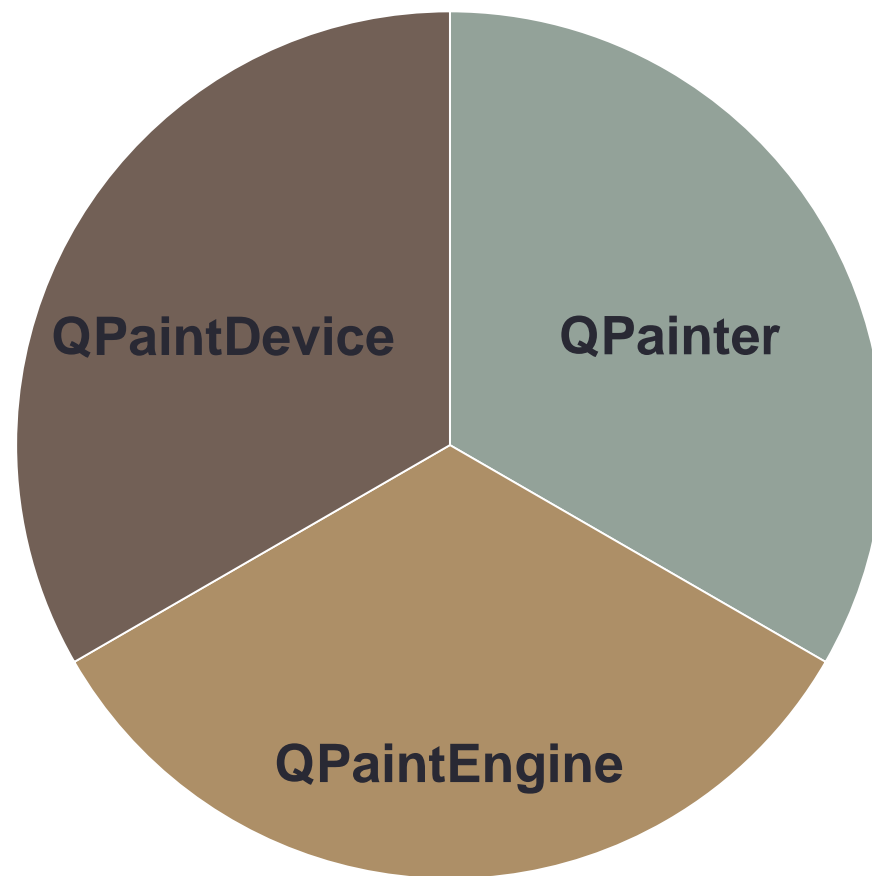


СЕМИНАР 7.1

Рисование в Qt

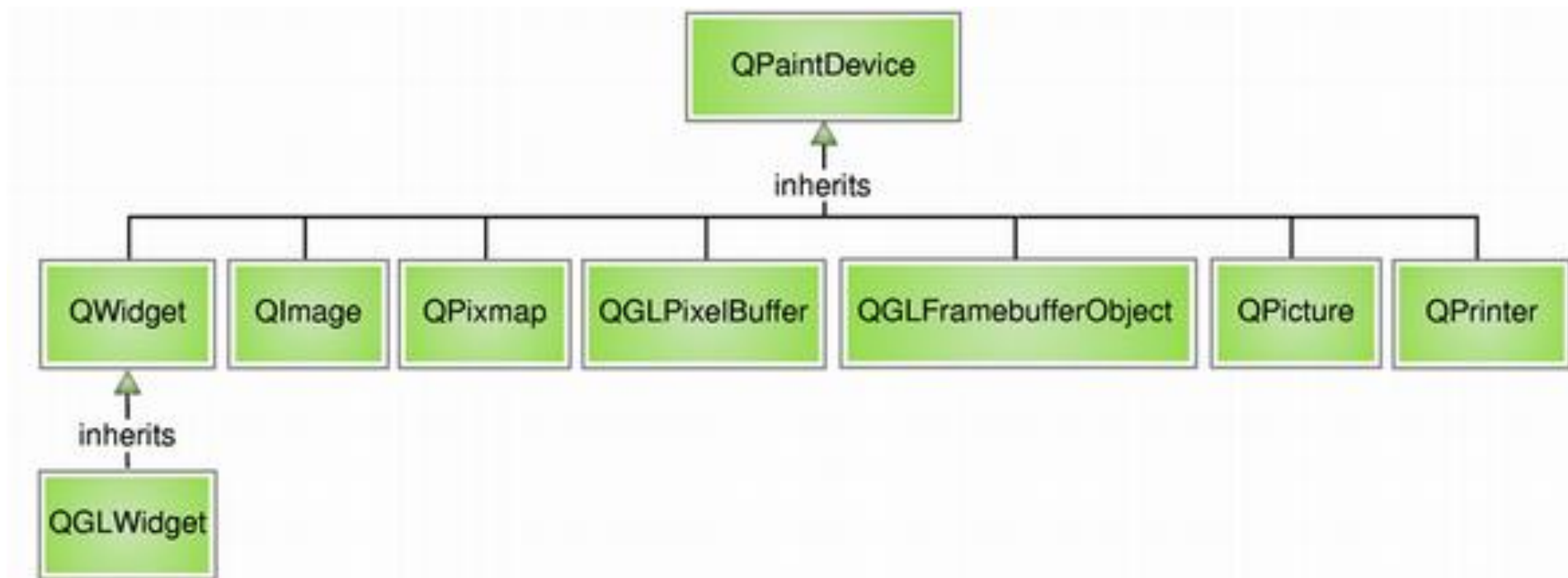
Arthur. Архитектура рисования Qt



QPaintDevice

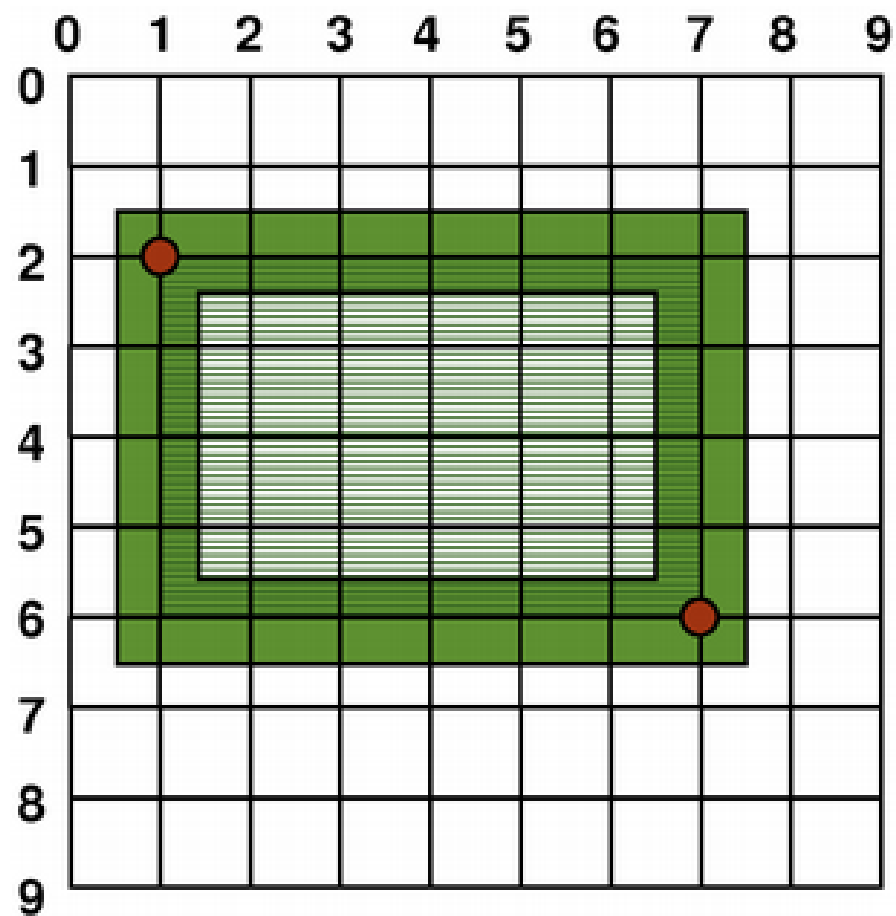
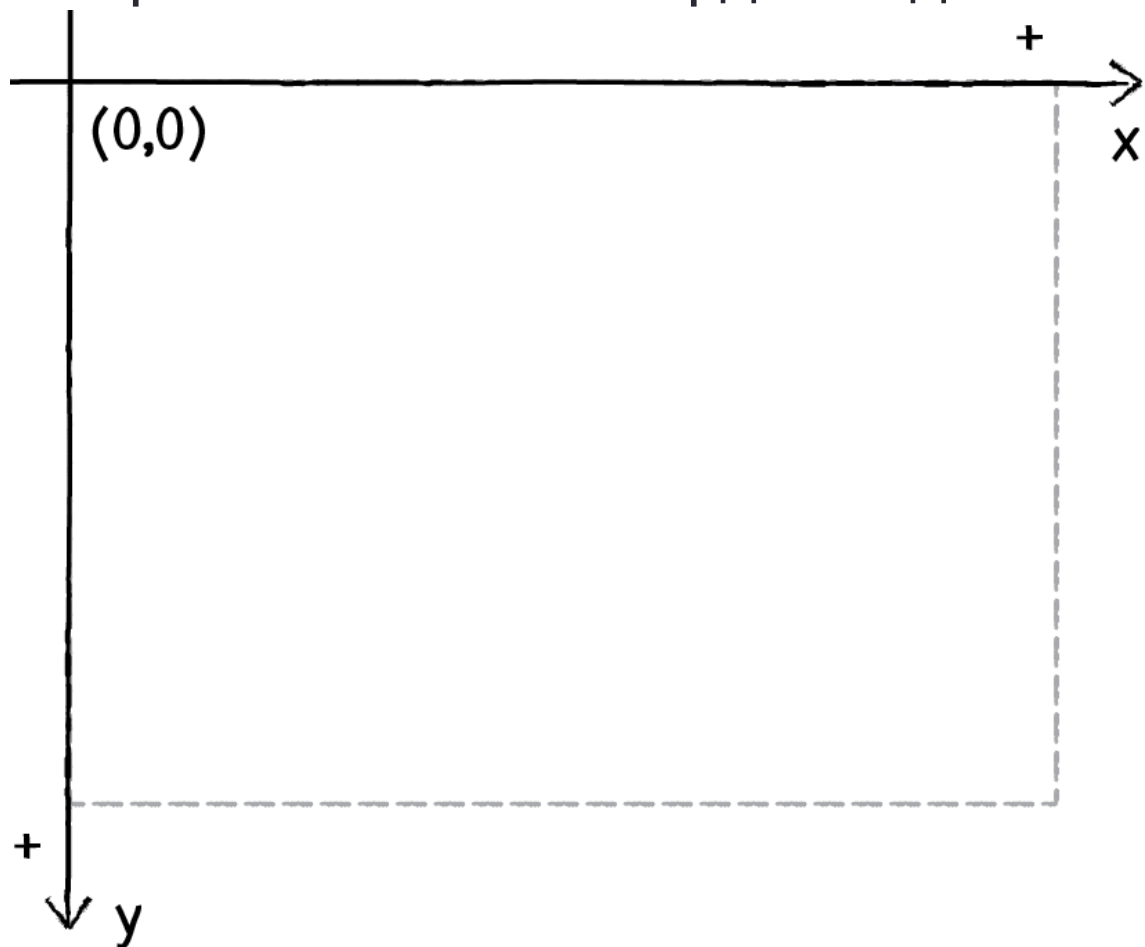
Контекст рисования (QPaintDevice) - абстракция двумерного пространства, на которой можно рисовать, используя QPainter.

