


# Програмування GUI

На основі мови C++ та фреймворку Qt

## Лекція 8

# Основи графіки в Qt

A series of small, light green decorative icons are placed along the left side of the text block. From top to bottom, they include: a tree diagram, a clipboard with a checkmark, a computer monitor, a skull and crossbones, and a gear.

Малювання в Qt здійснюється за допомогою об'єкта класу `QPainter` (малювальник). При малюванні на поверхні об'єкту малювальник використовує поточні настройки малювання - товщина і колір ліній, параметри заливки фігур, параметри шрифту, колір фону і т.д. Змінити поточні настройки можна шляхом створення об'єктів пера (параметри ліній, точок), кисті (параметри заливки) і т.д. з потрібними параметрами, і зробивши їх поточними за допомогою методів класу `QPainter`.



# Клас QPen



Використовується для малювання точок, відрізків, кривих, контуру фігур, тексту.



Конструктор:

```
QPen(const QColor &color),
```

де `color` – задає колір пера.



Після створення пера його можна зробити поточним пером малювальника методом `void setPen(const QPen &pen)` класа `QPainter`.




# Клас QPen

Qt

```
void MyWidget::paintEvent (QPaintEvent *e)
```


```
{  
    QPainter painter;
```

```
    painter.begin(this); //this - адрес об'єкту, на  
    //поверхні якого будемо малювати
```




```
    QPen pen(Qt::red); //червоний колір пера,  
    // за замовчанням чорний
```

```
    painter.setPen(pen); //робимо це перо поточним для  
    //малювальника
```



```
    pen.setWidth(3); // задаємо нову ширину пера=3 пікселя,  
    // за замовчанням =1 піксель
```

```
    pen.setStyle(Qt::DashLine); //задаємо стиль пера,  
    // за замовчанням Qt::SolidLine
```



```
    pen.setColor(QColor(255,255,0)); //задаємо колір пера -  
    //жовтий
```

```
    painter.setPen(pen); //встановлюємо поточне перо  
    //малюємо }
```



# Клас QColor



Використовується для створення об'єкту кольору.

Конструктор:

```
QColor (int r, int g, int b, int  
a=255) ,
```

де  $r, g, b$  – задають відповідно інтенсивність червоної, зеленої та синьої складових кольору, параметр  $a$  – задає прозорість.

Значення параметрів лежат в діапазоні  $[0;255]$ .

Чем більше значення – тим більша інтенсивність.

$a=255$  – непрозорий,  $a=0$  – абсолютно прозорий.

```
QColor cr1 (255, 0, 0); //червоний  
QColor cr2 (0, 0, 255); //синій  
QColor cr3 (255, 255, 255); // білий
```

# Клас QColor

В Qt є 20 «стандартних» кольорів:

`Qt::white` – білий

`Qt::black` – чорний

`Qt::green` – зелений

і т.д.

Також об'єкт кольору можна створити за допомогою конструктора `QColor(QRgb color)`, де тип `QRgb` є аналогом типу `unsigned int`.

Формат змінної `color`: `0xRRGGBB`

`0x` – 16а система числення

`RR` - червоний

`GG` - зелений

`BB` - синій

# Клас QColor

Qt



```
QRgb rgb=0xFF0000;
```

```
rgb=0; //чорний
```



```
rgb=12345678; //світло-коричневий
```

```
pen.setColor(rgb);
```



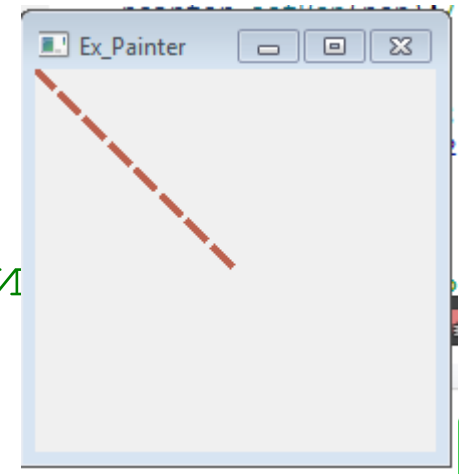
```
//поточний колір малювальника - світло-коричневий
```

```
painter.setPen(pen);
```



```
painter.drawLine(0,0,100,100);
```

```
//малюємо відрізок світло-коричневи
```



# Клас QBrush



Клас `QBrush` використовується для створення об'єкту пензля. Пензель застосовується для зафарбовування області фігур.



