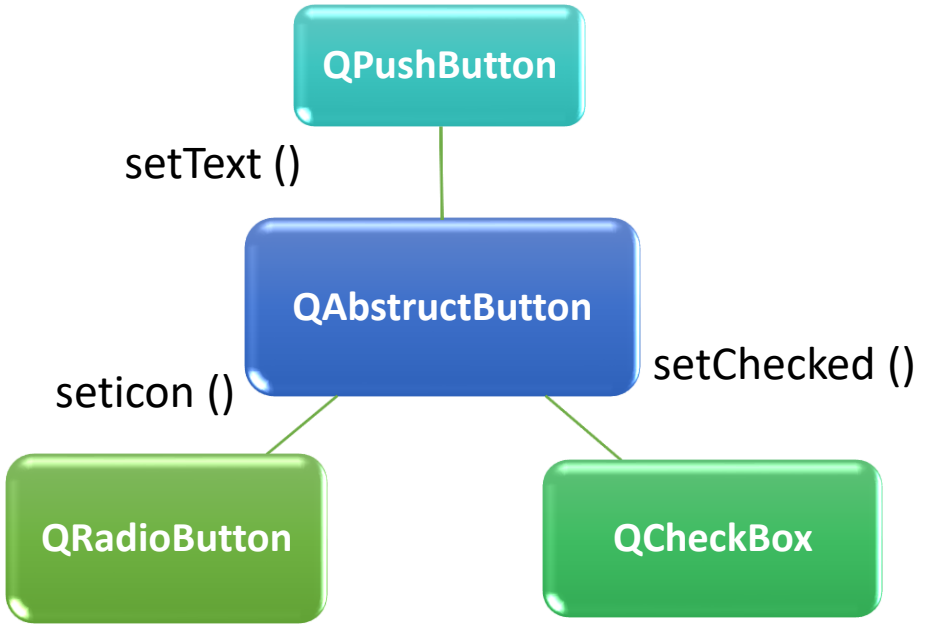
QLabel, QLCDNumber, QProgressBar



Кнопки, флажки, переключатели

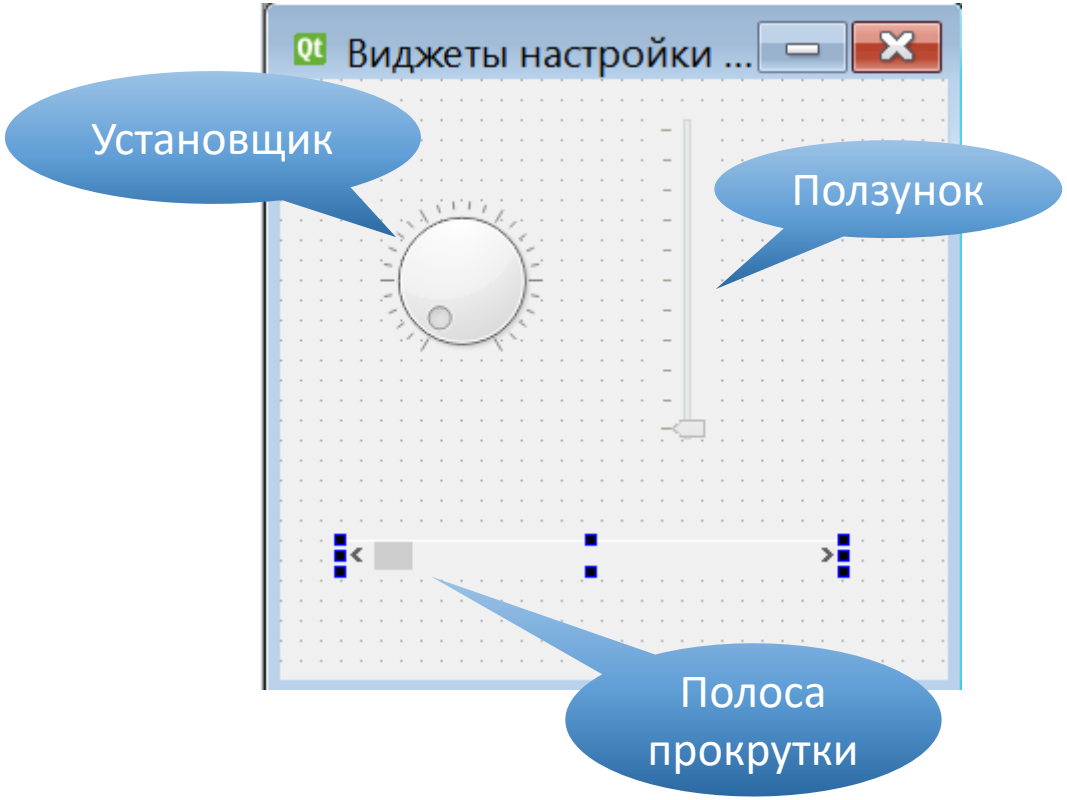


Класс **QAbstractButton** - базовый для всех кнопок. В приложениях применяются три основных вида кнопок: нажимающиеся кнопки (**QPushButton**), которые обычно называют просто кнопками, флажки (**QCheckBox**) и переключатели (**QRadioButton**). В классе **QAbstractButton** реализованы методы и возможности, присущие всем кнопкам.



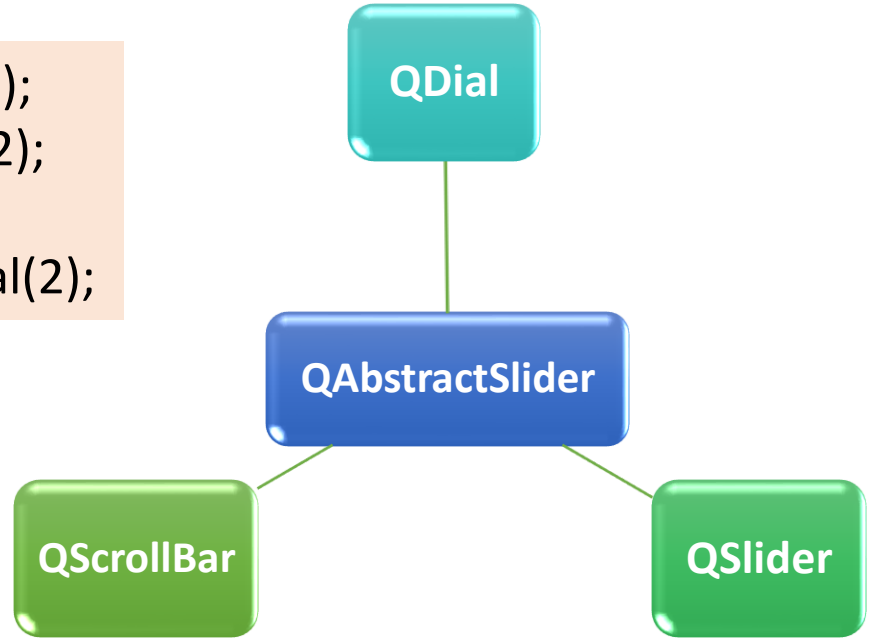
- clicked () - отправляется при щелчке кнопкой мыши;
- pressed () - отправляется при нажатии на кнопку мыши;
- released () - отправляется при отпускании кнопки мыши;
- toggled () - отправляется при изменении состояния кнопки, имеющей статус выключателя.

Элементы настройки



Группа виджетов, относимых к элементам настройки, используется, как правило, для установки значений, не требующих большой точности, - например, настройки громкости звука, скорости движения курсора мыши, скроллинга содержимого окна и других подобных действий

```
psld->setRange(0, 9);
psld->setPageStep(2);
psld->setValue(3);
psld->setTickInterval(2);
```



SIGNAL(valueChanged(int) -> SLOT(setValue(int))

П. Работа с элементами управления

- Проект «SettingsWidgets»