QPaintDevice – основной абстрактный класс для всех классов объектов, которые можно рисовать.



QPaintDevice

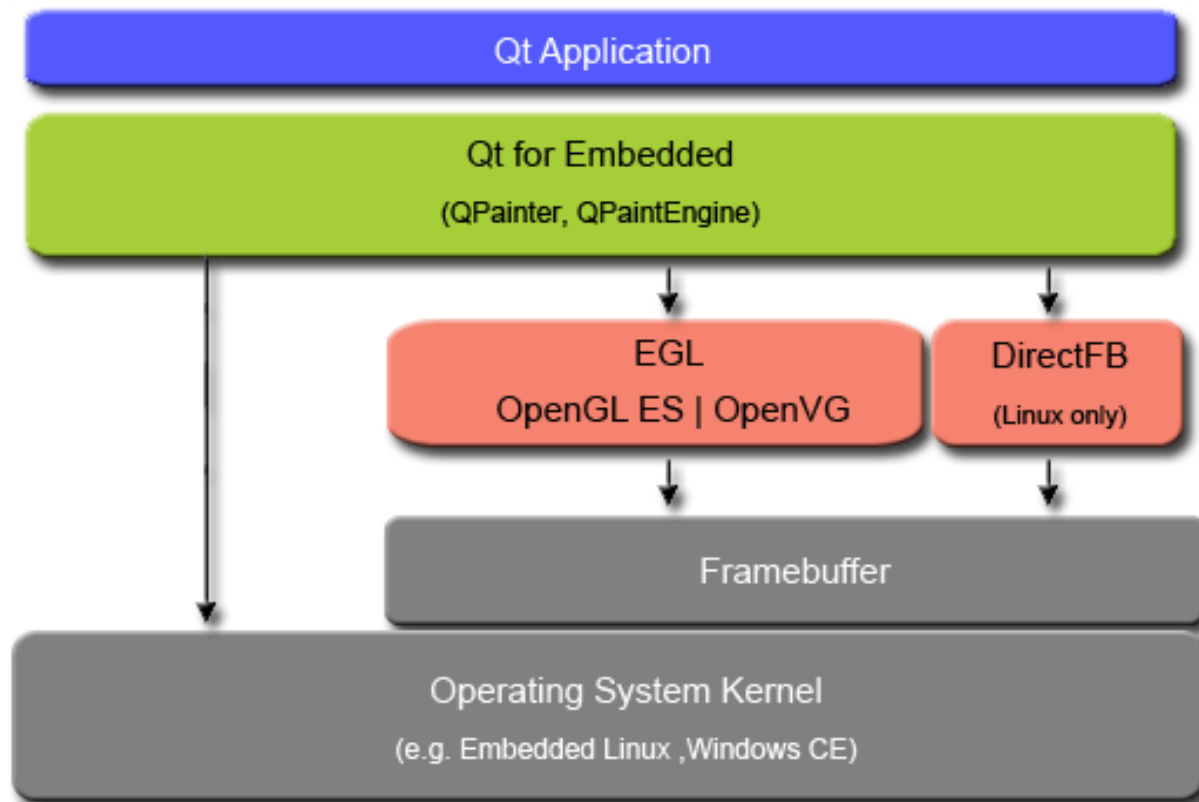
Направление осей координат для объектов-наследников QPaintDevice



QPaintEngine

QPaintEngine — класс абстрактного описания процесса рисования QPainter на указанном устройстве на указанной платформе.

Используется для внутренних нужд в QPainter и QPaintDevice, и невидим для программистов приложений если только они не создадут свой собственный тип устройства.



QPaintEngine



Предоставляет механизмы рисования для следующих платформ:

- Windows;
- Linux;
- Mac OS.

QPainter

- Выполняет низкоуровневое рисование на виджетах и других устройствах рисования, **являющихся наследниками класса QPainterDevice.**
- Может рисовать **всё**: от простых до сложных фигур..



QPainter

При рисовании на виджетах рисовать можно только! переопределив виртуальную функцию `paintEvent()` для класса виджета.

Алгоритм рисования на `QWidget` в методе `paintEvent()`:

`void QPainter::paintEvent()`



Шаг 1. Выбор «на чём» рисовать



Шаг 2. Выбор «чем» рисовать



Шаг 3. Рисование

END.

**Шаг 4. `end()` – освобождение
контекста рисования**

paintEvent()

- Для виджетов рисование производится из метода обработки события QPaintEvent.
- paintEvent() вызывается при вызове методов show(), repaint(), update().
- т.е. в классе объявляем (переопределяем) функцию paintEvent()

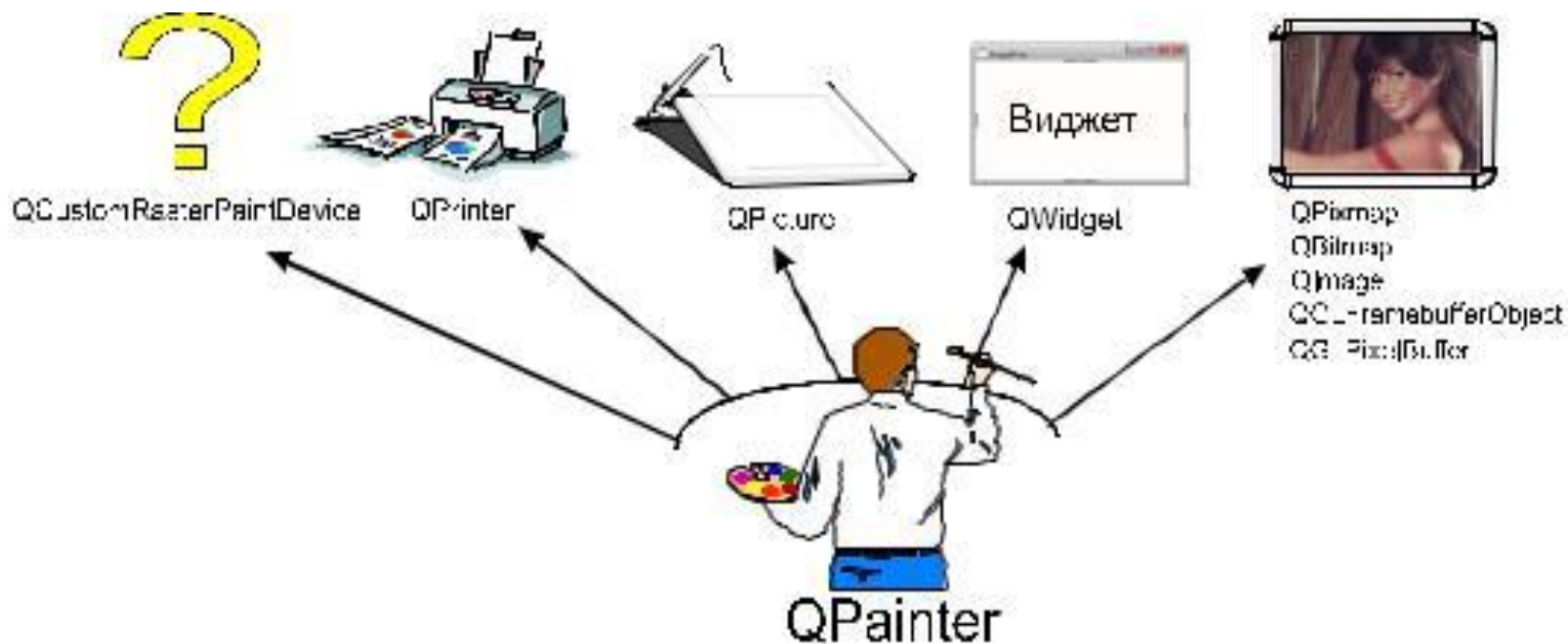
```
protected:  
    //переопределяем функцию paintEvent для отрисовки нашего безобразия  
    void paintEvent(QPaintEvent *e);
```

- И добавляем реализацию функции...

```
void PicFrame::paintEvent(QPaintEvent *e){  
    //создаем класс QPainter  
    //передаем ему адрес объекта контекста рисования (на чем рисуем)  
    //настраиваем параметры инструментов рисования (цвета, шрифты,  
    //толщины линий и т.п.)  
    //рисуем  
}
```

QPainter. Шаг 1. Выбор контекста рисования.

Чтобы использовать объект QPainter необходимо передать ему адрес объекта контекста, на котором должно осуществляться рисование.



QPainter. Шаг 1. Выбор контекста рисования.

Способы задания контекста рисования:

- 1. В конструкторе класса QPainter:

```
void PicFrame::paintEvent(QPaintEvent *e){  
    //в качестве примера рассмотрим рисование на форме  
    //данного класса (т.е. указать this)  
    //для указания контекста рисования просто передаем  
    //указатель на контекст в конструкторе класса QPainter  
    QPainter painter(this);
```

QPainter. Шаг 1. Выбор контекста рисования.