Конструктор:

```
QBrush (const QColor &color,  
Qt::BrushStyle style=Qt::SolidPattern)
```



`color` - колір пензля,

`style` - шаблон заливки (за замовчанням-суцільна)



`Qt::BrushStyle:`

`Qt::HorPattern, Qt::VerPattern,`

`Qt::NoBrush` і т.д.



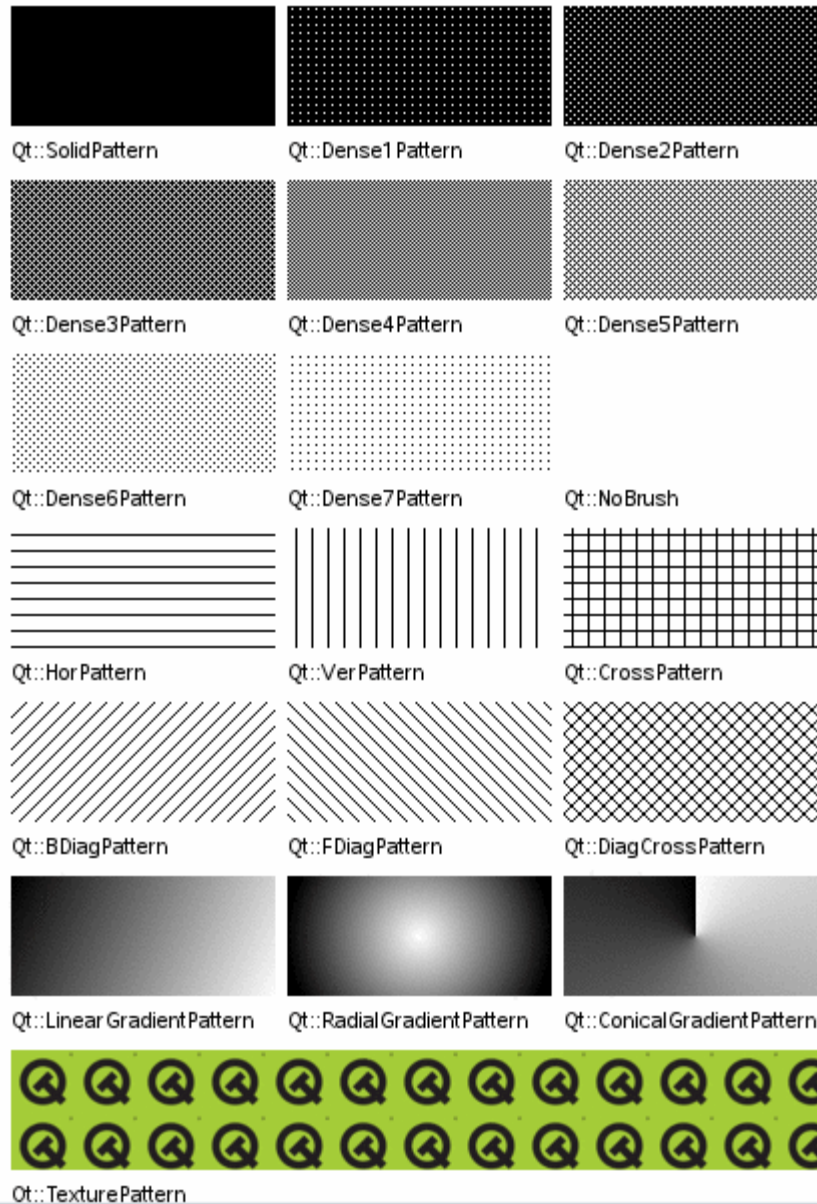
`Qt::NoBrush` – порожній пензель





# Клас QBrush

Qt

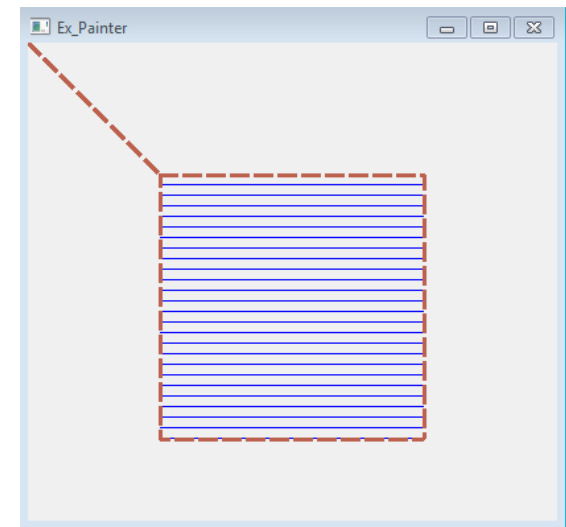


A  
Z  
↓

# Клас QBrush



```
QBrush brush(Qt::blue, Qt::HorPattern);  
painter.setBrush(brush);  
// робимо brush поточним пензлем  
// малювальника  
// за замовчання поточний пензель -  
// Qt::NoBrush  
painter.drawRect(100, 100, 200, 200);
```



# Класи QPoint, QSize, QRect



У процесі малювання зручно використовувати об'єкти класів `QPoint`, `QSize`, `QRect`.



Клас `QPoint` (точка) призначений для створення об'єкта точки в двовимірній системі координат. Створити точку можна за допомогою конструктора `QPoint (int x, int y)`, де  $x, y$  – координати точки.



Методи:



1. `void setX(int x)` – задає нову координату  $x$
2. `void setY(int y)` – задає нову координату  $y$
3. `int x(), int y()` – зчитують поточні координати точки
4. `int &rx(), int &ry()` – повертають посилання на координати точок (з їх допомогою можна зчитувати і змінювати координати)



# Класи QPoint, QSize, QRect



```
