МДК 01.01 Разработка программных модулей

Пушкарев Александр Николаевич
к.т.н., преподаватель
ГАПОУ ТО «Колледж цифровых и педагогических технологий»

Графические построения

Графика

- Появление операционной системы **Microsoft Windows** избавило программистов от необходимости учитывать аппаратные особенности видеоадаптеров – новая операционная система переложила эту задачу на драйверы видеоадаптеров. Драйверы создаются разработчиками видеоадаптеров и обеспечивают наилучшую реализацию возможностей аппаратуры.
- Для эффективной работы с графикой программных приложений в **Windows** был предусмотрен целый набор системных функций, реализующих интерфейс графических устройств. Указанный набор сокращенно называется **GDI** (**Graphics Device Interface**).
- Впоследствии с появлением платформы **.NET** в её библиотеке **Microsoft .NET Framework** приложениям стала доступна усовершенствованная версия интерфейса **GDI+**.

Графика

- С точки зрения приложений, интерфейс GDI состоит из контекста отображения и инструментов, предназначенных для рисования.
- Контекст отображения можно сравнить с листом бумаги, на котором приложение рисует изображение или пишет текст.
- Инструментами для рисования выступают перья, кисти, а также шрифты и даже целые графические изображения, с помощью которых создается итоговое изображение.
- Приложение может создать контекст отображения не только для окна приложения, но и для любого физического графического устройства вывода (например, для принтера). В этом случае оно может рисовать на принтере изображения при помощи тех же функций, что и для рисования в окне приложения. Такой контекст называется контекстом устройства.
- Контекст устройства выступает в роли связующего звена между приложением и драйвером устройства, определяя, как нужно выполнять операции вывода на данном устройстве (цвет и толщину линии, тип системы координат и т. д.).

Класс Graphics

- Концепция графического интерфейса **GDI+** несколько отличается от концепции «классического» графического интерфейса **GDI**, с которым привыкли иметь дело разработчики приложений **Microsoft Windows**.
- Прежде всего это касается класса **Graphics**, реализующего в себе как свойства контекста отображения, так и инструменты, предназначенные для рисования в этом контексте.
- Для того чтобы приложение могло что-нибудь нарисовать в окне, оно должно, прежде всего, получить или создать для этого окна объект класса **Graphics**. Далее, пользуясь свойствами и методами этого объекта, приложение может рисовать в окне различные фигуры или текстовые строки.
- Каждое окно в операционной системе **Windows** имеет свой идентификатор (handle). Зная идентификатор окна, можно легко получить связанный с этим окном контекст отображения. Приложения, разрабатываемый с использованием **Microsoft .NET Framework**, могут получить идентификатор формы или любого другого элемента управления через их свойство **Handle**.
- Рассмотрим пример приложения, позволяющего пользователю рисовать на форме, используя мышь.

Класс Graphics

```
public partial class Form1 : Form
{

    // Переменная doDraw указывает, следует ли рисовать мышью
    bool doDraw = false;

    // Нажатие кнопки мыши включает режим рисования
    private void Form1_MouseDown(object sender,
                                 MouseEventArgs e)
    {
     doDraw = true;
    }

    // Отпускание кнопки мыши выключает режим рисования
    private void Form1_MouseUp(object sender,
                               MouseEventArgs e)
    {
     doDraw = false;
    }

    // Продолжение – на следующем слайде
```

Класс Graphics

```
public partial class Form1 : Form
{

    // При движении мышью рисуем, если включен режим
    private void Form1_MouseMove(object sender,
                                 MouseEventArgs e)
    {
      if (doDraw)
      {
            Graphics g = Graphics.FromHwnd(this.Handle);
            SolidBrush redBrush = new SolidBrush(Color.Red);
            g.FillRectangle(redBrush, e.X, e.Y, 1, 1);
      }
    }

}
```