Способы задания контекста рисования:

- 2. С помощью метода QPainter::begin()

```
void PicFrame::paintEvent(QPaintEvent *e){  
  
    QPainter painter;  
    //второй метод передачи адреса объекта, на котором будем рисовать  
    //метод с использованием begin()  
    //достоинство такого подхода к рисованию в том, что позволяет  
    //рисовать на одном контексте несколькими объектами класса QPainter  
    //Для этого указатель на контекст передаем методу begin():  
    painter.begin(this);  
    //настраиваем цвета, шрифты и прочее для рисования  
    //рисуем  
    painter.end();//при использовании метода begin() нужно  
    //по окончании работы с контекстом вызвать метод  
    //QPainter::end(), чтобы отсоединить установленную этим методом  
    //связь с контекстом рисования, давая возможность для рисования другому объекту
```

QPainter. Шаг 2. Выбор инструментов рисования и заливки

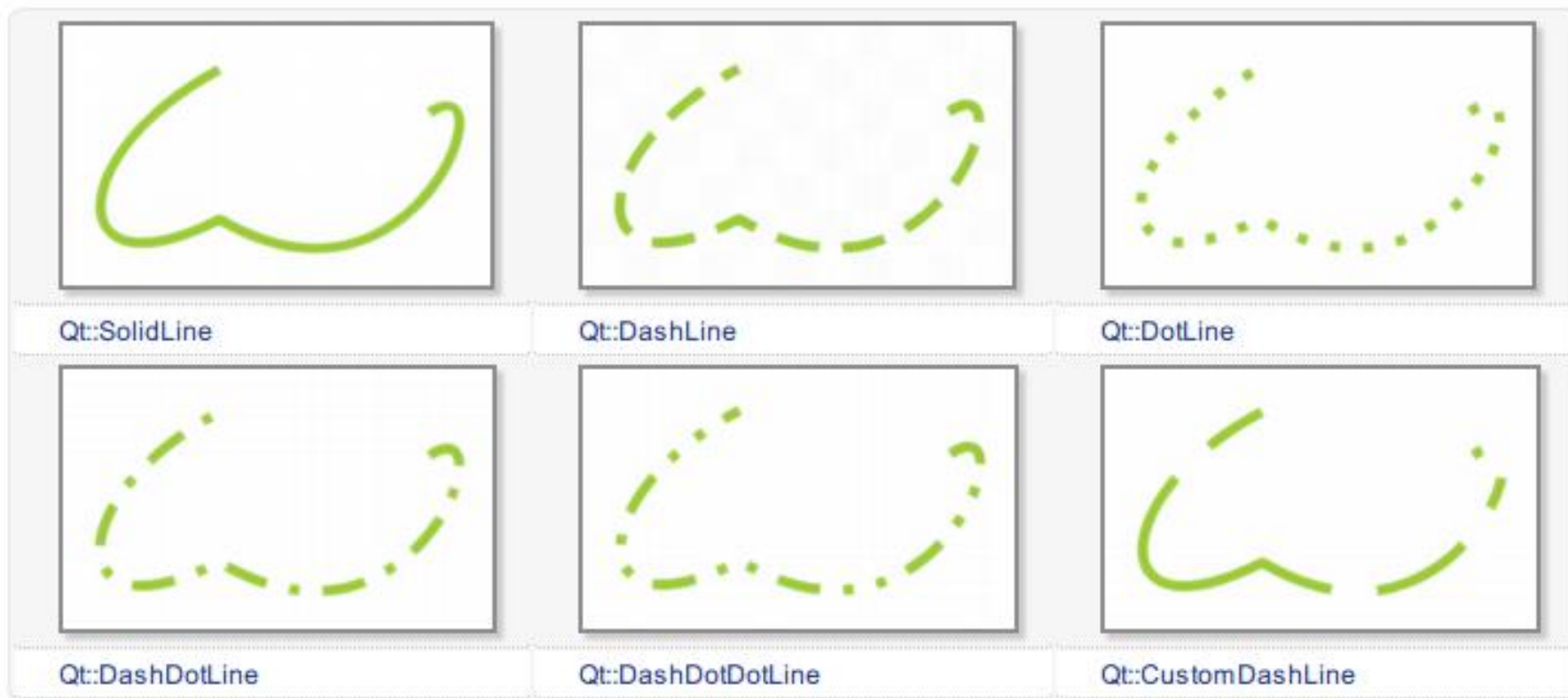
Для отрисовки контурных линий фигуры используется перо: QPen;
Для заполнения непрерывных контуров – кисть: QBrush.



QPainter. Шаг 2. QPen

Класс QPen определяет, как должен QPainter рисовать линии и контуры фигур. Установка пера – `setPen()`

Атрибуты: цвет (`setColor()`), толщина (`setWidth()`), стиль (`setStyle()`).

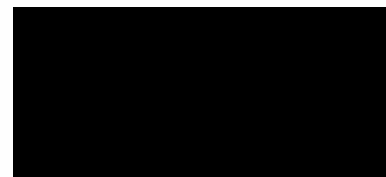


QPainter. Шаг 2. QBrush

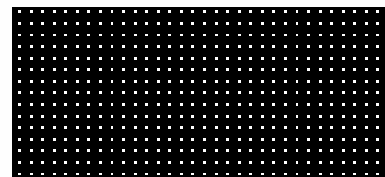
Класс QBrush задает образец заполнения фигур, рисуемых с помощью QPainter.

Атрибуты: цвет (setColor(), setTexture()), образец заливки (setStyle())

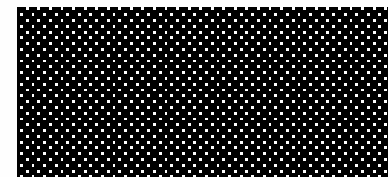
Образцы заливки:



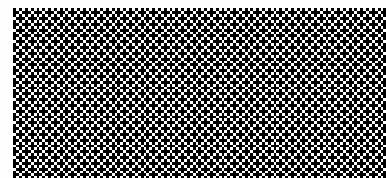
Qt::SolidPattern



Qt::Dense1Pattern



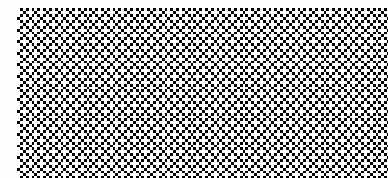
Qt::Dense2Pattern



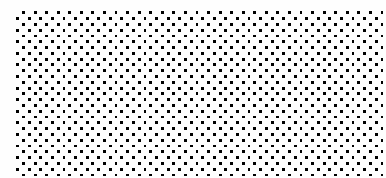
Qt::Dense3Pattern



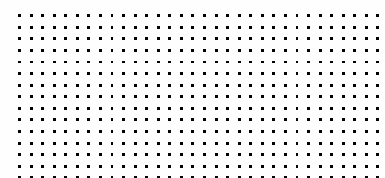
Qt::Dense4Pattern



Qt::Dense5Pattern



Qt::Dense6Pattern



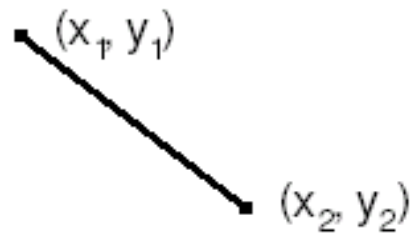
Qt::Dense7Pattern



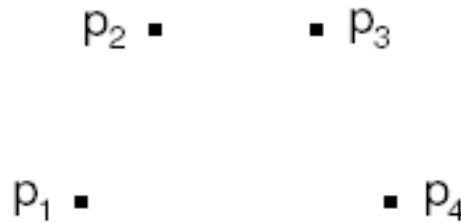
Qt::NoBrush

QPainter. Шаг 3. Рисование.

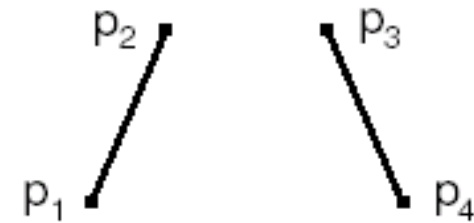
QPainter может нарисовать всё, начиная с простых графических примитивов (представляемых классами QPoint, QLine, QRect, QRegion и QPolygon) и заканчивая сложными фигурами, например, векторными траекториями (QPainterPath).



`drawLine()`



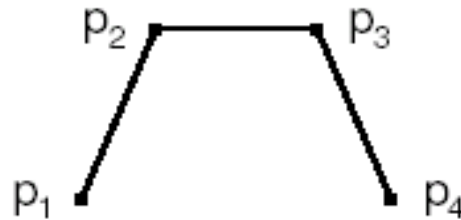
`drawPoints()`



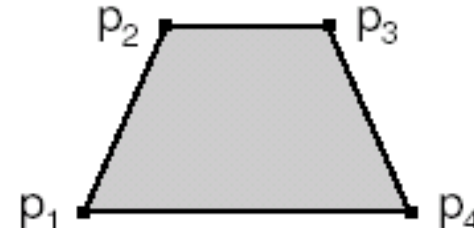
`drawLineSegments()`



`drawCubicBezier()`

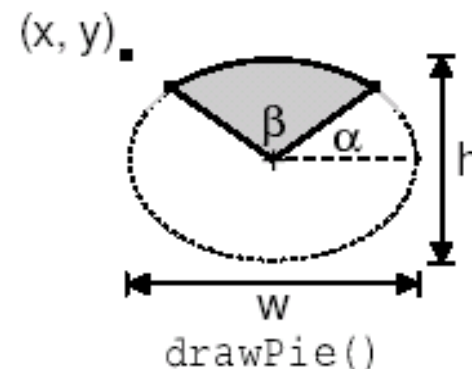
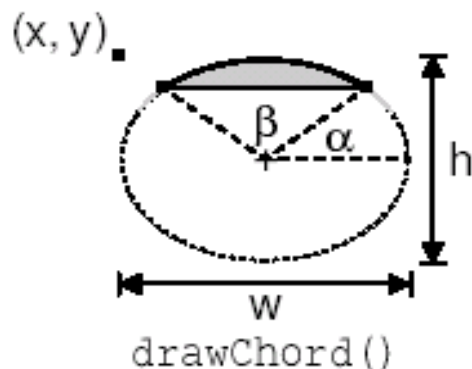
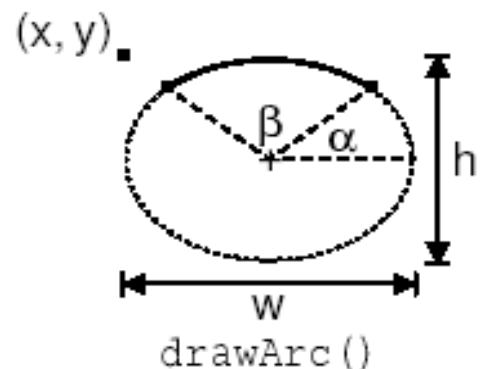
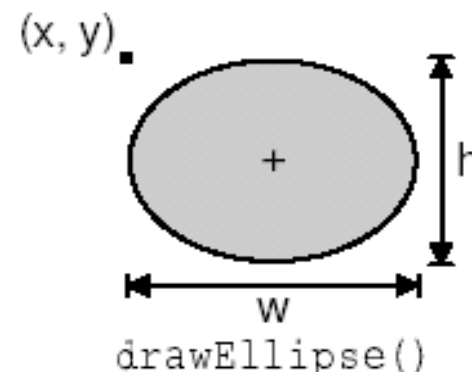
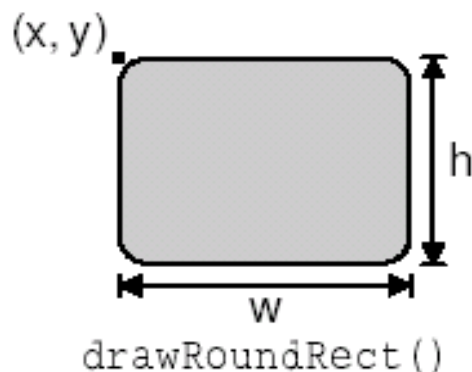
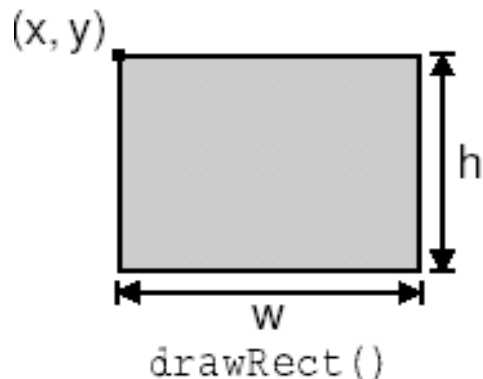


