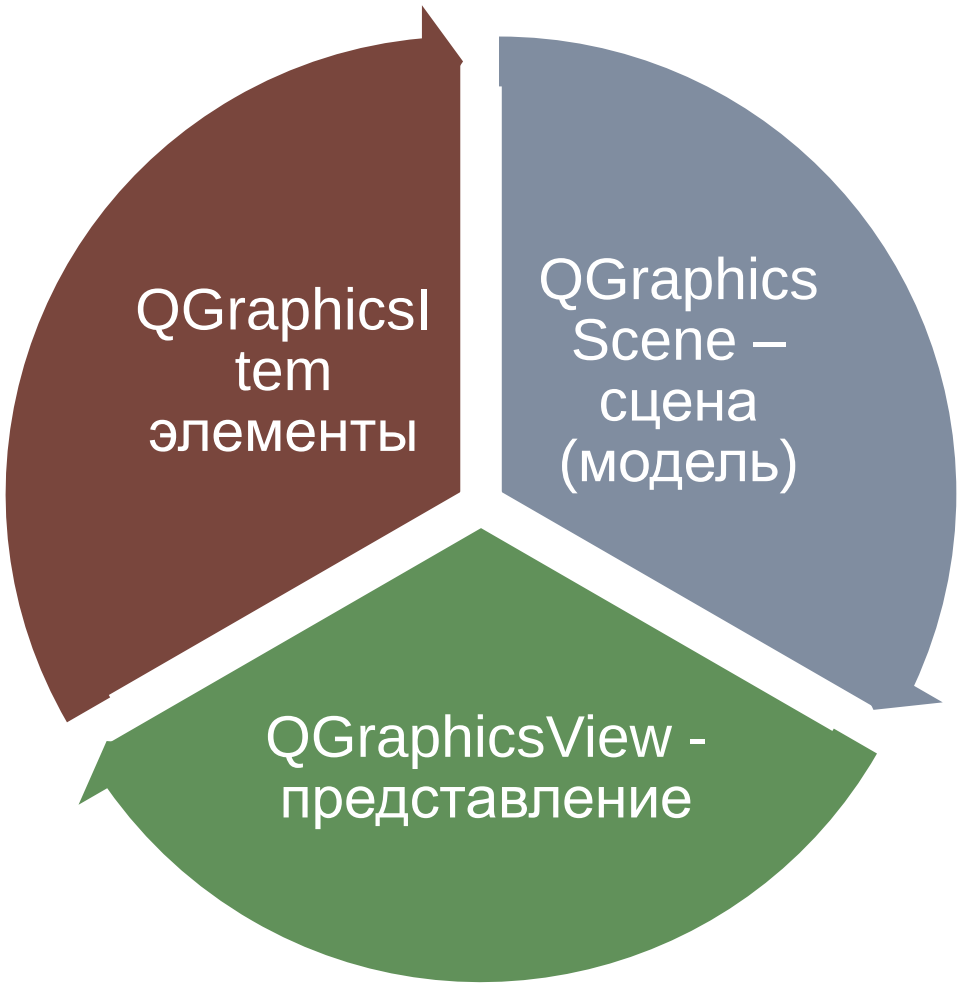
СЕМИНАР 8

Работа с графикой в Qt

Снова рисуем компас

Qt Graphics View Framework

- Графическое представление (Graphics View Framework) – инструмент для управления и взаимодействия с большим количеством элементов двумерных изображений.
- В основе 3 класса: QGraphicsItem, QGraphicsScene, QGraphicsView и концепция «модель-представление» (Model-View).



Интервью или модель-представление

Рис. 12.1. Взаимодействие компонентов «интервью»

Составляющие модели:

- Модель
- Представление
- Выделение элемента
- Делегат

Модель

Модель – отвечает за управление данными и предоставляет интерфейс для чтения и записи данных.

