

Explicit

Explicit - запрещает автоматическое создание конвертирующего конструктора.

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    ...
};
```

```
QWidget *test;

//нельзя
MainWindow w = test;

//можно
MainWindow w(test);
```

QGraphicsScene

- Поверхность для управления 2D графическими элементами
- Контейнер для **QGraphicsItems**.
- используется вместе с **QGraphicsView** для отображения графических объектов:
 - линии
 - прямоугольники
 - текст
 - ...
- QGraphicsScene не имеет собственного графического представления; он только управляет элементами. Для отображения сцены вам необходимо создать виджет **QGraphicsView**.

QGraphicsScene

Чтобы добавить элементы на сцену:

1. создать объект **QGraphicsScene**.
2. добавить объекты:
 - a. существующие объекты **QGraphicsItem** вызвав **addItem()**
 - b. использовать вспомогательных функций **addEllipse()**, **addLine()**, **addPath()**, **addPixmap()**, **addPolygon()**, **addRect()** **addText()**, которые возвращают указатель на добавленный элемент.
 - c. Отобразить сцену с помощью **QGraphicsView**.
3. При изменении сцены выбрасывается сигнал **changed()**

QGraphicsItem

- Базовый класс для всех графических элементов
- Основа для написания своих собственных элементов
- Включает определение геометрии элемента, обнаружение столкновений, реализацию его отрисовки и взаимодействие элементов с помощью указателей

QGraphicsItem

- Чтобы написать свой собственный графический элемент сначала вам надо создать подкласс QGraphicsItem
 - реализовать функцию **boundingRect()**, которая возвращает приблизительную площадь занимаемую элементом
 - реализовать функцию **paint()**, которая реализует непосредственную отрисовку элемента.

QGraphicsView

- Виджет для отображения содержимого **QGraphicsScene**
- **QGraphicsView** отображает содержимое **QGraphicsScene** в области прокрутки

QGraphicsItem

```
class Pacman : public QObject, public QGraphicsItem
{
    Q_OBJECT
public:
    explicit Pacman(QObject *parent = 0);
    ~Pacman();

    void moving(int x, int y);

protected:

    QRectF boundingRect() const;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
}
```

СЛОТЫ И СИГНАЛЫ

Средства, позволяющие эффективно производить обмен информацией о событиях между объектами.

- класс унаследованный от **QObject** может иметь сколько угодно слотов и сигналов
- сообщения, передаваемые через сигналы, могут иметь сколько угодно аргументов любого типа
- сигнал может соединяться с различным количеством слотов
- слот может принимать множество сигналов от множества объектов
- при уничтожении объекта все связи слот-сигнал этого объекта уничтожаются

Сигнал

- Сигнал - метод, осуществляющий пересылку сообщений.
- Сигналы определяются в классе как методы, только без реализации.
- Сигнал не обязательно соединять со слотом.
- Существуют готовые сигналы, также можно реализовывать свои.

СЛОТ

- Метод, присоединяющийся к сигналу.
- В слотах нельзя использовать значения по умолчанию.
- Нельзя определять их как `static`.

Соединение

```
QObject::connect(  
    const typename QtPrivate::FunctionPointer<Func1>::Object *sender,  
    Func1 signal,  
    const QObject *context,  
    Func2 slot,  
    Qt::ConnectionType type = Qt::AutoConnection)
```

- **sender** - объект, отправляющий сигнал
- **signal** - сигнал, с которым устанавливается соединение
- **context** - указатель на объект, имеющий слот для обработки сигнала
- **slot** - функция, вызываемая при получении сигнала
- **type** - режим обработки

Сигнал clicked()

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    ...
    Pacman *pacman; // пакман
    QPushButton *start_button; //кнопка начала игры

public slots:
    //обработчик нажатия кнопки start_button
    void start_handler();
};
```

QAbstractButton сигналы

- `pressed()` - на кнопку нажали
- `released()` - кнопку отпусти
- `toggled()` - изменение состояние кнопки, имеющей статус выключателя
- `clicked()` (*Pressed & Released*) - нажатие на кнопку

Установка обработчика кнопки

```
MainWindow::MainWindow(QWidget *parent) :  
    QMainWindow(parent),  
    ui(new Ui::MainWindow)  
{  
    ...  
  
    start_button = new QPushButton("START");  
    connect(start_button, SIGNAL(clicked()), this, SLOT(start_handler()));  
    ...  
}
```

```
//создаем диалоговое окно и просим подтвердить
QMessageBox* confirm_start = new QMessageBox(QMessageBox::Information, "Test dialog",
                                             "Are you sure?",
                                             QMessageBox::Yes | QMessageBox::Cancel);

//если пользователь нажал Yes
if (confirm_start->exec() == QMessageBox::Yes)
{
    ....
    return;
}

//если пользователь нажал Cancel или закрыл окно
else
{
    ...
}
```

Диалоговое окно

- Модальные - блокируют работу остального приложения
- Немодальные - позволяют продолжать работу

Обработка нажатия КЛАВИШ

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:

    ...
public slots:
    void start_handler();
    void keyPressEvent(QKeyEvent *ke);
};
```

Обработка нажатия

```
void MainWindow::keyPressEvent(QKeyEvent *ke)
{
    switch(ke->key())
    {
        case Qt::Key_W:
            pacman->moving(0, -5);
            break;

        case Qt::Key_S:
            pacman->moving(0, 5);
            break;

        case Qt::Key_A:
            pacman->moving(-5, 0);
            break;

        case Qt::Key_D:
            pacman->moving(5, 0);
            break;
    }
}
```