`drawPolyline()`



`drawPolygon()`

QPainter. Шаг 3. Рисование



```
void QPainter::drawArc(QRect, $\alpha \cdot 16, \beta \cdot 16$ )
```

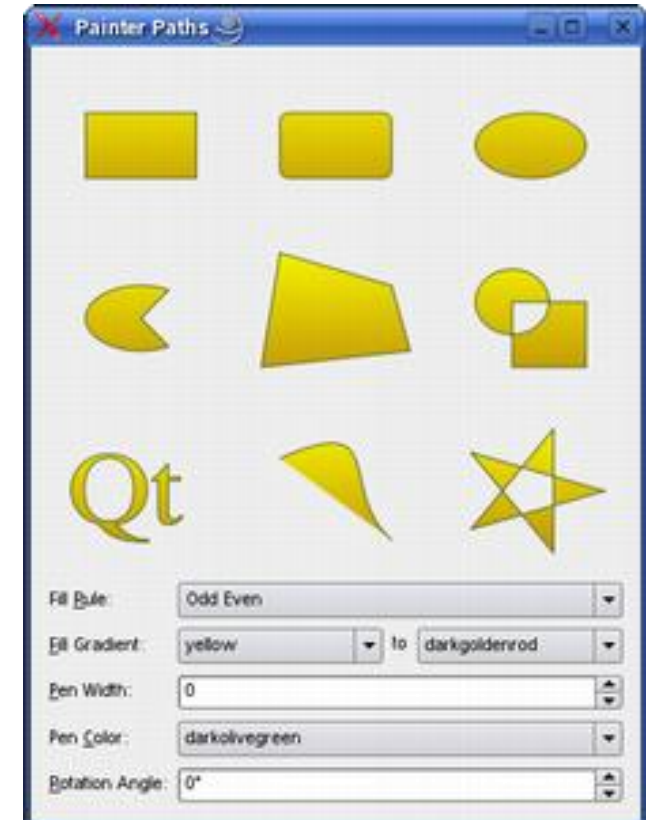
```
void QPainter::drawChord(QRect, $\alpha \cdot 16, \beta \cdot 16$ )
```

```
void QPainter::drawArc(QRect, $\alpha \cdot 16, \beta \cdot 16$ )
```

α – начальный угол,
 β – угол, определяющий размер дуги,
хорды или сектора. ЦМР углов - $\frac{1}{16}$ градуса

QPainter. Шаг 3. Рисование

В Qt векторные траектории представлены классом [QPainterPath](#). [QPainterPath](#) предоставляет контейнер для операций рисования, позволяющий создавать и повторно использовать графические фигуры.



QPainter. Шаг 3. Рисование. Трансформация систем координат.



Смещение `translate()`



Поворот `QPainter::rotate()`



Масштабирование `scale()`



Скос `shear()`

QPainter. Шаг 3. Рисование.

Трансформация систем координат.

1. Смещение – QPainter::translate()

Передвигает систему координат рисунка на заданное смещение dx , dy .

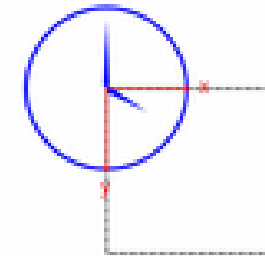
($dx > 0$ – смещение вправо, $dx < 0$ – смещение влево)

($dy > 0$ – смещение вниз, $dy < 0$ – смещение вверх)

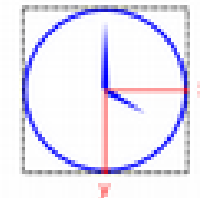
```
void QPainter::translate ( qreal dx, qreal dy )
```

```
void QPainter::translate ( const QPoint & offset )
```

```
void QPainter::translate ( const QPointF & offset )
```



Nop



Translate

QPainter. Шаг 3. Рисование.

Трансформация систем координат.

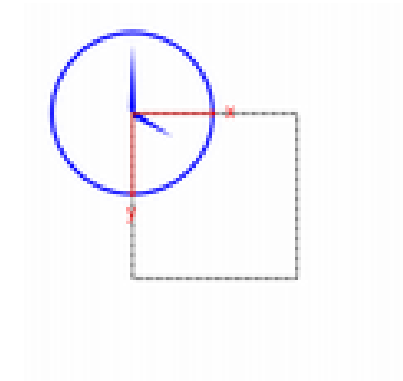
2. Поворот – QPainter::rotate()

Поворачивает изображение на указанный угол (angle).

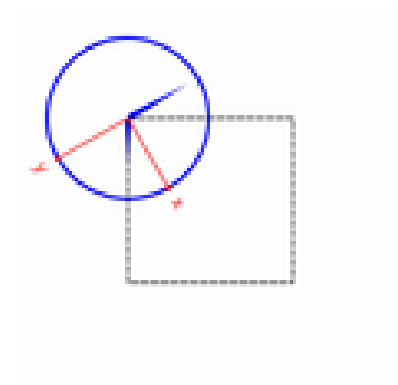
`angle > 0` – поворот по часовой стрелке

`angle < 0` – поворот против часовой стрелки

```
void QPainter::rotate ( qreal angle )
```



Nop



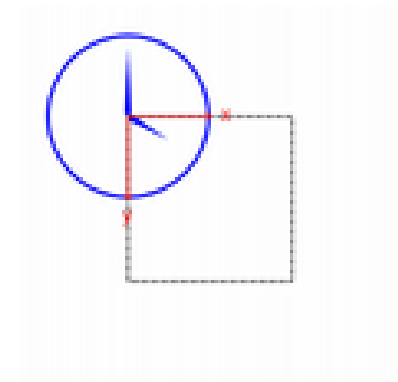
Rotate

QPainter. Шаг 3. Рисование.

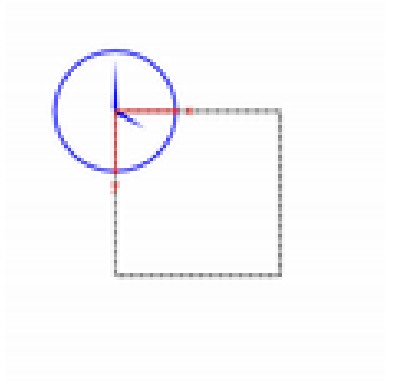
Трансформация систем координат

3. Масштабирование – QPainter::scale()

```
void QPainter::scale ( qreal sx, qreal sy )
```



Nop



Scale

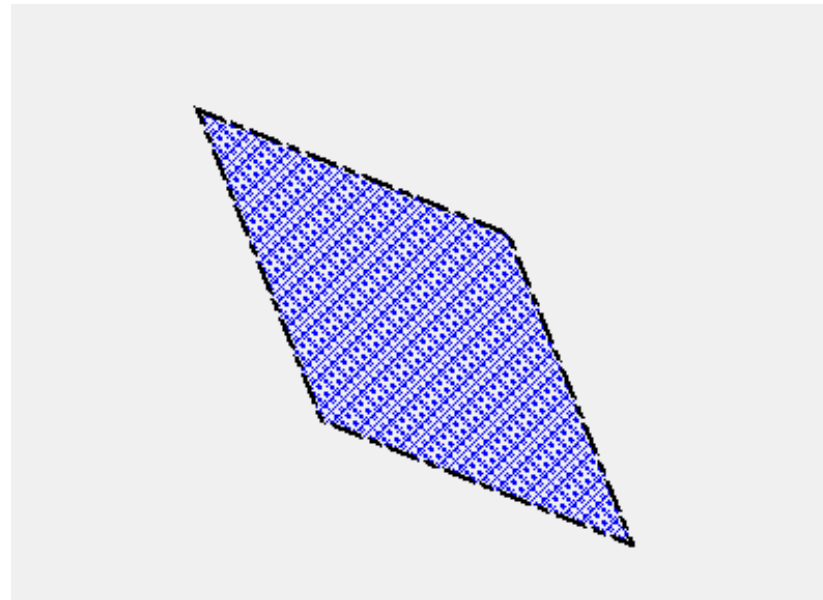
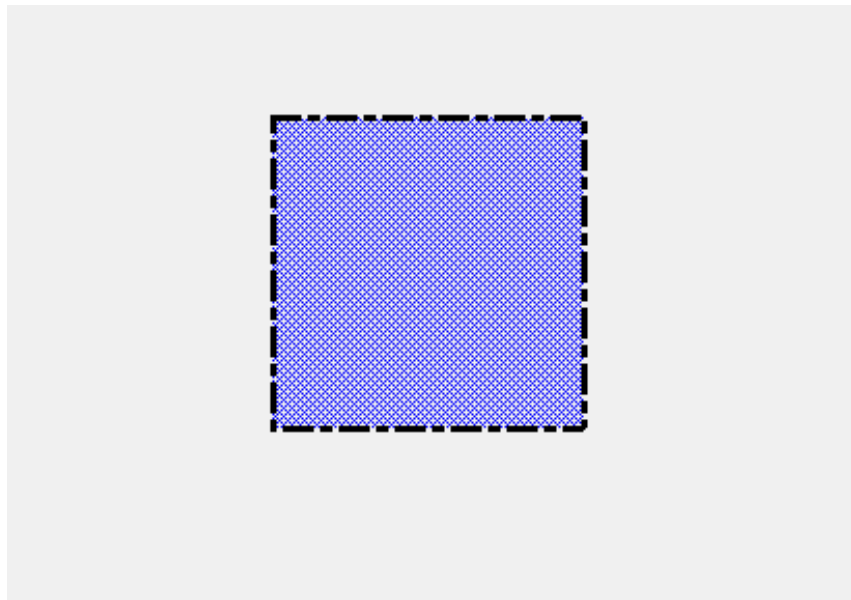
QPainter. Шаг 3. Рисование.

Трансформация систем координат

4. Скос – QPainter::shear()

*Первый параметр задает сдвиг по вертикали, а второй по горизонтали.
Выполняет скос (искажение СК)*

```
void QPainter::shear ( qreal sh, qreal sv )
```



QPainter. Шаг 3. Рисование.

Трансформация систем координат

5. Трансформационные матрицы. setTransform()

Трансформационная матрица QTransform

m11	m12	m13
m21	m22	m23
m31 dx	m32 dy	m33

Формула расчета новых координат:

$$\begin{cases} x' = m11x + m21y + dx \\ y' = m22y + m12x + dy \end{cases}$$

QPainter. Шаг 3. Рисование.

Трансформация систем координат

5. Трансформационные матрицы. `setTransform()`

Элемент матрицы	Смещение	Поворот	Скос	Масштабирование
m11	1	$\cos(a)$	1	Dx
m12	0	$\sin(a)$	Dx	0
m21	0	$-\sin(a)$	Dy	0
m22	1	$\cos(a)$	1	Dy
dx	dx	0	0	0
dy	dy	0	0	0

QPainter. Шаг 3. Рисование.