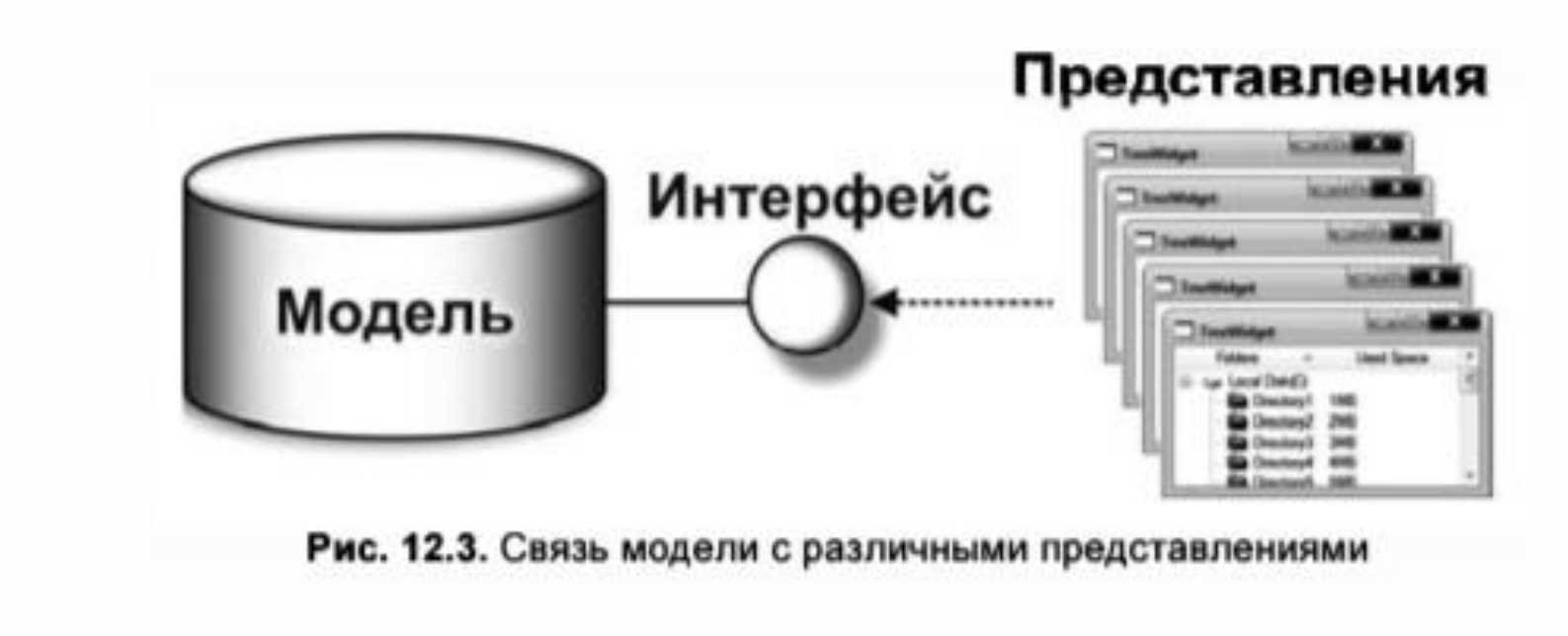


Рис. 12.3. Связь модели с различными представлениями

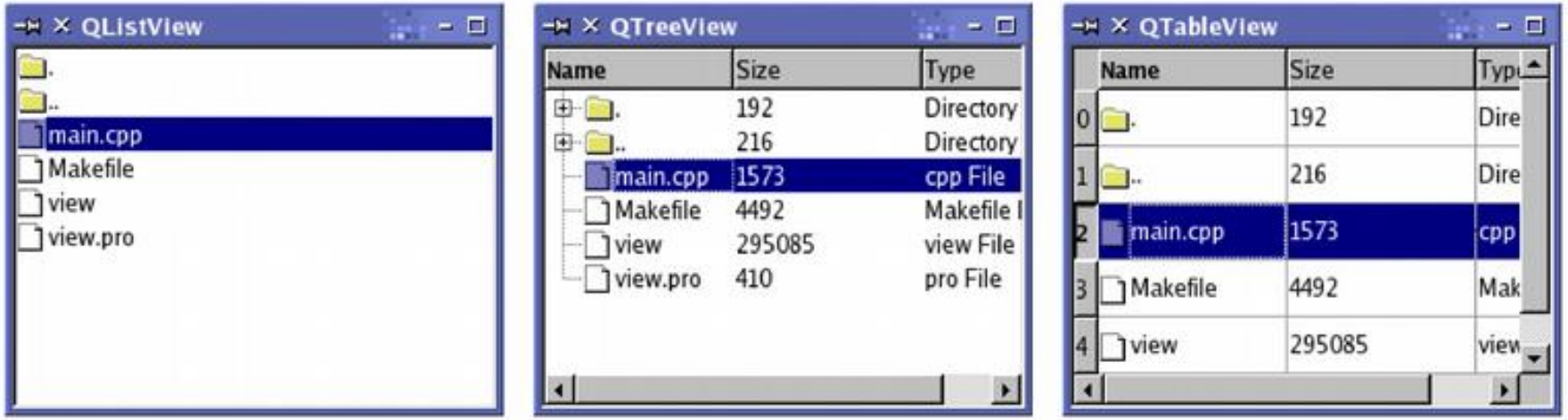
Представление

Представление – отвечает за представление данных пользователю и за их расположение.

Одна модель может использовать множество представлений.

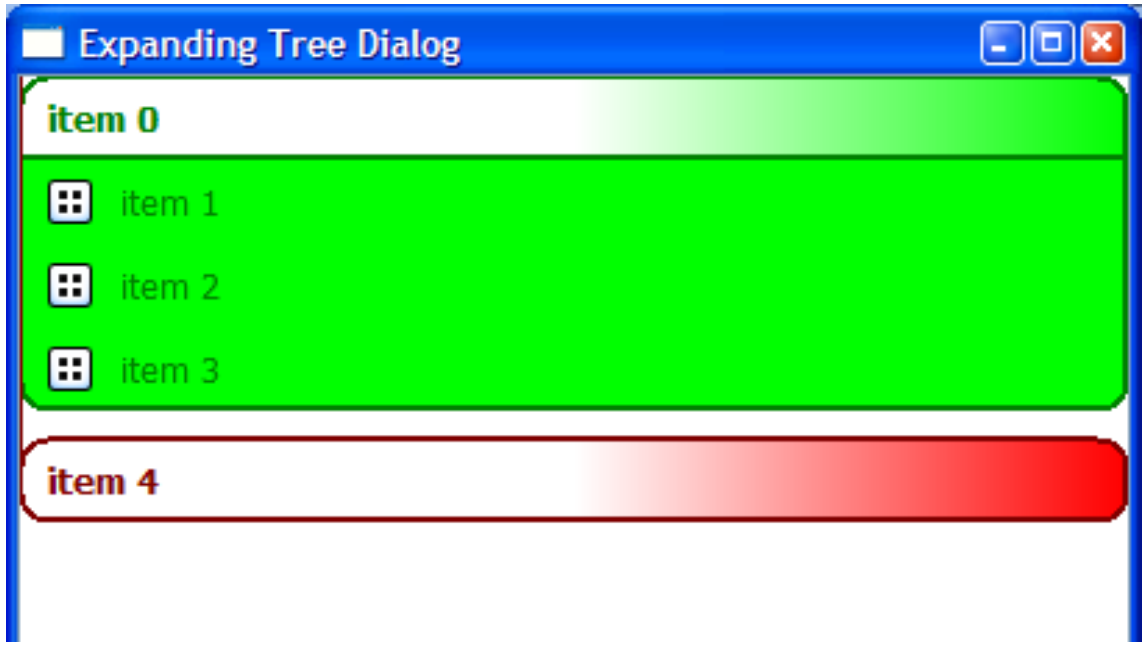
Выделение элемента

Выделение элемента – специальная модель, отвечающая за централизованное использование элементов



Делегат

Делегат – отвечает за рисование каждого элемента в отдельности, а также за его редактирование