QPoint p1(10, 10);  
p1.rx() += 10;  
p1.ry() += 20;  
int x = p1.x(); // x=20  
int y = p1.y(); // y=30
```



Клас `QPoint` містить оператори, що дозволяють виконувати з точками арифметичні операції `+`, `-`, ... і операції порівняння `==`, `!=` і т.д.



```
QPoint p2(30, 30), p3;  
p3 = p1 + p2; // p3 = (50, 60)  
bool ok = p1 == p2; // ok = false
```



# Клас QSize



Аналогічний класу `QPoint`, але призначений для створення об'єкта, що зберігає розміри геометричних фігур, віджетів.



Конструктор:

```
QSize (int w, int h),
```

w – ширина, h – висота.



```
QSize s(10,20);
```

```
int w=s.width(); //w=10
```

```
int h=s.height(); //h=20
```




```
s.rwidth()+=5; // w=15
```

```
s.transpose(); //w=20, h=15 - змінює значення  
місцями
```



# Клас QRect




 Призначений для завдання \ зберігання координат і розмірів геометричних фігур і віджетів.


Конструктори:

 1. QRect (const QPoint & *topLeft*, const QPoint & *bottomRight*)


```
QPoint p1(10,10), p2(30,30);  
QRect rect(p1,p2);
```

 2. QRect (const QPoint & *topLeft*, const QSize & *size*)

```
QSize size(30,30);  
QRect rect2(p1,size);
```

 3. QRect (int *x*, int *y*, int *width*, int *height*)

```
QRect rect3(10,20,30,10);  
QPoint p3=rect3.topLeft(); //p3.x=10, p3.y=20  
QPoint p4=rect3.bottomRight(); //p4.x=40, p3.y=30
```