Трансформация систем координат

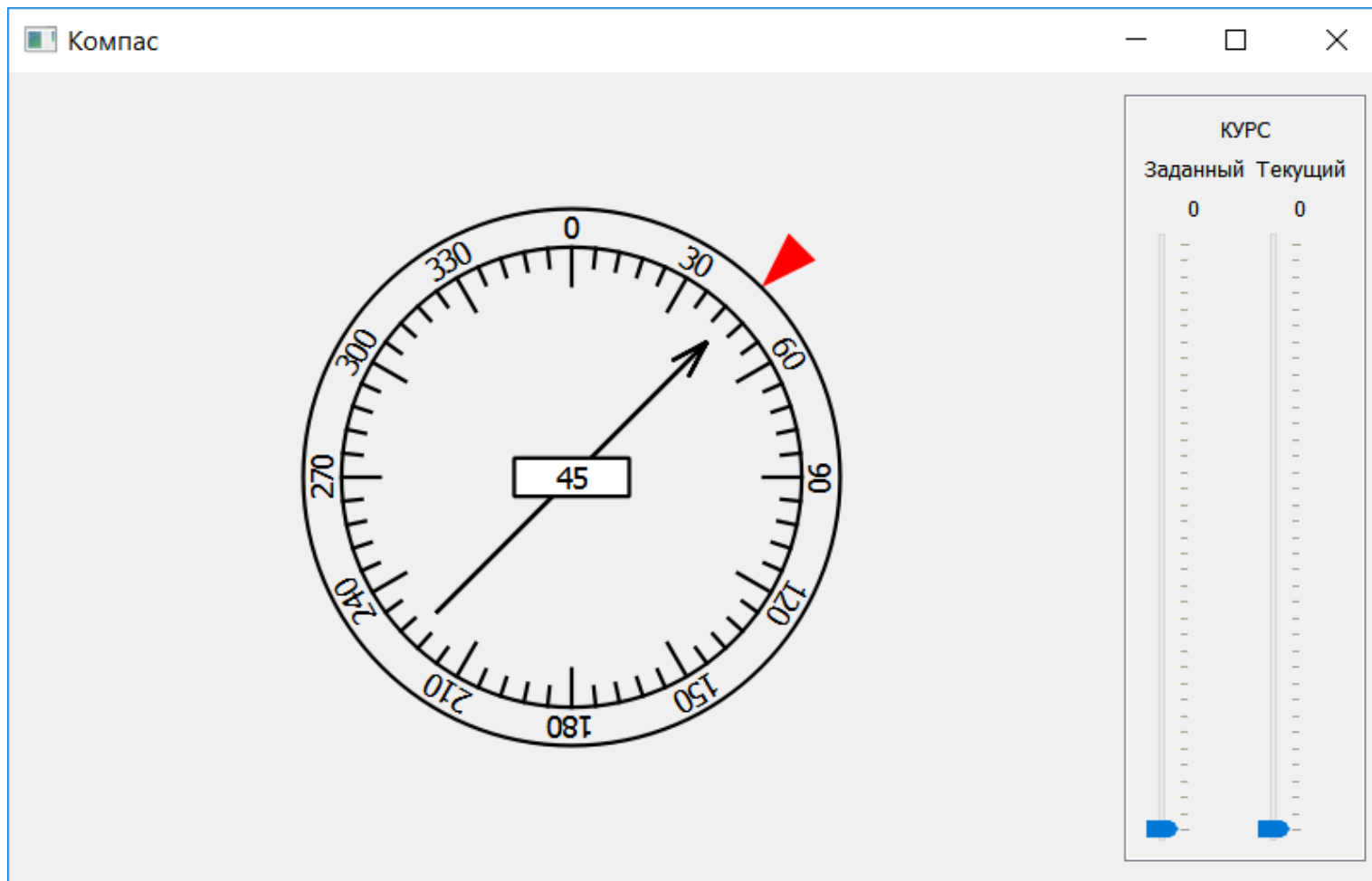
5. Трансформационные матрицы. setTransform()

```
QTransform translationTransform(1, 0, 0, 1, 50.0, 50.0);  
QTransform rotationTransform(cosa, sina, -sina, cosa, 0, 0);  
QTransform scalingTransform(0.5, 0, 0, 1.0, 0, 0);  
QTransform transform;  
transform = scalingTransform * rotationTransform *  
translationTransform;  
QPainter painter(this);  
painter.setPen(QPen(Qt::blue, 1, Qt::DashLine));  
painter.drawRect(0, 0, 100, 100);  
painter.setTransform(transform);
```

QPainter. Важные методы.

Метод	Описание
[slot] QWidget::update()	Этот метод не является методом класса QPainter, но тем не менее очень полезен для нас при отрисовке объектов, кроме того это слот. update() отправляет событие QPaintEvent, которое обрабатывается функцией paintEvent() в которой происходит отрисовка QPainter. При вызове функции update Qt стирает все, что было нарисовано на виджете ранее и затем вызывает функцию paintEvent()
save()	Сохраняет текущее состояние объекта QPainter (помещает состояние в стек). save() должно сопровождаться соответствующим restore(). restore () освобождает стек. Удобно использовать перед различными трансформациями СК для размещения рисунков.
restore()	Восстанавливает текущее состояние живописца (извлекает сохраненное ранее состояние из стека).
begin()	Начинает процесс рисования на устройстве, указатель на которое передан в параметрах функции. (В один момент времени на устройстве рисования можно рисовать только одним объектом QPainter).
end()	Заканчивает рисование. Все используемые для рисования ресурсы освобождаются.

Практическая часть 1. Рисуем компас!



Практическая часть 1. Рисуем компас!

Структура проекта:

1. Создать класс компаса* (наследник от QFrame)
 - 1.1. и к нему можно подключить форму (файл .ui) по желанию;
2. Создать класс виджета**, на котором будут размещены компас и задатчики курса. (по желанию добавить форму к проекту)
3. main.cpp

*В моем проекте класс компаса назван PicFrame

**класс виджета с задатчиками и компасом CompassForm
(не ищите логики в названиях))))

Практическая часть 1. Рисуем компас!

В классе отрисовки компаса:

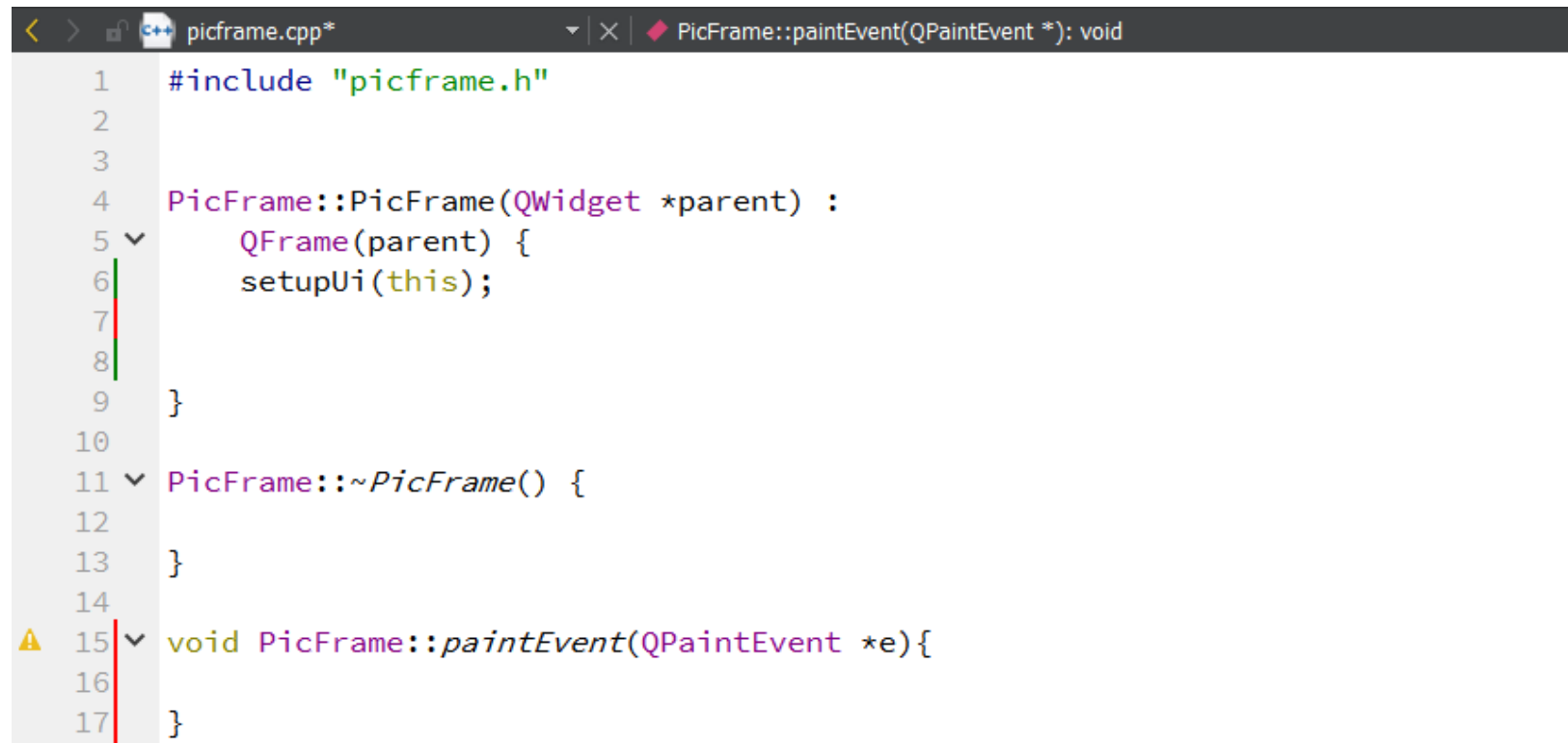
Подключаем форму к проекту (по желанию) и объявляем функцию `paintEvent()`:

```
< > h picframe.h* paintEvent(QPaintEvent *): void # L
1  #ifndef PICFRAME_H
2  #define PICFRAME_H
3
4  #include <QFrame>
5  #include <QPainter>
6
7  #include "ui_picframe.h"
8
9
10 class PicFrame : public QFrame, public Ui::PicFrame {
11     Q_OBJECT
12
13 public:
14     explicit PicFrame(QWidget *parent = 0);
15     virtual ~PicFrame();
16
17 protected:
18     //переопределяем функцию paintEvent для отрисовки нашего безобразия
19     void paintEvent(QPaintEvent *e);
20
21 };|
```

Picframe.h

Практическая часть 1. Рисуем компас!