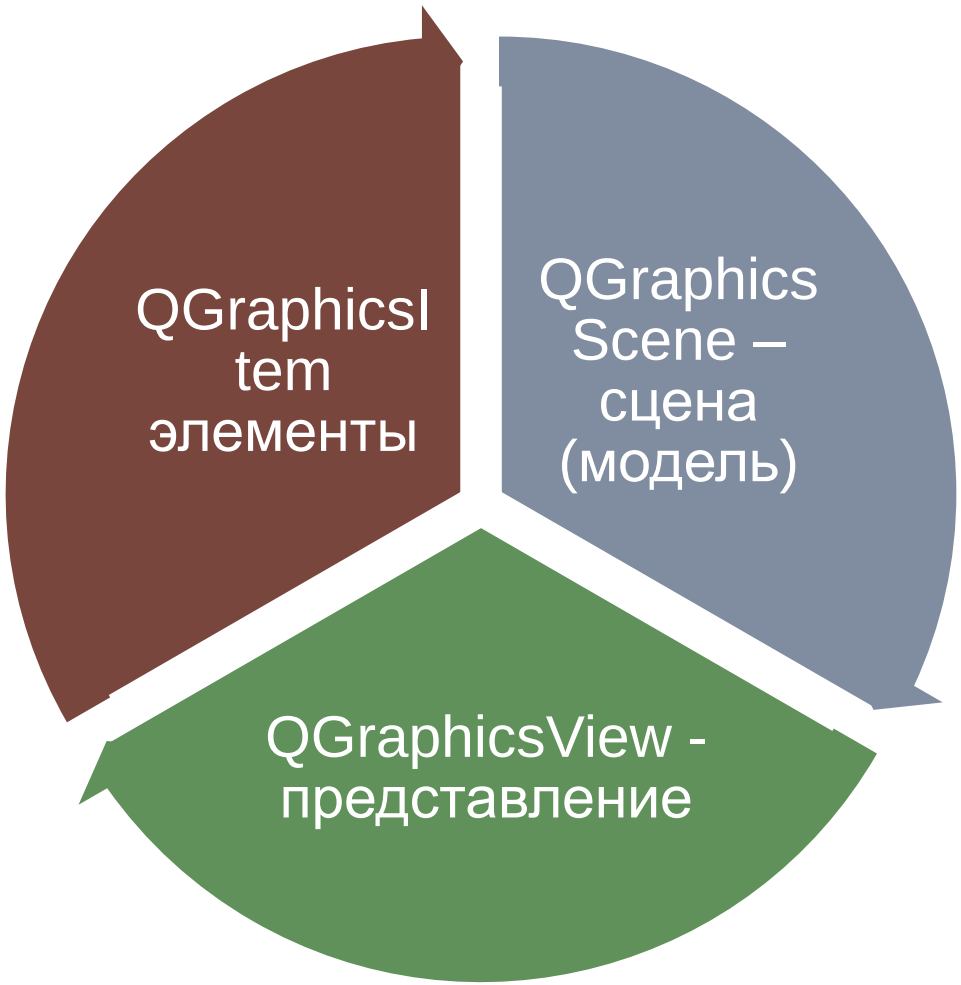
Интервью или модель-представление

Достоинства:

- Возможность показа данных в нескольких представлениях без дублирования.
- Возможность внесения изменений с минимумом временных затрат.
- Удобство программного кода.
- Упрощение интеграции баз данных.

Qt Graphics View Framework

- Графическое представление (Graphics View Framework) – инструмент для управления и взаимодействия с большим количеством элементов двумерных изображений.
- В основе 3 класса: QGraphicsItem, QGraphicsScene, QGraphicsView и концепция «модель-представление» (Model-View).

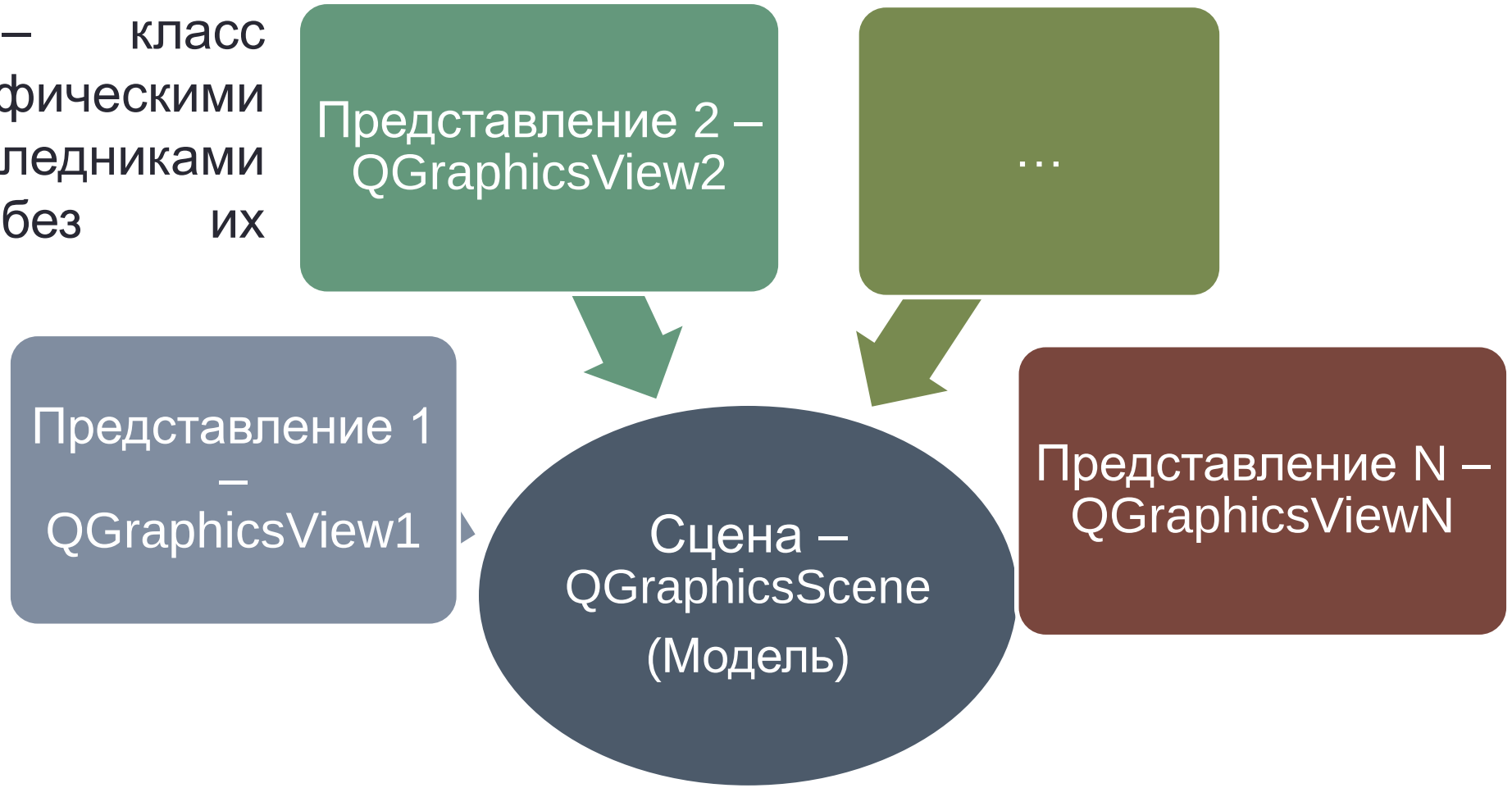


Графическое представление Qt ::QGraphicsScene (Сцена)

Одной сцене могут соответствовать несколько виджетов представления (QGraphicsView):

QGraphicsScene – класс управления графическими элементами (наследниками QGraphicsItem) без их отображения.

