# Урок 1. Графические методы

## Некоторые методы класса Graphics для выведения текстовых надписей, дуг, секторов, многоугольников и прочее

Используя объект *Graphics*, можно нарисовать много разных ***фигур и линий***. Кроме основных методов, рассмотренных ранее, существуют также:

* *DrawArc* – Рисует дугу, которая является частью эллипса,
* *DrawPie* – Рисует сектор, который определяется эллипсом и двумя радиальными линиями,
* *DrawPolygon* – Рисует многоугольник, определяемый массивом точек,
* *DrawRectangles* – Рисует набор прямоугольников
* *DrawLines* – Рисует набор сегментов линий, которые соединяют массив точек,
* *DrawCurve* – Строит фундаментальную кривую через точки указанного массива точек,
* *DrawClosedCurve* – Строит замкнутую фундаментальную кривую,
* *DrawBezier* – Рисует кривую Безье,
* *DrawPath* – Рисует объект *GraphicsPath*.
* *DrawString* – Создает указываемую текстовую строку в заданном месте с помощью определяемых объектов *Brush* и *Font*

***Рисунки и значки*** можно также рисовать с помощью методов:

* *DrawImage* – Рисует заданный объект Image в заданном месте,
* *DrawIcon* – Формирует изображение, представленное указанным объектом Icon, расположенным по указанным координатам,
* *DrawImageUnscaled* – Рисует заданное изображение в заданном месте, используя его исходный фактический размер,
* *DrawImageUnscaledAndClipped* – Рисует заданное изображение без масштабирования и при необходимости обрезает его, чтобы оно вмещалось в указываемом прямоугольнике,
* *CopyFromScreen* – Выполняет передачу данных о цвете, соответствующих прямоугольной области пикселей, блоками битов с экрана на поверхность рисования

Методы прорисовки ***залитых объектов***:

* *FillPie* – Заполняет внутреннюю часть сектора, определяемого эллипсом и двумя радиальными линиями,
* *FillPolygon* – Заполняет внутреннюю часть многоугольника, определяемого массивом точек
* *FillClosedCurve* – Заполняет внутреннюю часть замкнутой фундаментальной кривой, определяемой массивом точек,
* *FillPath* – Заполняет внутреннюю часть объекта *GraphicsPath*,
* *FillRegion* – Заполняет внутреннюю часть объекта *Region*

Кроме того, можно манипулировать системой координат, используемой объектом *Graphics*.

Для любого объекта, потребляющего системные ресурсы (например, для объектов *Pen* и *Graphics*), всегда нужно вызывать метод *Dispose*.

## Рисование фигур и изображений и управление ними

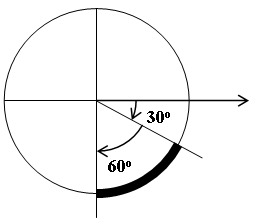
### Дуги и сектора

<http://msdn.microsoft.com/ru-ru/library/hk087c6d.aspx>

<http://msdn.microsoft.com/ru-ru/library/system.drawing.graphics.drawarc.aspx>

***Дуги*** рисуются методом *DrawArc*.

Дуга — это часть эллипса. Поэтому первые пять параметров метода имеют тот же смысл, что и первые пять параметров метода *DrawEllipse*. Добавляются еще два параметра, задающие в градусах размер и положение дуги на эллипсе (или окружности). На рисунке поясняется смысл этих параметров. Углы измеряются по часовой стрелке, начиная от оси X. Первый из двух параметров указывает начало дуги (30°), второй – длину дуги (60°).



*Синтаксис метода:*

*object*.DrawArc (Pens.*Цвет*, x, y, width, height, startAngle, sweepAngle);

Если значение параметра *sweepAngle* больше 360 или меньше -360 градусов, он интерпретируется, соответственно, как 360 или -360 градусов.

<http://msdn.microsoft.com/ru-ru/library/system.drawing.graphics.drawpie.aspx>

***Секторы*** рисуются методом *DrawPie*. Этот метод рисует сектор, определяемый дугой эллипса и двумя радиальными линиями, которые пересекаются с конечными точками дуги.

Все параметры имеют тот же смысл, что и в *DrawArc*.

*Синтаксис метода:*

*object*.DrawPie (Pens.*Цвет*, x, y, width, height, startAngle, sweepAngle);

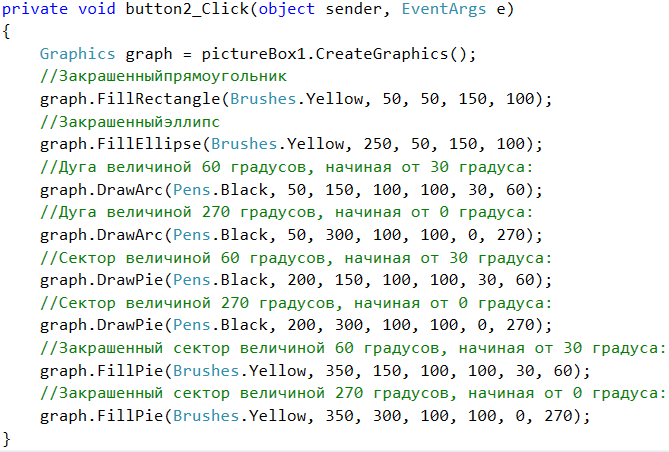
<http://msdn.microsoft.com/ru-ru/library/vstudio/hhfwf6kx%28v=vs.90%29.aspx>

***Закрашенные секторы*** рисуются методом *FillPie*. Этот метод заполняет внутреннюю часть сектора, определяемого дугой эллипса и двумя радиальными линиями, которые пересекаются с конечными точками дуги. Отличается он от *DrawPie* только наличием кисти вместо пера.