# Клас QRect



За допомогою методу `bool contains (const QPoint &p, bool proper=false)` можна перевірити, чи міститься точка в прямокутнику. Якщо параметр `proper=false` – буде враховуватися контур прямокутника (кромка), `proper=true` – не буде.

```
QRect r (QPoint(0,0), QPoint(20,20));  
bool c=r.contains(QPoint(10,10)); //c==true
```

Задати нові координати прямокутника можна :

```
r.setSize(QSize(30,30));  
r.setTopLeft(QPoint(5,5));
```

# Режим відображення фону



Режим відображення фону визначає, чи буде використовуватися колір фону для зафарбовування пустот між пунктиром ліній, відрізками в шаблоні несучільного пензля, між символами тексту і т.д.

Для встановлення режиму відображення фону використовується метод `void`

`setBackgroundMode (Qt::BGmode m)` класу `QPainter`.

`Qt::BGmode`:

`Qt::TransparentMode` – задає прозорий режим (колір фону не буде використовуватися), використовується за замовчанням

`Qt::OpaqueMode` – задає непрозорий режим (колір фона буде використовуватися для зафарбовування пустот).



# Режим відображення фону



Колір фону можна встановити методом `void setBackground (const QBrush &b)`, де `b` – задає пензель, який буде використовуватися для заливки порожнин. За замовчуванням використовується пензель білого кольору.

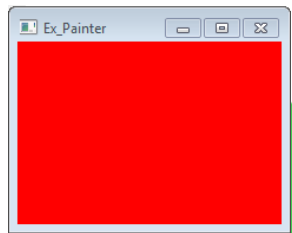
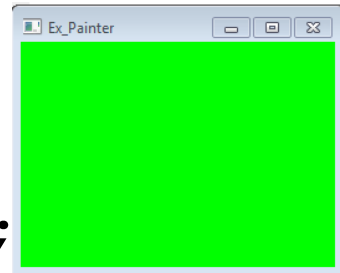
```
painter.setBackgroundMode (Qt::OpaqueMode);  
painter.setBackground (Qt::green);  
// порожнечі будуть зафарбовуватися зеленим суцільним пензлем
```

Зафарбувати повністю клієнтську область віджета кольором фону :

```
painter.eraseRect (0, 0, width (), height ());
```

Зафарбувати повністю клієнтську область віджета довільним пензлем :

```
painter.fillRect (0, 0, width (), height (),  
Qt::red);
```