Отображением занимается QGraphicsView

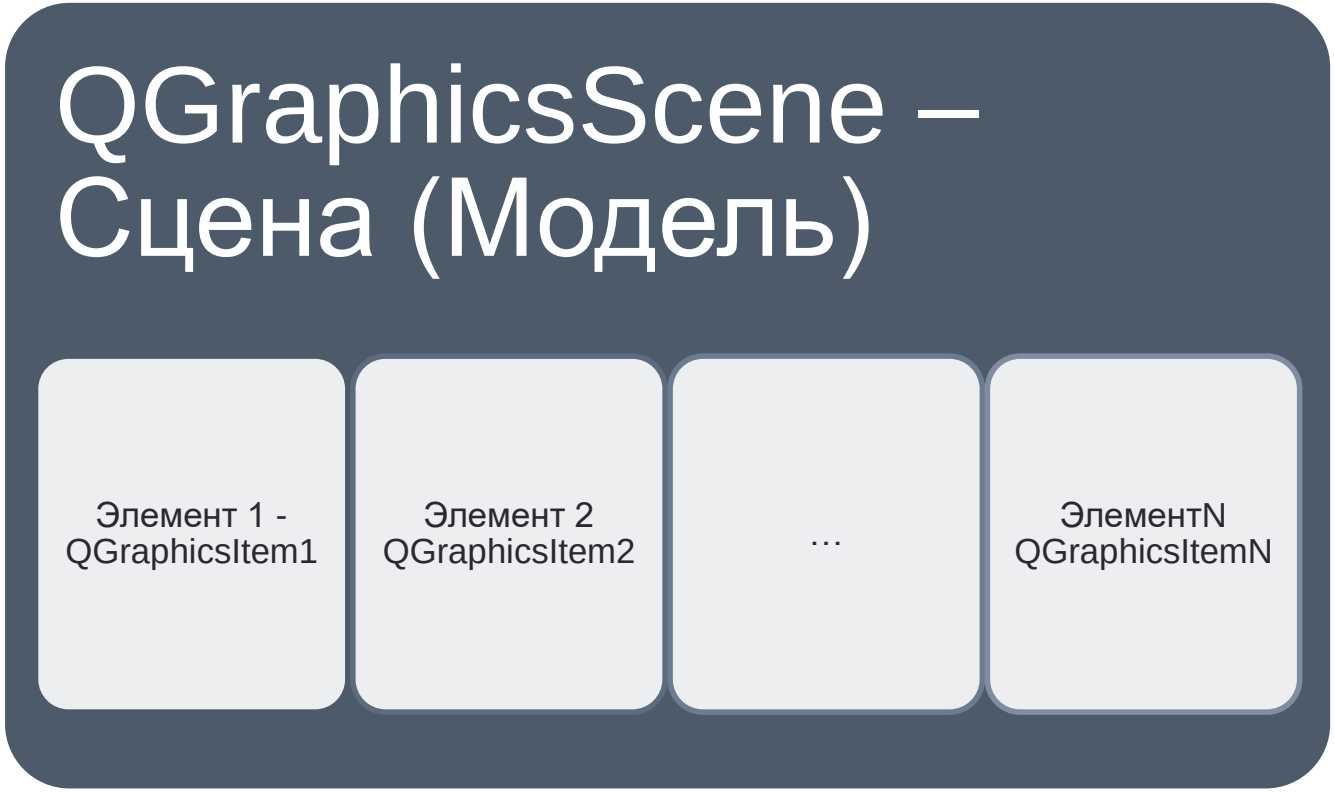


Графическое представление Qt ::QGraphicsScene (Сцена)

Для чего нужен QGraphicsScene?

1. Этот класс служит как контейнер для элементов сцены (QGraphicsItems).

1.1 При этом элемент (QGraphicsItems) может одновременно принадлежать только одной сцене



Графическое представление Qt ::QGraphicsScene (Сцена)

Для чего нужен QGraphicsScene?

2. Определяет и запоминает положение элементов.
3. Отправляет сигналы и события элементам сцены.
   - Управляет обнаружением столкновений (collision detection)
   - Управляет «выделением» элементов и т.п. взаимодействием с пользователем
4. Управляет отрисовкой и перерисовкой сцены.
   - Управляет Z-индексированием (z-order) элементов;

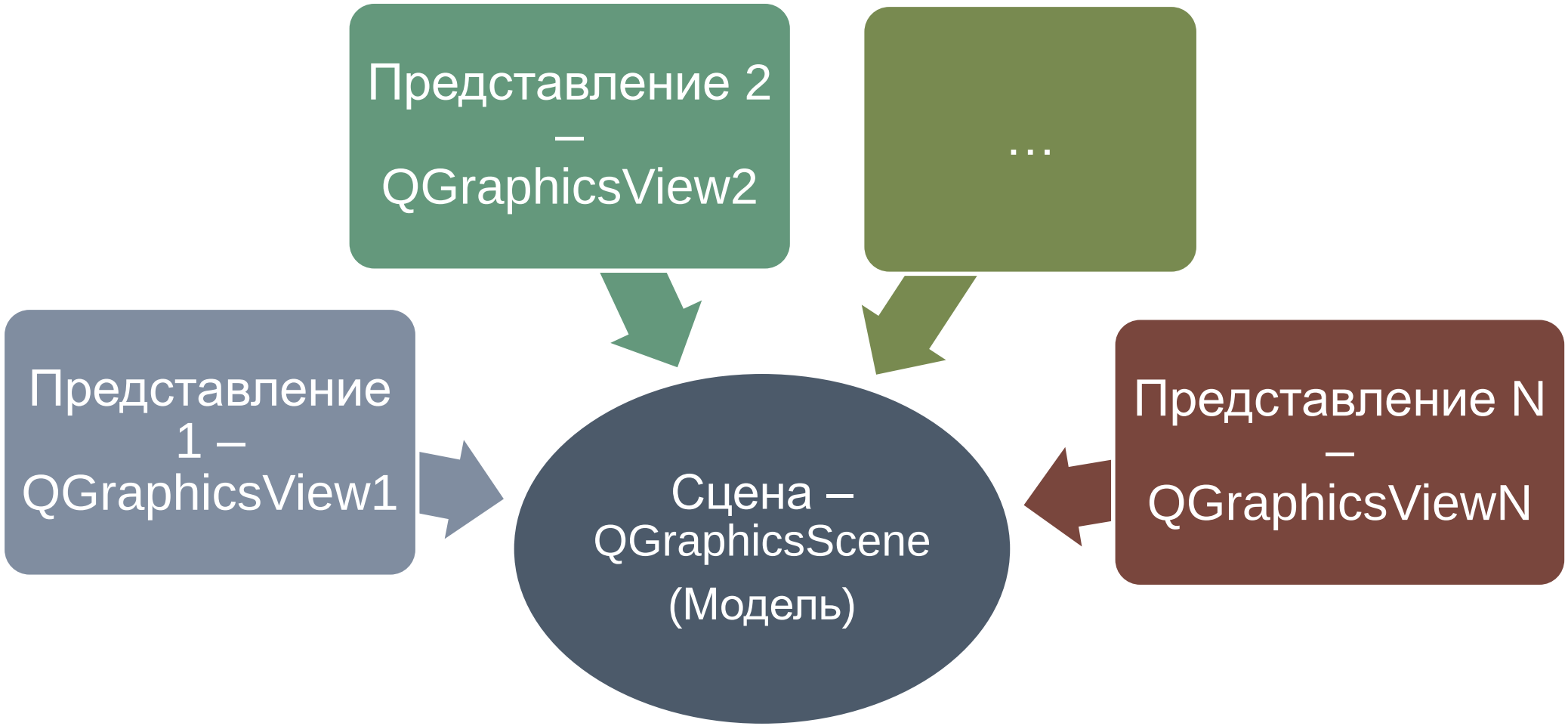
Графическое представление Qt ::QGraphicsScene (Сцена)