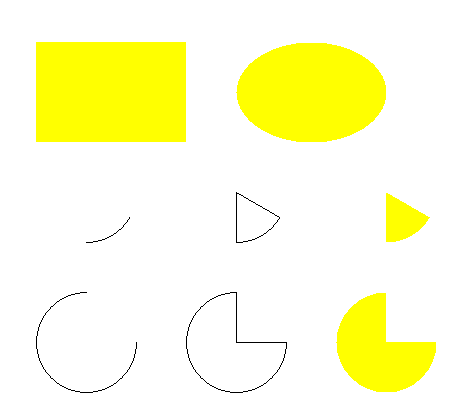
*Синтаксис метода:*

*object*.FillPie (Brushes.*Цвет*, x, y, width, height, startAngle, sweepAngle);

С помощью следующего кода



Получаем следующее изображение:



### Многоугольники

Чтобы нарисовать многоугольник, используется метод *DrawPolygon*. Рисует многоугольник, определяемый массивом точек (структур *Point*). Массив объектов *Point* содержит точки, которые должны быть соединены отрезками прямых. Каждая пара двух следующих друг за другом точек массива определяет сторону многоугольника. Кроме того, если последняя и первая точки массива не совпадают, то они определяют последнюю сторону многоугольника.

*Синтаксис метода:*

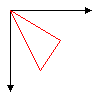
*object*.DrawPolygon (Pens.*Цвет*, Points());

В приведенном ниже примере демонстрируется рисование многоугольника с тремя сторонами. Метод *DrawPolygon* автоматически замыкает многоугольник, соединяя точки (30, 60) и (0, 0) прямой линией.

Point[] p = {new Point(0, 0), new Point(50, 30), new Point(30,60)};

g.DrawPolygon(myPen, PArr);

На приведенном ниже рисунке изображен полученный многоугольник.



Аналогичный метод для рисования незамкнутого многоугольника:

g.DrawLines(myPen, PArr);

### Текстовые надписи

<http://msdn.microsoft.com/ru-ru/library/system.drawing.graphics.drawstring.aspx>

<http://msdn.microsoft.com/ru-ru/library/a3a2bads%28v=vs.100%29.aspx>

Указываемую текстовую строку в заданном месте создает метод *DrawString*.

Класс *Graphics* содержит несколько методов *DrawString*, позволяющих указывать различные свойства текста, такие как расположение, ограничивающий прямоугольник, шрифт и формат.

Если необходимо нарисовать текст в виде одной горизонтальной строки, которая начинается в указанной точке, тогда используем следующие параметры метода *DrawString*:

*Синтаксис метода:*

*object*.DrawString (s, font, brush, x, y);

*s* – строка для рисования.

*font* – объект *Font*, определяющий формат текстовой строки.

*brush* – объект *Brush*, определяющий цвет и текстуру создаваемого текста.

*x* – координата X верхнего левого угла отображаемого текста.

*y* – координата Y верхнего левого угла отображаемого текста.

Чтобы нарисовать текст с переносом по строкам внутри прямоугольника, можно воспользоваться методом *DrawString*, который принимает параметр *Rectangle*.

*Синтаксис метода:*

*object*.DrawString (s, font, brush, layoutRectangle);

*layoutRectangle* – структура *Rectangle*, которая задает место расположения создаваемого текста.

Текст, представляемый параметром *s*, отображается внутри прямоугольника, определяемого параметром *layoutRectangle*. Если текст не помещается в прямоугольнике, он обрезается до ближайшего слова.

## Основные объекты, используемые с объектом Graphics.

### Класс *Pen*

<http://msdn.microsoft.com/ru-ru/library/system.drawing.pen.aspx>

<http://msdn.microsoft.com/ru-ru/library/8z5dw491.aspx>

Рассматривался в предыдущем уроке.

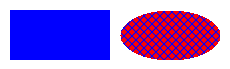
### Класс *Brush*

<http://msdn.microsoft.com/ru-ru/library/system.drawing.brush.aspx>

<http://msdn.microsoft.com/ru-ru/library/cwka53ef.aspx>

Служит для заливки областей, например, фигур, изображений или текста.