В исходный файл класса добавляем предварительную реализацию



```
< > picframe.cpp* | PicFrame::paintEvent(QPaintEvent *): void
1  #include "picframe.h"
2
3
4  PicFrame::PicFrame(QWidget *parent) :
5  ▼    QFrame(parent) {
6      setupUi(this);
7
8
9  }
10
11 ▼ PicFrame::~~PicFrame() {
12
13 }
14
15 ▼ void PicFrame::paintEvent(QPaintEvent *e){
16
17 }
```

Picframe.cpp

Практическая часть 1. Рисуем компас!

Класс итоговой формы отрисовки задающих слайдеров и компаса. Compassform.h

```
< > compassform.h* hlay: QHBoxLayout *
1  #ifndef COMPASSFORM_H
2  #define COMPASSFORM_H
3
4  #include <QWidget>
5  #include <QHBoxLayout>
6
7  #include "ui_compassform.h"
8  #include "picframe.h"
9
10
11 class CompassForm : public QWidget, public Ui::CompassForm {
12     Q_OBJECT
13
14 public:
15     explicit CompassForm(QWidget *parent = 0);
16     ~CompassForm();
17
18 private:
19     //создадим указатель на объект класса, рисующего наш компас
20     PicFrame * compass;
21     //создадим горизонтальный компоновщик для размещения виджетов
22     QHBoxLayout *hlay;
23
24 };
25
26 #endif // COMPASSFORM_H
27
```


Практическая часть 1. Рисуем компас!

Класс итоговой формы отрисовки задающих слайдеров и компаса. Compassform.cpp

```
1  #include "compassform.h"
2
3
4  CompassForm::CompassForm(QWidget *parent) :
5  ▼    QWidget(parent) {
6
7      setupUi(this);
8
9  }
10
11
12  ▼ CompassForm::~CompassForm() {
13
14  }
15
```

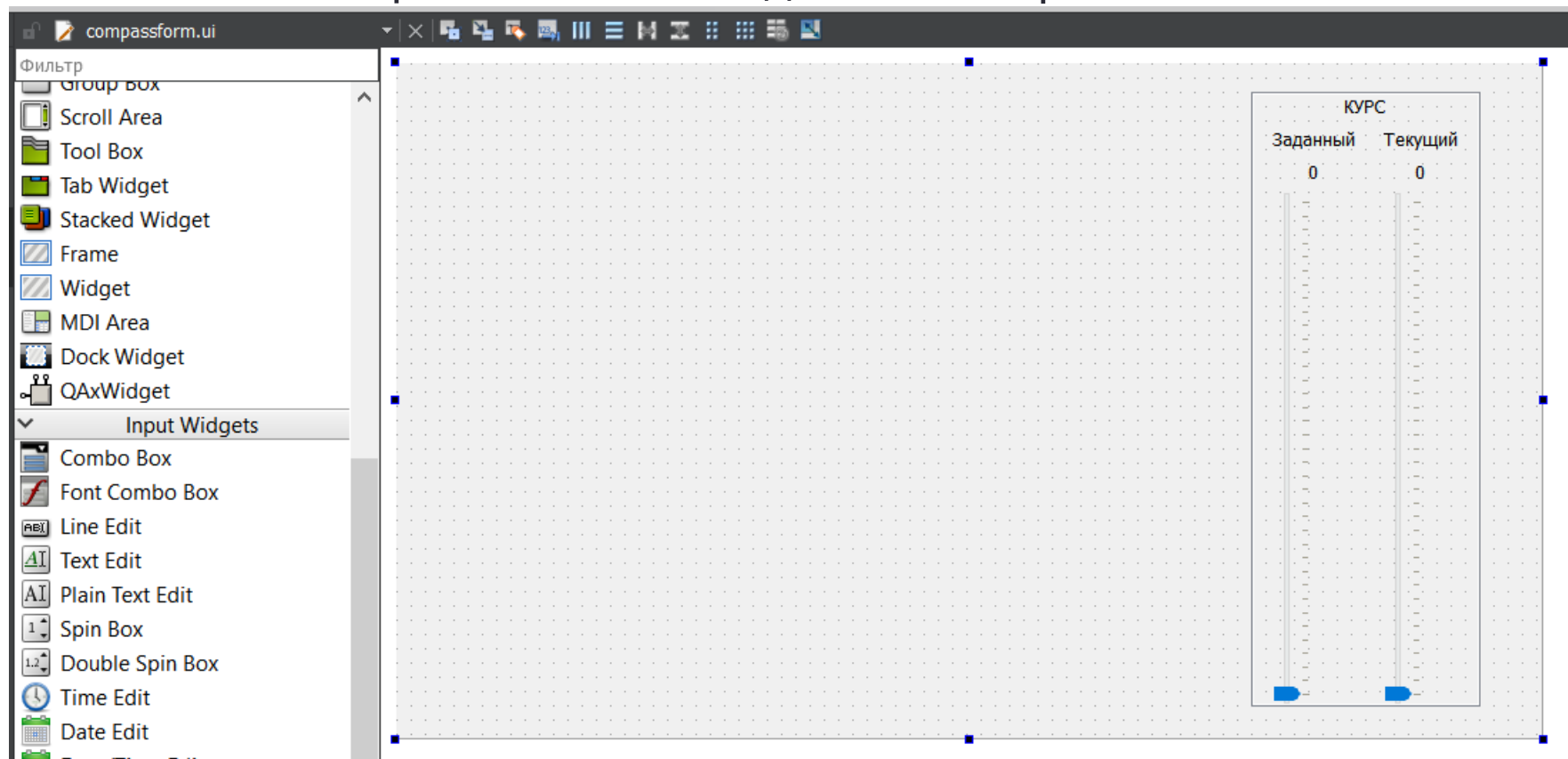
Практическая часть 1. Рисуем компас!

Класс итоговой формы отрисовки задающих слайдеров и компаса. Compassform.cpp

```
1  #include "compassform.h"
2
3
4  CompassForm::CompassForm(QWidget *parent) :
5  ▼    QWidget(parent) {
6
7      setupUi(this);
8
9  }
10
11
12  ▼ CompassForm::~CompassForm() {
13
14  }
15
```

Практическая часть 1. Рисуем компас!

Форма итогового виджета – compassform.ui



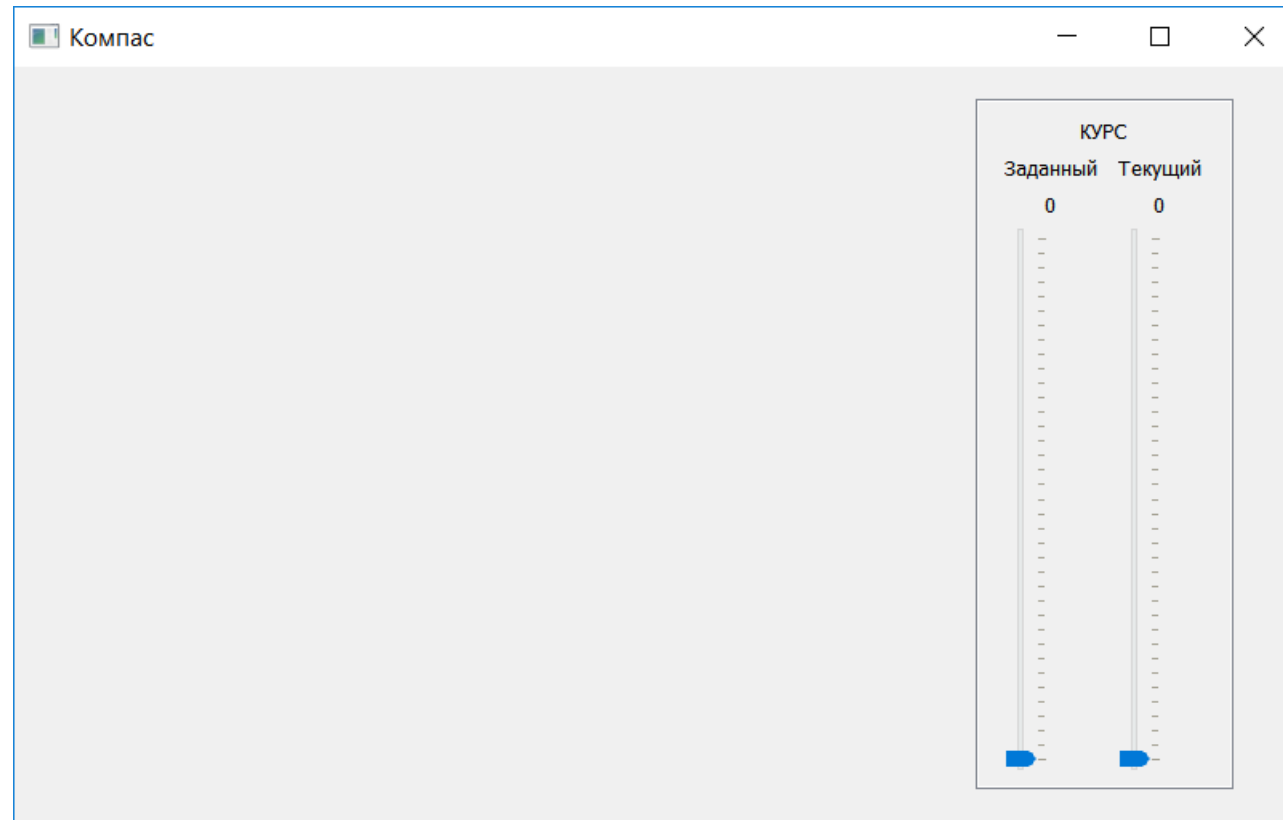
Практическая часть 1. Рисуем компас!

Класс итоговой формы отрисовки задающих слайдеров и компаса. Compassform.cpp

```
LED/main.cpp
1  #include "widget.h"
2  #include "picframe.h"
3  #include "compassform.h"
4  #include <QApplication>
5
6  int main(int argc, char *argv[]) {
7      QApplication a(argc, argv);
8      CompassForm comp;
9      comp.setWindowTitle("Компас");
10     comp.show();
11     return a.exec();
12
13 }
14
15
```

Практическая часть 1. Рисуем компас!

Компилируем и наслаждаемся промежуточным результатом:



Практическая часть 1. Рисуем компас!

В picframe.h:

```
//переопределяем функцию paintEvent для отрисовки нашего безобразия
void paintEvent(QPaintEvent *e);
```

```
private:
```

```
float yaw; //значение курса аппарата
float yawDesirable; //значение заданного курса аппарата
//зададим координаты точек стрелки компаса
//пусть центр стрелки находится в начале координат (0,0)
```

```
QPoint arrowCompass[6]= {
    QPoint (0,50),
    QPoint (0,-50),
    QPoint (3,-40),
    QPoint (0,-50),
    QPoint (-3,-40),
    QPoint (0,-50)
};
```

```
//зададим координаты точек внешней стрелки
//отвечающей за заданное положение ПА
```

```
QPoint arrowDesirable [4] = {
    QPoint (0, -70),
    QPoint (5, -85),
    QPoint (-5, -85),
    QPoint (0,-70)
};
```

QPoint – класс Qt для работы с точками (он ничего не рисует, просто содержит координаты точек, которые потом будут отрисованы),

По точкам, которые мы задаем в этом массиве (с учетом последовательности) затем можно будет нарисовать стрелку. Можете попробовать нарисовать ее на бумаге, с учетом того, что центр координат (0,0) находится в левом верхнем углу виджета, а положительное направление оси X – вправо, а оси Y – вниз.