Важные методы класса QGraphicsScene:

addItem() **addElipse(), addPolygon(), addText() etc.**	**Добавляет элемент на сцену**

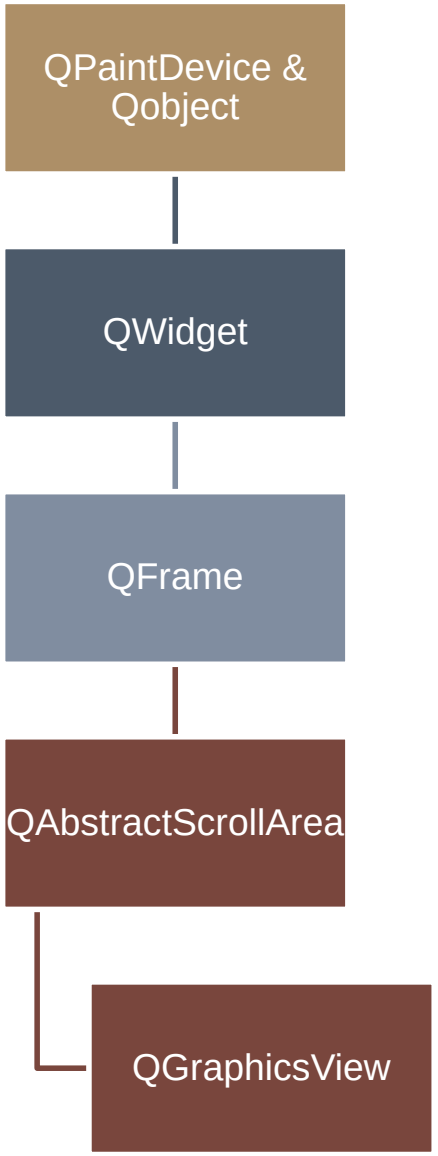
Графическое представление Qt ::QGraphicsView (Представление)

Одной сцене могут соответствовать несколько виджетов представления (QGraphicsView):

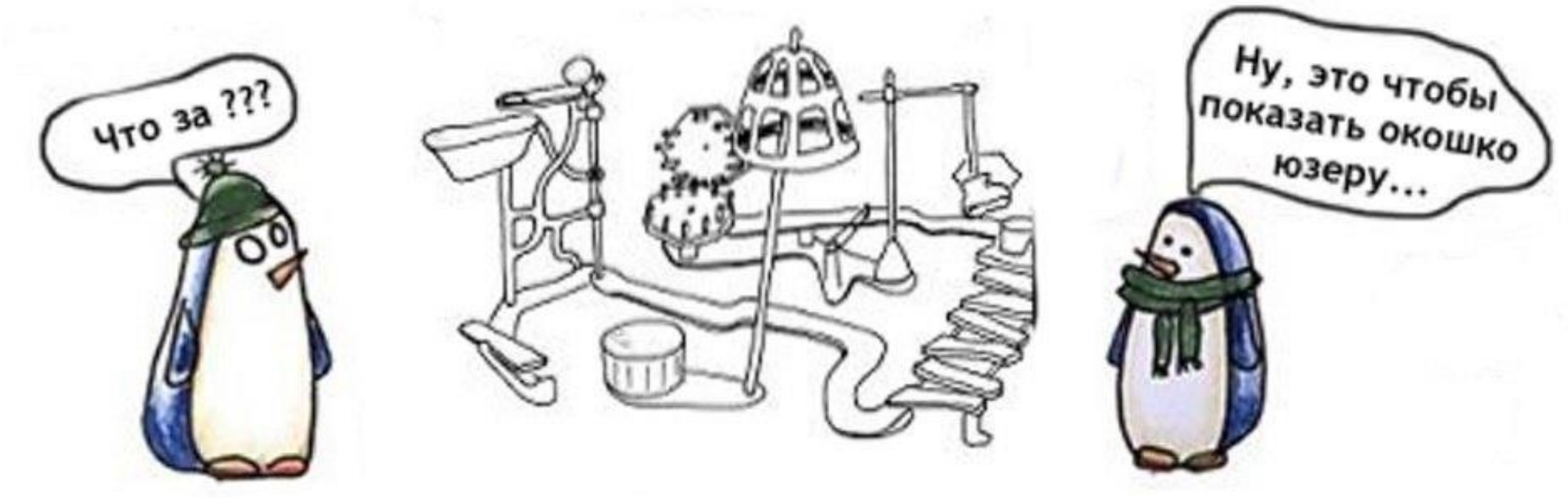


Графическое представление Qt ::QGraphicsView (Представление)

- Позволяет отображать одну и ту же сцену (QGraphicsScene) в разных представлениях QGraphicsView.
  - Поддерживает масштабирование, вращение и прочие трансформации.
  - Передает события в QGraphicsSceneEvents.
  - Управляет преобразованием координат между сценой и представлением.
  - Поддерживает OpenGL.
- Все элементы, хранящиеся в сцене, автоматически отображаются в представлении.



Qt Graphics View Framework



Графическое представление Qt ::QGraphicsView (Представление)

Важные методы класса QGraphicsView:

setScene()	**Устанавливает сцену, с которой будет связано представление**
setRenderHint()	Установка Antialiasing и т.п. параметров отображения сцены
centerOn()	Центрирование представления относительно некоторой точки или виджета
Scale(), rotate(), translate(), setMatrix()	Трансформации изображения

Графическое представление Qt ::QGraphicsItem (Элемент)

QGraphicsItem – абстрактный класс, который позволяет создавать классы элементов сцены (QGraphicsScene).

Тип графического изображения	**Класс элемента сцены Qt**
Геометрические фигуры	QGraphicsEllipseItem, QGraphicsLineItem, QGraphicsPathItem, QGraphicsPolygonItem, QGraphicsRectItem
Изображения	QGraphicsPixmapItem;
Текст	QGraphicsTextItem
Виджеты	QGraphicsWidget
И тп…	…

Графическое представление Qt ::QGraphicsItem (Элемент)