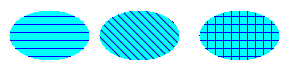
Замкнутые фигуры, такие как прямоугольники или эллипсы, состоят из двух частей — из контура и из внутренней области. Контур рисуется с помощью пера, а внутренняя область заливается с помощью кисти. В GDI+ имеется несколько классов кистей для заливки замкнутых фигур: *SolidBrush*, *HatchBrush*, *TextureBrush*, *LinearGradientBrush* и *PathGradientBrush*. Все эти классы являются производными от класса *Brush*. На приведенном ниже рисунке изображен прямоугольник, залитый сплошной кистью, и эллипс, залитый штриховой кистью.



SolidBrush br = new SolidBrush(Color.Yellow);

При заливке формы ***штриховой кистью*** необходимо задать основной цвет, цвет фона и стиль штриховки. Основной цвет определяет цвет штриховки.

В GDI+ имеется более 50 стилей штриховки; на следующем рисунке показаны стили *Horizontal*, *ForwardDiagonal* и *Cross*.



***Текстурная кисть*** позволяет заливать фигуру с использованием узора, сохраненного в виде растрового изображения.

В GDI+ имеется два типа ***градиентных кистей***: кисти линейного градиента и кисти градиента контура. Кисти линейного градиента можно использовать для заливки фигуры цветом, постепенно меняющимся от одного края фигуры до другого по горизонтали, по вертикали или по диагонали.

### Класс *Font*

Содержит описание фигур, которые должны использоваться при отрисовке текста. Определяет конкретный формат текста, включая начертание шрифта, его размер и атрибуты стиля.

<http://msdn.microsoft.com/ru-ru/library/system.drawing.font.aspx>

Font myFont = new Font("Arial", 20, FontStyle.Italic);

### Структура *Color*

Содержит различные цвета. Представляет цвета в терминах каналов альфа, красного, зеленого и синего (ARGB).

<http://msdn.microsoft.com/ru-ru/library/system.drawing.color.aspx>

Именованные цвета представляются с помощью свойств структуры *Color*. Цвета по имени библиотеки MSDN: <http://msdn.microsoft.com/en-us/library/aa358802.aspx>.

Цвет каждого пикселя представляется 32-битным числом: по 8 бит для интенсивности альфа-канала, красного, зеленого и синего (*ARGB*). Каждый из четырех указанных компонентов представляет собой число от 0 до 255, где 0 соответствует нулевой интенсивности, а 255 — наибольшей интенсивности. Альфа-компонент определяет прозрачность цвета: 0 соответствует полной прозрачности, а 255 — полной непрозрачности. Чтобы определить альфа-, красный, зеленый или синий компоненты цвета, можно использовать свойство *A*, *R*, *G* или *B*, соответственно.

Пользовательский цвет можно создать с помощью одного из методов *FromArgb*. Если этот метод используется с тремя параметрами, то это красный, зеленый или синий компоненты цвета. Например:

|  |  |
| --- | --- |
| FromArgb (255, 0, 0) | Красный цвет |
| FromArgb (0, 255, 0) | Зеленый цвет |
| FromArgb (0, 0, 255) | Синий цвет |
| FromArgb (255, 255, 0) | Желтый цвет |
| FromArgb (0, 0, 0) | Черный цвет |
| FromArgb (255, 255, 255) | Белый цвет |
| FromArgb (90, 90, 90) | Темно-серый цвет |
| FromArgb (220, 220, 220) | Светло-серый цвет |

Чем меньше каждой краски мы положим, тем темнее будет цвет, чем больше – тем светлее.

Функция *FromArgb* позволяет управлять прозрачностью цвета. Для этого используется ее вариант не с тремя, а с четырьмя параметрами. Второй, третий и четвертый параметры имеют привычный нам смысл количества красной, зеленой и синей краски. А вот первый параметр определяет прозрачность цвета. Если он равен 255, то цвет полностью непрозрачен, а если 0 – то полностью прозрачен (невидим).

Также возможно использовать функцию *FromArgb* с двумя параметрами: первый определяет прозрачность цвета, а второй – сам цвет по имени.

Color c1, c2, c3;

//Непрозрачный случайный цвет

Random rnd = new Random();

c1 = Color.FromArgb(rnd.Next(0, 256), rnd.Next(0, 256), rnd.Next(0, 256));

//Полупрозрачный синий цвет

c2 = Color.FromArgb(127, 0, 0, 255);

//Полупрозрачный зеленый цвет

c3 = Color.FromArgb(127, Color.Green);

Для определения цвета рисования можно также использовать системные цвета. Это те цвета, на которые настроены элементы Windows: окна, их заголовки, цвет текста в заголовках, меню и т.д.

Несистемные цвета C# предоставлял в структуре *Color* и в классах *Pens* и *Brushes* пространства имен *System.Drawing*. Системные же цвета находятся соответственно в классах *SystemColors*, *SystemPens* и *SystemBrushes* пространства имен *System.Drawing*.

Если вы не используете в своем проекте системных цветов, цвета вашего приложения на любом компьютере будут одинаковы независимо от настроек Windows.

Опасайтесь делать цвета одних элементов проекта системными, а других – несистемными.