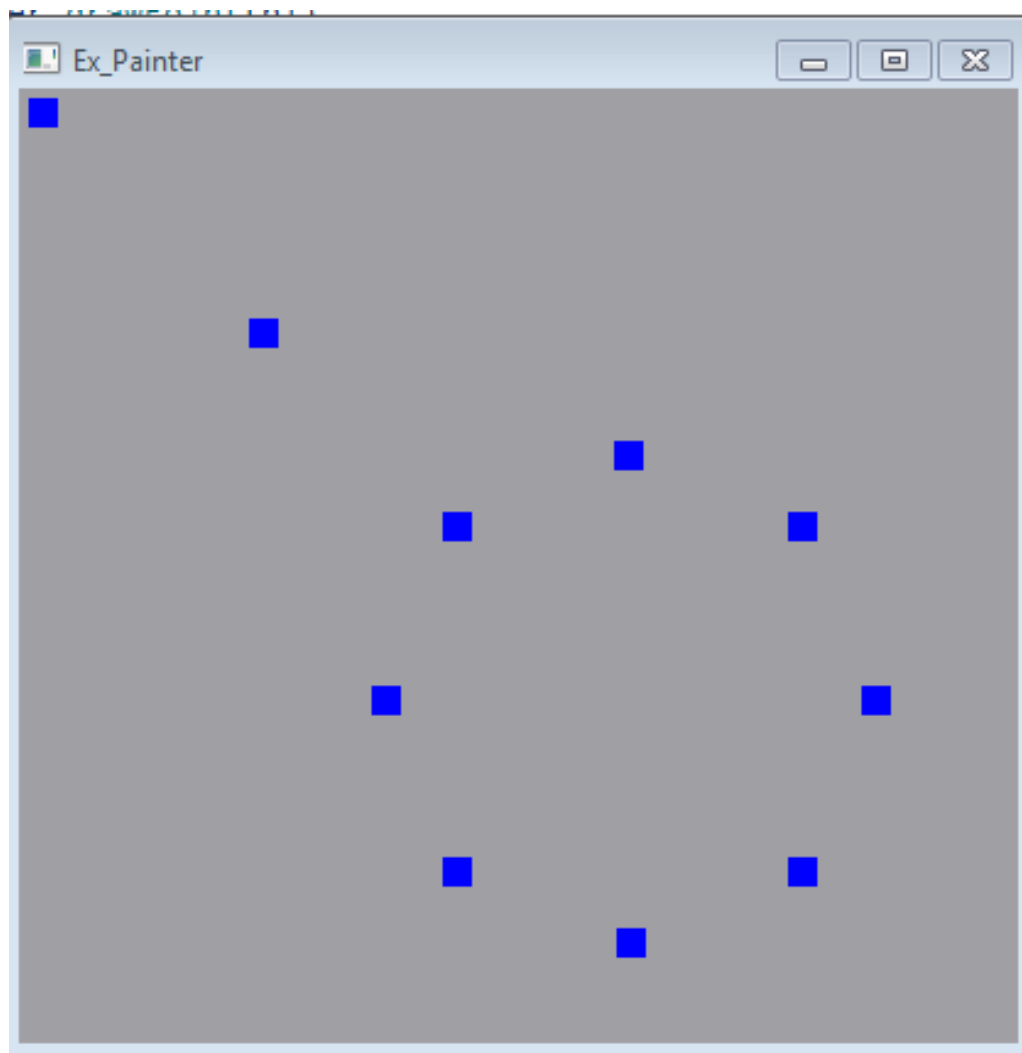
# Методи малювання класу QPainter Qt

1. Методи `void drawPoint (int x, int y),`  
`void drawPoint (const QPoint &p)` – малюють точку. `x, y` – координати точки. Буде відображатися з урахуванням поточних налаштувань пензля.

```
painter.drawPoint(100,100);  
painter.drawPoint(p1);
```

2. `void drawPoints (const QPoint *points, int pointCount)` – дозволяє намалювати множину точок, `points` – масив точок, `pointCount` – кількість точок, яку необхідно відобразити.

```
QPoint p[8];  
for (int i=0;i<8;i++)  
{ p[i]=QPoint(250 + 100 * cos(2 * 3.14 * i / 8), 250 +  
100 * sin(2 * 3.14 * i / 8));}  
painter.drawPoints(p,8); }
```



# Методи малювання класу QPainter Qt

## 3. Малювання відрізків:

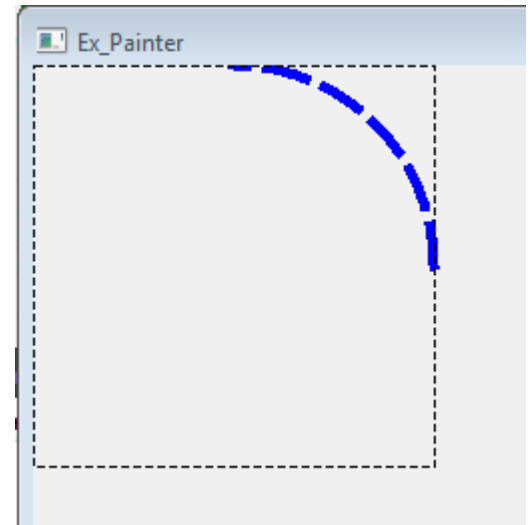
```
void drawLine (int x1, int y1, int x2, int y2)  
void drawLine (const QPoint &p1, const QPoint  
&p2)
```

```
painter.drawLine (200, 0, 0, 200);
```