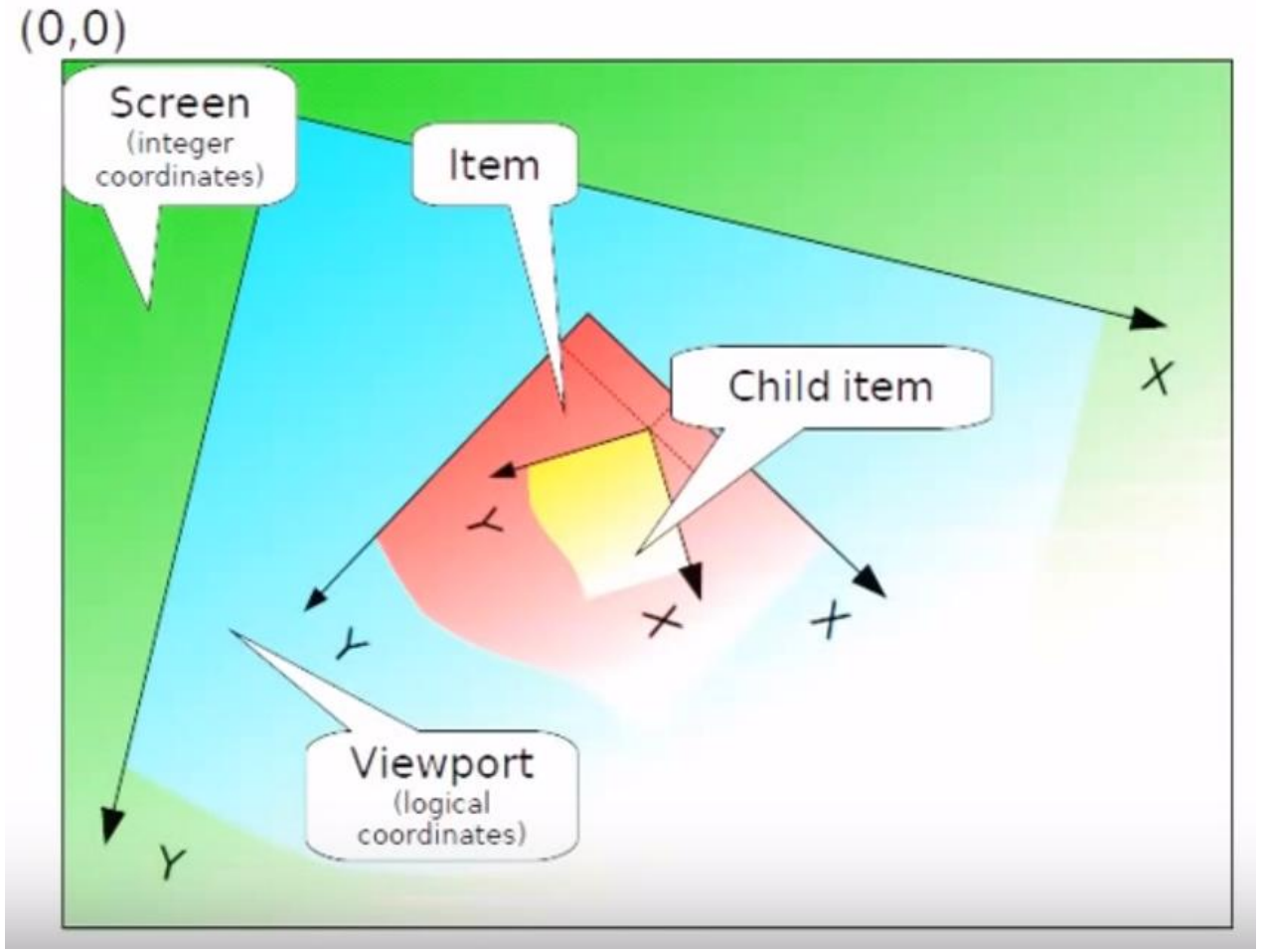
- Поддерживает механизмы преобразования координат элементов
  - Перемещение, масштабирование, вращение;
  - Использует при этом локальную систему координат элемента.
- Поддерживает обработку событий мыши, клавиатуры и механизм drag & drop
- Поддерживает механизм родитель/потомок для элементов сцены

Графическое представление Qt ::QGraphicsItem (Элемент)

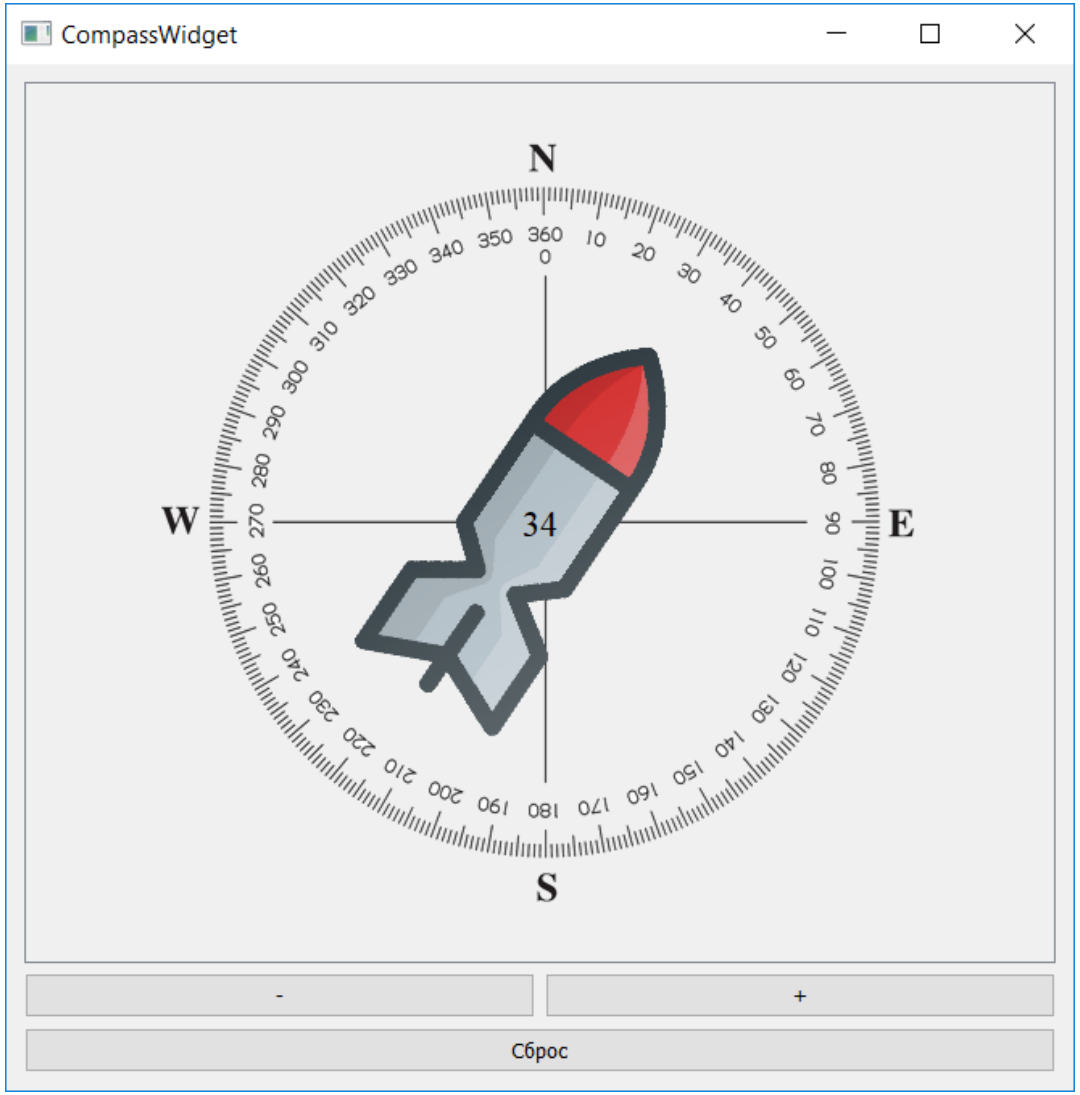
Важные методы класса QGraphicsItem

Hide(), show()	Скрытие и показ элемента на сцене
setParent	Установка родительского элемента
Pos() / setPos()	Текущее положение элемента / Установка местоположения элемента
Scale(), rotate(), translate(), setMatrix()	Трансформации элемента в его локальных СК
setEnabled()	Установка доступного/недоступного состояния
moveBy	Перемещение элемента, относительно текущего положения
zValue()	Текущий индекс в Z-порядке элементов сцены
setFlags()	Настройка различных параметров элемента Например: QGraphicsItem::ItemIsMovable – элемент можно перемещать