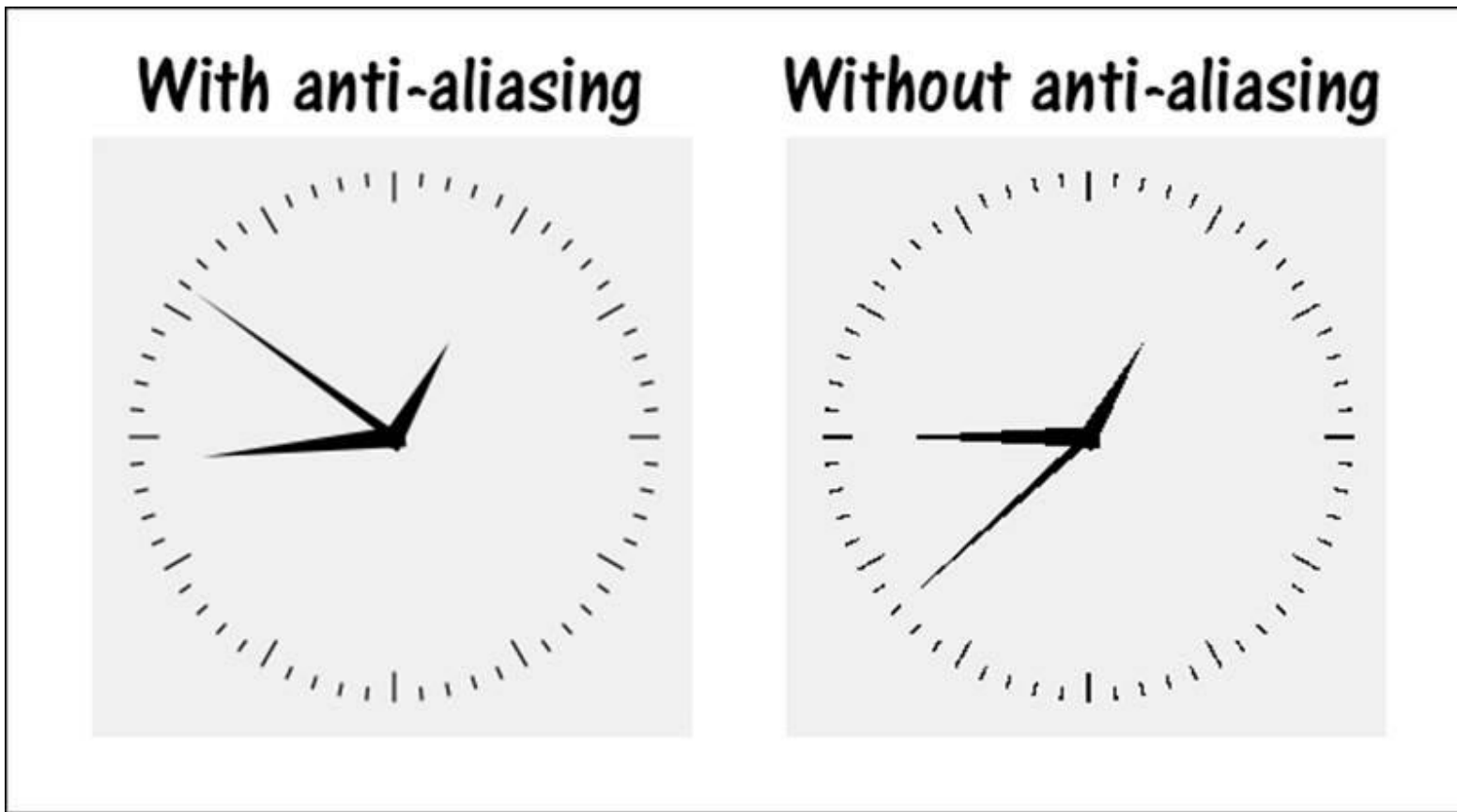
Практическая часть 1. Рисуем компас!

В picframe.cpp:

```
< > picframe.cpp* PicFrame::paintEvent(QPaintEvent *): void # Line: 43, Col: 2
17
18 void PicFrame::paintEvent(QPaintEvent *e){
19     //При создании класса в качестве параметра передаем указатель
20     //на тот объект, на котором мы рисуем
21     QPainter painter(this);
22     //настраиваем инструменты рисования (цвета, шрифты, толщины линий и прочее)
23     QFont font; //создадим объект класса шрифта
24     //включим технику сглаживания (Anti-aliasing), устраняющую ступенчатость рисунка
25     painter.setRenderHint(QPainter::Antialiasing, true);
26     //Транслируем систему координат так, что точка (0, 0) располагается в
27     //центре виджета, вместо того чтобы быть в верхнем левом углу.
28     //(функции width() и height()) как раз возвращают нам значения ширины и длины для
29     //виджета)
30     painter.translate(width() / 2, height() / 2);
31     //Мы также масштабируем систему координат на величину side / 100,
32     //где side - ширина либо высота виджета, которая меньше по величине.
33     //Мы хотим, чтобы часы были квадратными даже если устройство не квадратное.
34     //Это даст нам квадратную область 200 x 200 с началом координат (0, 0) по центру,
35     //в которой мы можем рисовать. То, что мы рисуем будет показано в наибольшем
36     //возможном квадрате, который помещается в виджет.
37     int side = qMin(width(), height());
38     painter.scale(side / 200.0, side / 200.0);
39     //мы закончили настройку и можем приступить к рисованию
40     //чтобы сохранить текущие параметры (положения координат, масштаб,
41     //а кроме того цвета, настройки пера и прочее вызываем метод save(),
42     //который сохраняет все эти параметры в стек)
43     painter.save();|
```

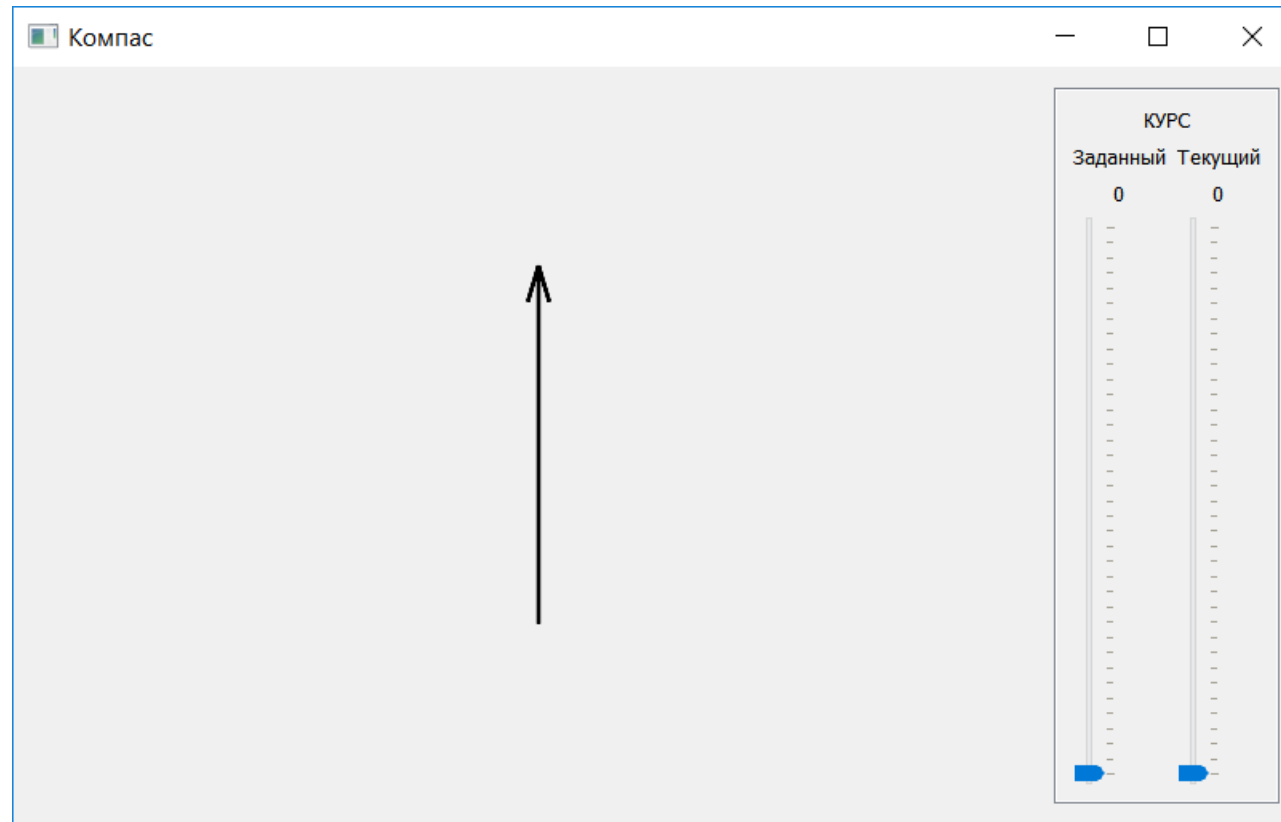
Практическая часть 1. Рисуем компас!

Ремарка по поводу включения режима сглаживания Anti-aliasing:
`painter.setRenderHint(QPainter::Antialiasing,true);`



Практическая часть 1. Рисуем компас!

Компилируем и наслаждаемся промежуточным результатом:



Практическая часть 1. Рисуем компас!

В `picframe.cpp`:

```
44 //функция rotate() вращает нашу СК таким образом, что теперь направление оси Y повернуто
45 // относительно вертикального для нас направления на угол yaw, переданный в качестве
46 //параметра в эту функцию. А мы ранее задали точками стрелку такой, чтобы она
47 //совпадала по направлению с осью OY
48 painter.rotate(yaw);
49 //Мы нарисовали стрелку компаса повернув систему координат и вызвав
50 //QPainter::drawConvexPolygon(). Спасибо вращению, она нарисована в
51 //правильном направлении.
52 painter.drawConvexPolygon(arrowCompass, 6);|
53 painter.restore();
54 //Используем функции save() и restore() чтобы сохранить и восстановить матрицу
55 //преобразований до и после вращения, поскольку нам нужно разместить
56 //стрелку заданного значения компаса без учёта любых предыдущих вращений.
```

Практическая часть 1. Рисуем компас!

А давайте-ка проверим, работает ли наша стрелка как задумано..

Для этого нужно добавить в наш класс отрисовки компаса слот, который будет принимать значение курса, на которое наша стрелка должна повернуться.