Класс Graphics

- В приведённом примере в классе формы создаётся булева переменная **doDraw**, которая должна сигнализировать о том, следует ли рисовать линию на форме, когда пользователь перемещает курсор мыши по её поверхности.
- Для установки определённого значения этой переменной используются события нажатия и отпускания кнопки мыши.
- В событии движения мыши по форме создаётся переменная **g** класса **Graphics**, в которую записывается ссылка на экземпляр данного класса, реализующего контекст отображения формы. Для получения ссылки используется метод **FromHwnd** класса **Graphics**, в качестве параметра принимающий идентификатор элемента управления, который записан в его свойстве **Handle**. В нашем примере мы получаем идентификатор формы при помощи конструкции **this.Handle**.
- Для рисования на форме используется один из инструментов, доступных в **GDI+**, – кисть, описываемая классом **SolidBrush**. При помощи конструктора данного класса в переменную **redBrush** передаётся ссылка на новый экземпляр кисти красного цвета. С её помощью через метод **FillRectangle** класса **Graphics**, закрашивается квадрат с единичной стороной (пиксель), расположенный в том месте, где пользователь провёл мышью.

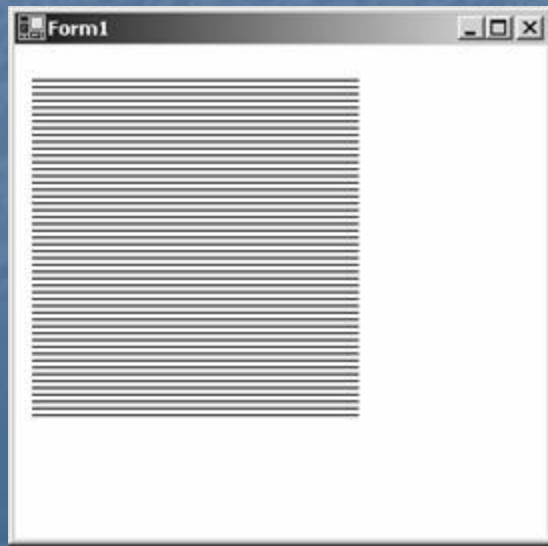
Линия

- Метод **DrawLine** рисует линию, соединяющую две точки с заданными координатами. У этого метода существует несколько перегруженных версий:

```
public void DrawLine(Pen, Point, Point);
public void DrawLine(Pen, PointF, PointF);
public void DrawLine(Pen, int, int, int, int);
public void DrawLine(Pen, float, float, float, float);
```

- Пример отрисовки 50 линий:

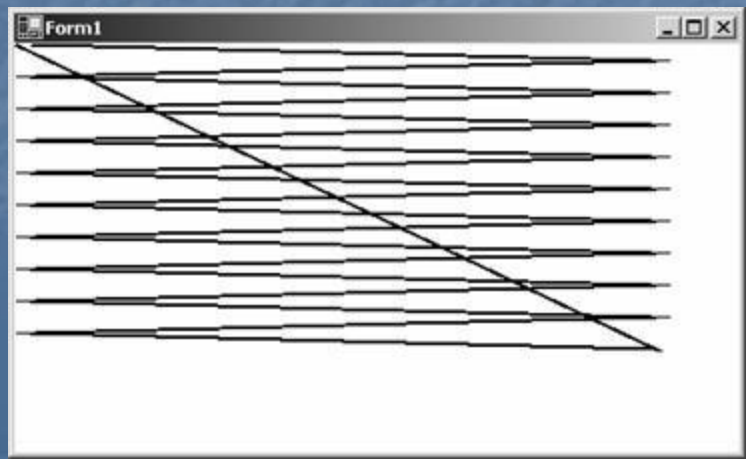
```
for (int i = 0; i < 50; i++)
{
    g.DrawLine(new Pen(Brushes.Black, 1),
        10, 4 * i + 20, 200, 4 * i + 20);
}
```