Графическое представление Qt. Системы координат



Практическая часть. Рисуем компас 2.



- В шаблоне CompassWidget лежат ui.-файл, файл ресурсов и изображения, необходимые для создания проекта.
- Необходимо дописать код таким образом, чтобы при нажатии на кнопки «+», «-» увеличивалось значение текущего курса. Текущий курс должен быть выведен в текстовом поле в центре виджета, а рисунок аппарата должен поворачиваться в соответствии с накопленным значением текущего курса.
- При нажатии на кнопку «Сброс» текущее значение курса должно быть 0.

Практическая часть. Рисуем компас 2.

compasswidget.h

CompassWidget/compasswidge...* | # | scene: QGraphicsScene *

```
#ifndef COMPASSWIDGET_H
#define COMPASSWIDGET_H

#include <QWidget>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsPixmapItem>
#include <QGraphicsTextItem>

#include "ui_compasswidget.h"

class CompassWidget : public QWidget, Ui::CompassWidget {
    Q_OBJECT

public:
    explicit CompassWidget(QWidget *parent = 0);
    ~CompassWidget();

private:
    double currentYaw, setYaw;

    QGraphicsScene * scene; //создаем указатель на объект сцены
     //создаем указатели на элементы сцены (картинки)
    //просто так картинку на сцену добавить не можем, для этого необходим класс
    //QGraphicsPixmapItem,
    //наследник от абстрактного класса QGraphicsItem, предназначенный для
    //работы с риснуками в графическом представлении Qt
```

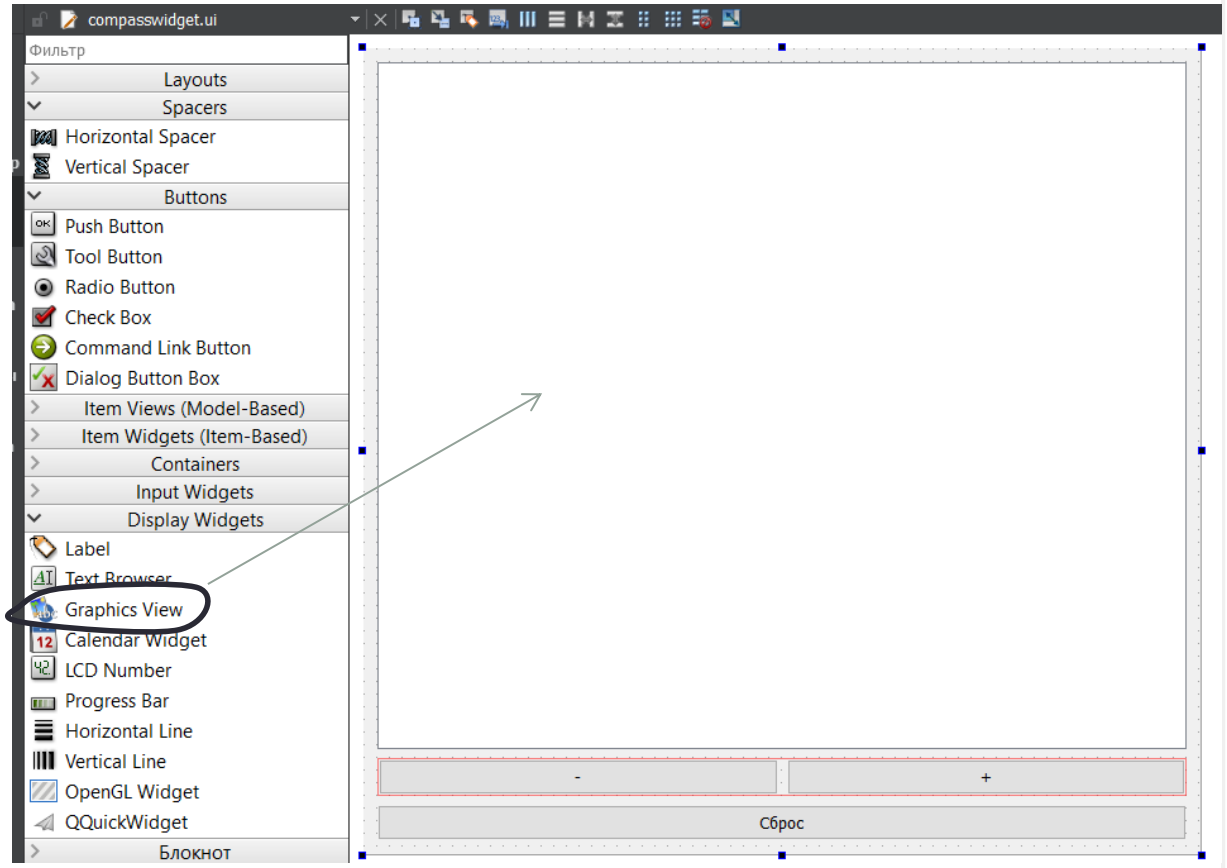
Практическая часть. Рисуем компас 2.

compasswidget.h

```
    QGraphicsPixmapItem *picROV; //картинка стрелки компаса (в нашем варианте это АНПА)
    QGraphicsPixmapItem *picDial; //картинка циферблат компаса
    //txtCurrentYaw - текст с текущим значением угла курса
    QGraphicsTextItem *txtCurrentYaw;//аналогично создаем указатель на элемент сцены (текст)
    //просто так текст на сцену добавить не можем, для этого используем
    //класс QGraphicsTextItem - наследник от абстрактного класса QGraphicsItem,
    //предназначенный для работы с текстом в графическом представлении Qt

public slots:
    //слоты, которые будут соединены с сигналами кнопок
    //увеличения, уменьшения и сброса текущего курса
    void addCurrentYaw(bool);
    void decreaseCurrentYaw(bool);
    void resetCurrentYaw(bool);

};

#endif // COMPASSWIDGET_H
```

Практическая часть. Рисуем компас 2.