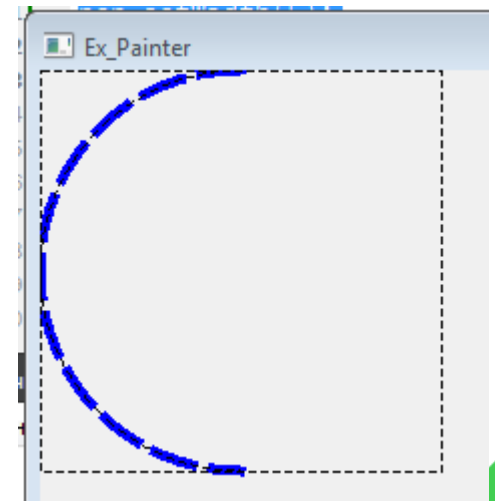
4. Метод `void drawArc (const QRect &r, int startAngle, int spanAngle)` малює дугу, обмежену прямокутником. Одиниця вимірювання - 1/16 градуса. Нульове значення `startAngle` відповідає «3 годинам». Додатне значення кутових величин відраховується проти годинникової стрілки.

# Методи малювання класу QPainter Qt

```
QRect r1 (0,0,200,200);  
pen.setWidth(1);  
pen.setColor(Qt::black);  
painter.setPen(pen);  
painter.drawRect(r1);  
pen.setWidth(5);  
pen.setColor(Qt::blue);  
painter.setPen(pen);  
painter.drawArc(r1,0,16*90);
```




```
painter.drawArc(r1,16*90,16*180);
```





# Методи малювання суцільних фігур



Контур фігури малюється поточним пером, площа фігури заливається поточним пензлем.

 1. Метод `void drawRect (const QRect &rect), void drawRect (int x,int y,int w,int h)` – малює прямокутник.

 2. `void fillRect(const QRect &r, const Qbrush &brush), void eraseRect(const QRect &r)` – малює прямокутник без контуру.

 3. `void drawEllipse (int x, int y, int w, int h), void drawEllipse (const QRect &r)` – малюють еліпс, вписаний в прямокутну область. Параметри методів задають цю область.



# Методи малювання суцільних фігур



4. `void drawChord (const QRect &r, int startAngle, int spanAngle)` – малює хорду.



```
painter.drawChord(QRect(0, 0, 100, 100), 0, 16*90);
```

5. `void drawPie (const QRect &r, int startAngle, int spanAngle)` – малює сектор.



```
painter.drawPie(QRect(0, 0, 100, 100), 0, 16*90);
```

6. `void drawPolygon (const QPoint *points, int pointCount)` – малює замкнуту фігуру довільної форми.



```
QPoint points[3]={QPoint(0, 100), QPoint(100, 0),  
QPoint(200, 100)};
```

```
painter.drawPolygon(points, 3);
```



# Малювання тексту



Намалювати текст на поверхні віджета (картинки)

можна методом `void drawText (int x, int y, const QString &text)`.



`x` – задає горизонтальну координату «лівого краю» рядку тексту, `y` – вертикальну координату базової лінії тексту.



Текст виводиться поточним шрифтом, кольором поточного пера. Встановити новий шрифт можна



методом `void setFont (const QFont &font)`, де `font` – посилання на об'єкт шрифту.





# Малювання тексту

Конструктор класа QFont:

`QFont (const QString &family, int pointSize=-1, int weight=-1, bool italic=false)`, де `family` – назва шрифту (“Arial”, “Times New Roman” і т.д.), `pointSize` – висота шрифту в пікселях (-1 – значення за замовчанням (висота==11)), `weight` – задає ширину символів [0-99] (`QFont::Normal=50`, `QFont::Bold=75` і т.д.), `italic` – задає курсивний шрифт.

```
QFont font("Times New Roman", 20, QFont::Bold,  
true);  
painter.setFont(font);  
painter.drawText(50, 50, "Hello");
```