Для этого:

1. Добавим новый слот в объявлении класса в `picframe.h`:

```
43  
44 public slots:  
45     void setYaw(int yaw);  
46 |
```

2. Добавим реализацию в классе `picframe.cpp`:

```
108 ▼ void PicFrame::setYaw(int yawNew){  
109     yaw=yawNew;  
110     //благодаря вызову функции update() будет высылаться событие QPaintEvent  
111     //с виджета стирается все, что было нарисовано ранее и вызывается наша  
112     //функция paintEvent(), где как раз отрисовывается стрелка с новым значением yaw  
113     update();  
114 }  
115
```

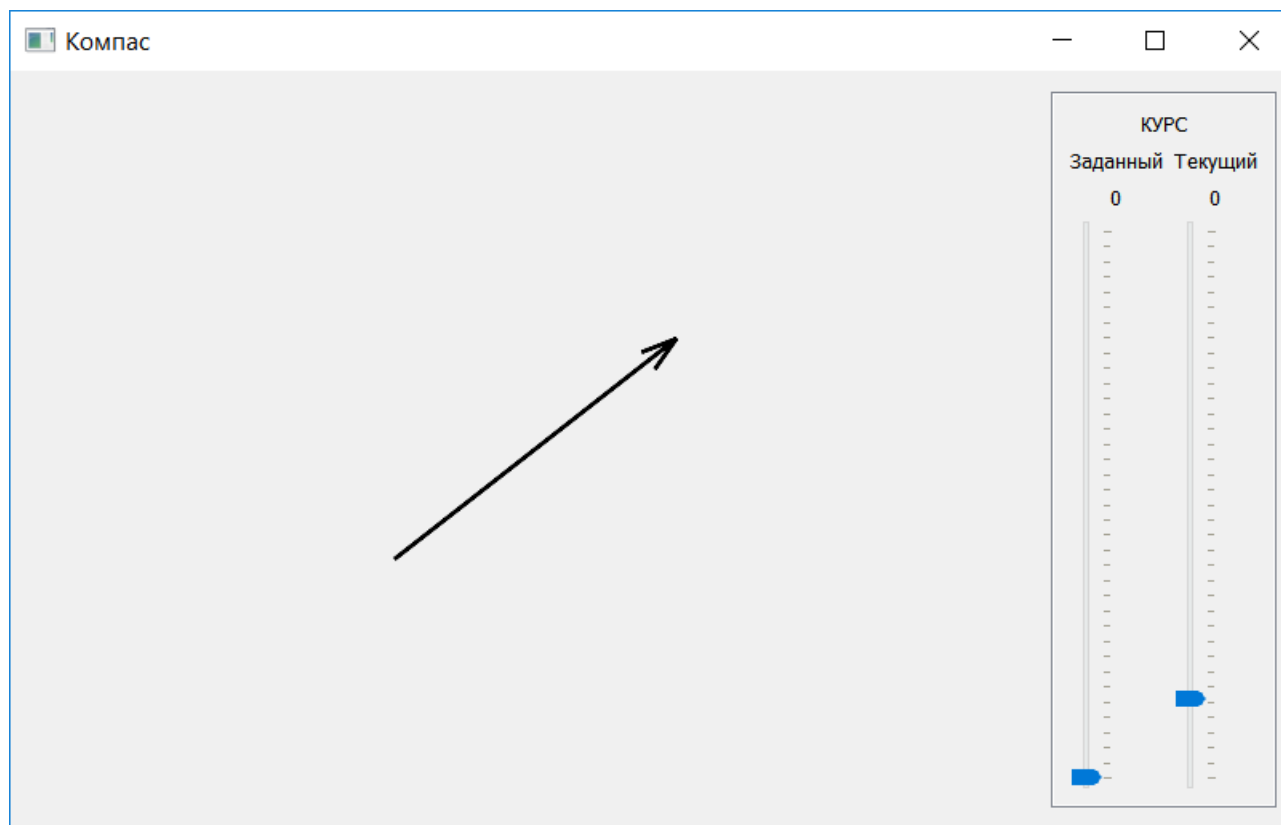
Практическая часть 1. Рисуем компас!

В классе CompassForm соединим сигнал о том, что изменилось заданное значение на слайдере со слотом объекта класса отрисовки компаса, который изменит положение стрелки:

```
compassform.cpp*  CompassForm::CompassForm(QWidget *)
1  #include "compassform.h"
2
3
4  CompassForm::CompassForm(QWidget *parent) :
5  QWidget(parent) {
6
7      setupUi(this);
8      hlay = new QHBoxLayout;
9      compass = new PicFrame;
10     compass->setSizePolicy(QSizePolicy::Ignored, QSizePolicy::Ignored);
11     hlay->addWidget(compass);
12     hlay->addWidget(frameSetYaw);
13     this->setLayout(hlay);
14
15     connect (sldYawCurrent, SIGNAL(valueChanged(int)),
16             compass, SLOT(setYaw(int)));
17
18
19 }
20
21 CompassForm::~CompassForm() {
22
23 }
24
```

Практическая часть 1. Рисуем компас!

Компилируем и проверяем, что стрелка реально крутится при изменении положения слайдера:



Практическая часть 1. Рисуем компас!

В picframe.cpp:

```
57 //снова сохраняем текущие параметры рисунка
58 painter.save();|
59 //рисуем стрелку для вывода заданного значения курса
60 painter.setPen(Qt::NoPen); //Перо можно устанавливать прозрачным Qt::NoPen
61 //если нужно залить контур, но не нужно, чтобы контур был отрисован
62 painter.setBrush(Qt::red); //Зададим цвет заливки сплошных контуров для
63 //этой сессии рисования (работает между прошлым save() и следующим restore())
64 painter.rotate (yawDesirable); //поворачиваем СК в соответствии с желаемым
65 //значением курса
66 //рисуем полигон методом drawConvexPolygon(массив с точками, кол-во точек на отрисовку);
67 painter.drawConvexPolygon(arrowDesirable, 4);
68 painter.restore();
69 painter.save();
70 //рисуем контур нашего компаса
71 painter.drawEllipse(-60,-60,120,120);
72 painter.drawEllipse(-70,-70,140,140);
```

Практическая часть 1. Рисуем компас!

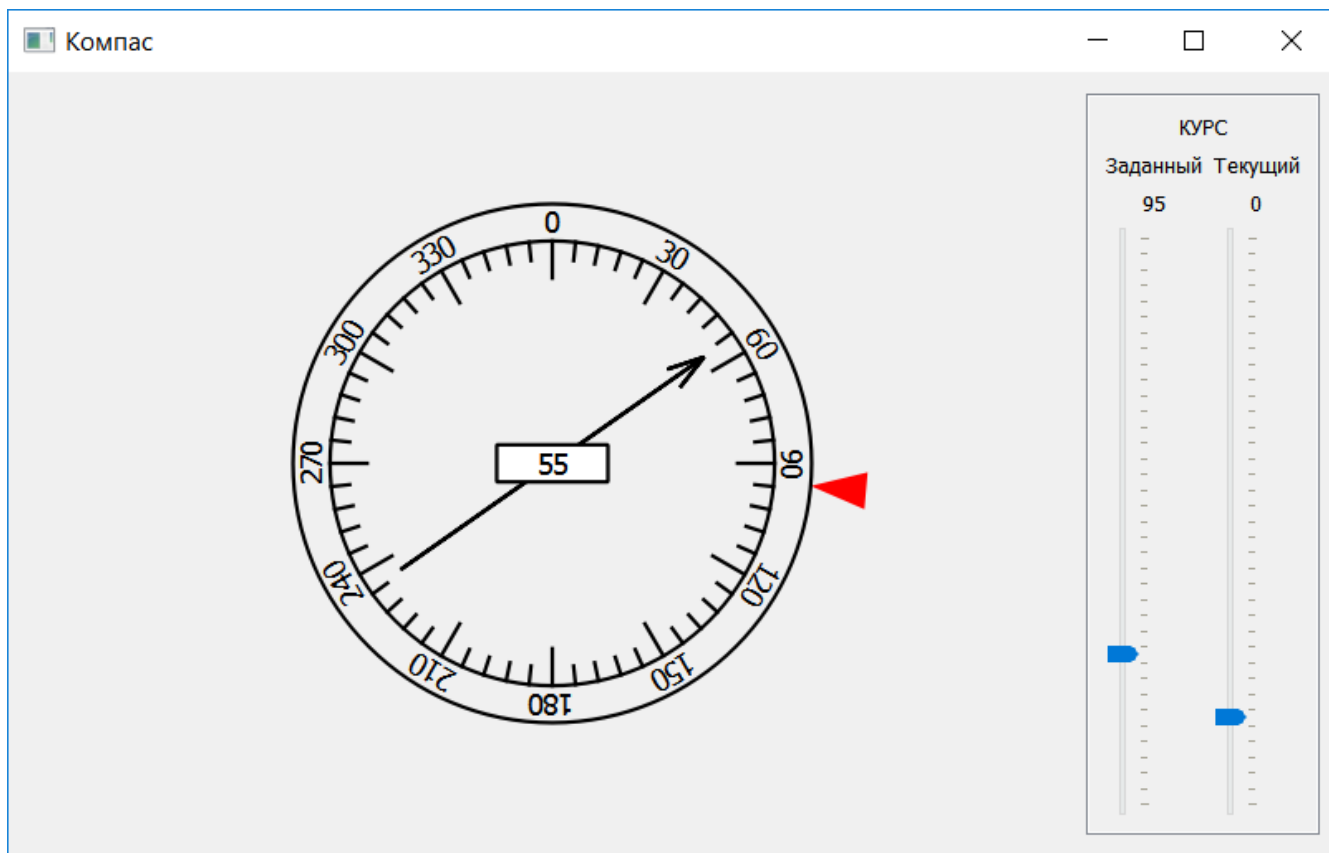
В picframe.cpp:

```
73
74 //рисуем шкалу компаса
75 for (int j = 0; j < 60; ++j) {
76     if ((j % 5) != 0)
77         painter.drawLine(0, -55, 0, -60);
78     else {
79         painter.drawLine(0, -50, 0, -60);
80
81         font.setPointSize(5);
82         painter.setFont(font);
83         painter.drawText(-20, -70, 40, 40,
84                             Qt::AlignHCenter | Qt::AlignTop,
85                             QString::number(j*6));
86     }
87     painter.rotate(6.0);
88 }
89
90 painter.setBrush(Qt::white);
91 painter.drawRect(-15,-5,30,10);
92 font.setPointSize(10);
93 painter.drawText(QRect(-20,-10,40,20), Qt::AlignCenter, QString::number(yaw));
94
95 painter.restore();
96
```

Практическая часть 1. Рисуем компас!

Также необходимо проверить работу новой стрелки, для этого надо добавить новый слот для установки желаемого значения курса, и соединить сигналы и слоты в классе итоговой отрисовки виджетов.

В результате:



ДЗ.

- Создать свой виджет отображения дифферента и крена аппарата -> гиригоризонт.



ДЗ

Посмотреть как работает гиригоризонт можно
здесь:

<https://www.youtube.com/watch?v=k223jAq2Yu8>