compasswidget.ui



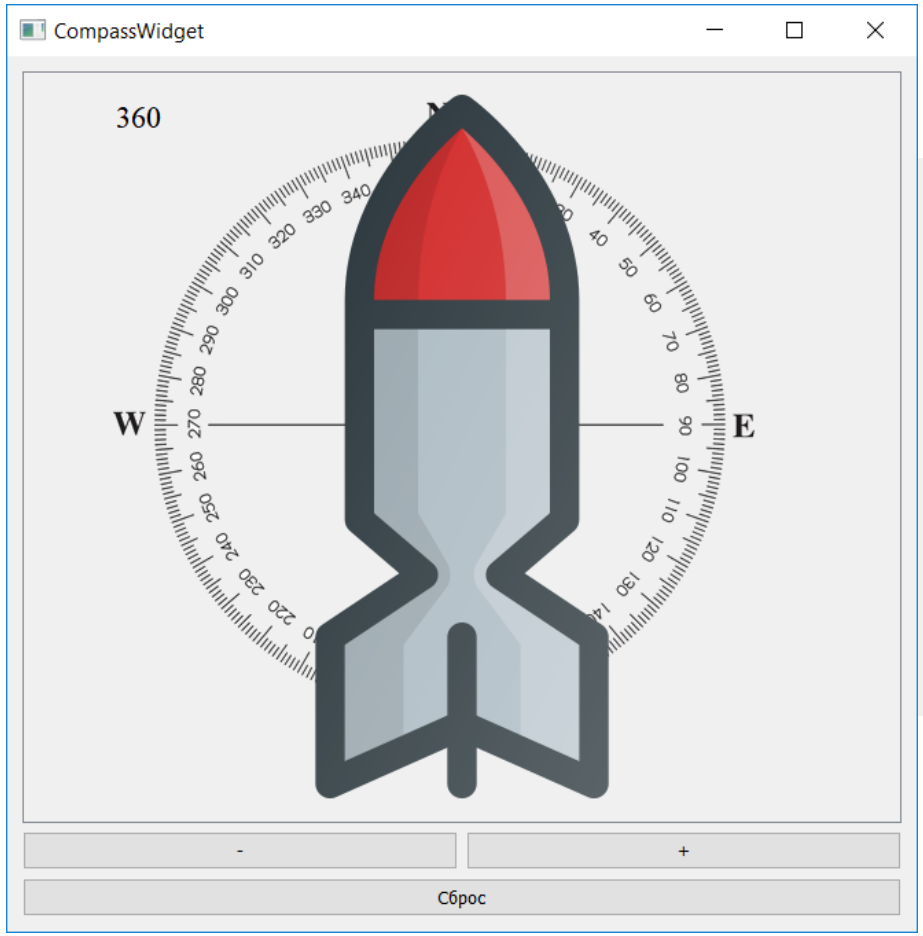
compasswidget.cpp

Тем временем в конструкторе класса CompassWidget…

```cpp
#include "compasswidget.h"
#include <QDebug>


CompassWidget::CompassWidget(QWidget *parent) :
    QWidget(parent) {
    setupUi(this);
    currentYaw=0;//начальная инициализация переменной курса

    //для создания объекта сцены в конструктор класса можно передать
    //параметры прямоугольной области, которую будет занимать сцена.
    //Это можно сделать следующим образом:
    //1. передав координаты x,y, ширину и длину вещественными числами,
    //2 - передав объект QRectF.
    //Однако мы воспользуемся упрощенным способом создания сцены и
    //передадим в контструктор только указатель на объект родитель сцены
    //им будет объект представления (QGraphicsView) view,
    //созданный в форме compasswidget.ui
    scene = new QGraphicsScene (view);

    //Создать объект графического представления QGraphicsView() можно
    //в коде, а можно и в Qt Designer сразу разместить его на форме
    //в нашем случае объект был создан и размещен на форме и называется view
    //представление должно быть связано со сценой, для этого
    //используется метод setScene()
    view->setScene(scene);
```

Практическая часть. Рисуем компас 2.

Промежуточный результат:



compasswidget.cpp

```
27
28      //установим прозрачный фон для нашего представления
29      view->setStyleSheet("background: transparent");
30      //и "сглаживание" Antialiasing
31      view->setRenderHint(QPainter::Antialiasing);
32      //картинки для компаса представляют собой изображения формата PNG,
33      //размещены в директории images проекта и добавлены в файл ресурсов resources.qrc
34      //чтобы создать элементы сцены QGraphicsPixmapItem - picDial и picROV
35      // воспользуемся методом addPixmap для сцены, который вернет указатель на
36      //объект QGraphicsPixmapItem, который нам понадобится для дальнейшей работы
37      picDial = scene->addPixmap(QPixmap(":/images/dial.png"));
38      picROV = scene->addPixmap(QPixmap(":/images/rov.png"));
39      //аналогично добавим текст с текущим значением курса
40      txtCurrentYaw = scene->addText("360", QFont("Times New Roman",14));
41
```

Практическая часть. Рисуем компас 2.

compasswidget.cpp

```
    //объект QGraphicsPixmapItem, который нам понадобится для дальнейшей работы
    picDial = scene->addPixmap(QPixmap(":/images/dial.png"));
    picROV = scene->addPixmap(QPixmap(":/images/rov.png"));
    //аналогично добавим текст с текущим значением курса
    txtCurrentYaw = scene->addText("360", QFont("Times New Roman",14));
    //Если на этом этапе скомпилировать программу, то очевидно
    //то необходимо подкорректировать размеры и положения некоторых элементов
    //изменим масштаб изображения аппарата
    picROV->setTransform(QTransform::fromScale(0.5, 0.5));
    //сдвинем по вертикали и горизонтали НПА, таким образом, чтобы совместить
    //его центр с центром подложки компаса
    picROV->setPos(picDial->pixmap().width()/2-picROV->pixmap().width()/4, \
                   picDial->pixmap().height()/2-picROV->pixmap().height()/4);
    //Трансформации координат элементов можно также задавать с помощью
    //класса QTransform. передвинем текст, используя методы этого класса
    //и метод setTransform для графического элемента текста
    QTransform t;
    t.translate(picDial->pixmap().width()/2-14, picDial->pixmap().height()/2-14);
    txtCurrentYaw->setTransform(t);
    //повернуть картинку можно используя метод setRotation()
    //picROV->setRotation(угол на который хотим повернуть)
    //но элемент будет вращать СК не относительно центра картинки
    //а относительно начала своей локальной системы координат (т.е. левого верхнего угла)
    //однако можно установить центр вращения методом setTransformOriginPoint()
    //чтобы все преобразования в дальнейшем производились относительно указанной
    //в этом методе точки
    picROV->setTransformOriginPoint(picROV->pixmap().width()/2, picROV->pixmap().height()/2);
64
```

Практическая часть. Рисуем компас 2.

Осталось только соединить сигналы и слоты…

И виджет готов!