Набор линий

- Метод **DrawLines** позволяет отрисовать по точкам набор отрезков, последовательно соединяя их между собой. Координаты этих точек передаются методу через массив. У метода **DrawLines** существует несколько перегруженных версий:

```
public void DrawLines(Pen, Point[ ]);
public void DrawLines(Pen, PointF[ ]);
```



Прямоугольник

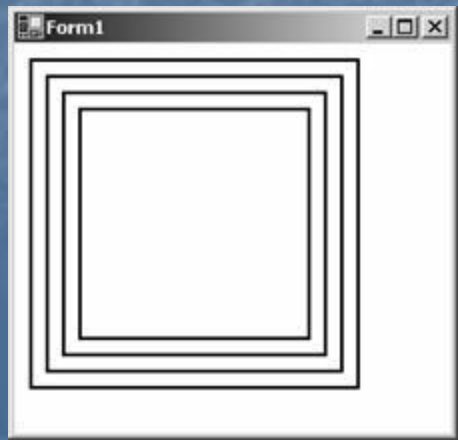
- Метод **DrawRectangle** обеспечивает отрисовку прямоугольников одним из трёх доступных способов:

```
public void DrawRectangle(Pen, Rectangle);
public void DrawRectangle(Pen, int, int, int, int);
public void DrawRectangle(Pen, float, float, float, float);
```

Набор прямоугольников

- Метод **DrawRectangles** позволяет нарисовать прямоугольники, составляющие единый набор. Существует два перегруженных варианта этого метода:

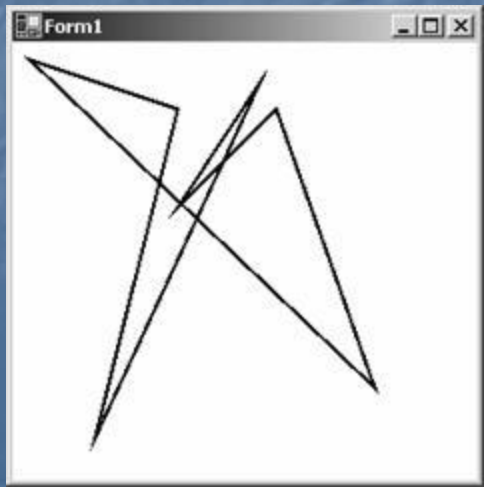
```
public void DrawRectangles(Pen, Rectangle[ ]);
public void DrawRectangles(Pen, RectangleF[ ]);
```



Многоугольник

- Метод **DrawPolygon** рисует многоугольник по набору точек, переданных в виде массива. Многоугольники можно рисовать двумя способами:

```
public void DrawPolygon(Pen, Point[ ]);
public void DrawPolygon(Pen, PointF[ ]);
```



Эллипс

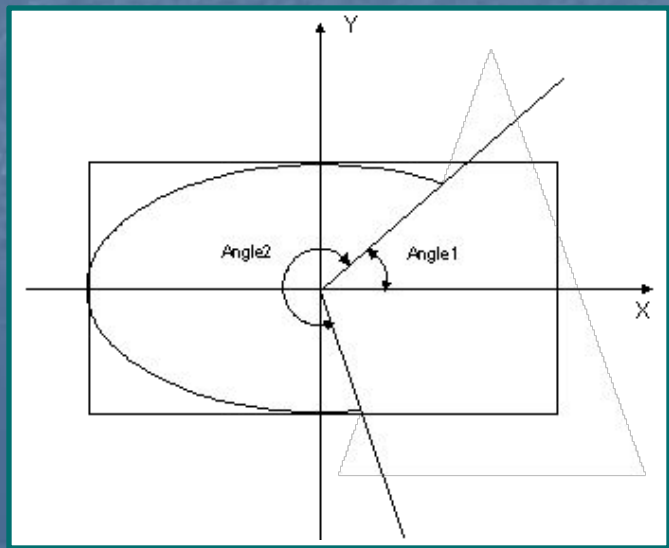
- Метод **DrawEllipse** рисует эллипс, вписанный в заданную прямоугольную область. Расположение и размеры области передаются в метод одним из четырёх способов:

```
public void DrawEllipse(Pen, Rectangle);
public void DrawEllipse(Pen, RectangleF);
public void DrawEllipse(Pen, int, int, int, int);
public void DrawEllipse(Pen, float, float, float, float);
```

Сегмент эллипса (дуга)

- Метод **DrawArc** позволяет построить сегмент эллипса, или дугу. Сегмент задаётся координатами прямоугольной области, в которую вписан эллипс, и величинами двух углов, которые определяют начало и конец линии дуги. Углы отсчитываются в направлении против движения часовой стрелки. Для метода **DrawArc** предусмотрено четыре перегруженных варианта:

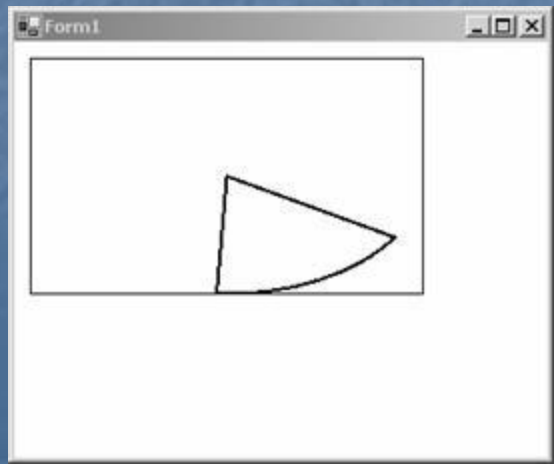
```
public void DrawArc(Pen, Rectangle, float, float);
public void DrawArc(Pen, RectangleF, float, float);
public void DrawArc(Pen, int, int, int, int, int, int);
public void DrawArc(Pen, float, float, float, float, float, float);
```



Замкнутый сегмент эллипса

- Метод **DrawPie** позволяет нарисовать замкнутый сегмент эллипса, который по форме напоминает кусок пирога (pie). Перегруженные варианты вызова данного метода аналогичны вариантам вызова метода **DrawArc**:

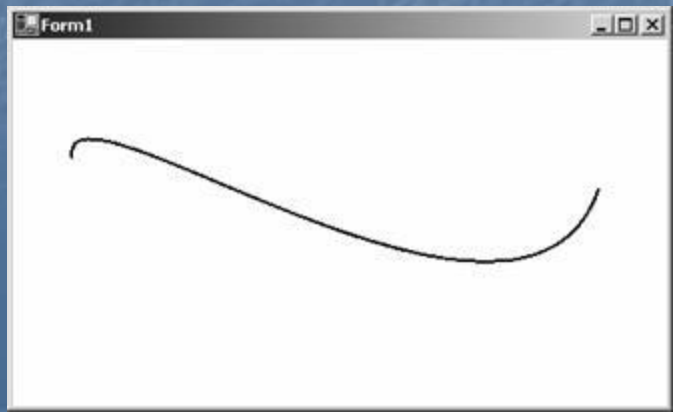
```
public void DrawPie(Pen, Rectangle, float, float);
public void DrawPie(Pen, RectangleF, float, float);
public void DrawPie(Pen, int, int, int, int, int, int);
public void DrawPie(Pen, float, float, float, float, float, float);
```



Кривые Безье

- Кривая Безье представляет собой кривую, которая задается четырьмя точками. Две из них определяют начало и конец дуги, а две другие управляют её изгибами – кривая стремится достигнуть этих двух точек. Для отрисовки кривых Безье используется метод, имеющий три перегруженных варианта:

```
public void DrawBezier(Pen, Point, Point, Point, Point);
public void DrawBezier(Pen, PointF, PointF, PointF, PointF);
public void DrawBezier(Pen, float, float, float, float, float, float , float, float);
```



Набор кривых Безье

- Метод **DrawBeziers** позволяет задавать координаты точек в виде массива. Данный метод представлен двумя вариантами:

```
public void DrawBeziers(Pen, Point[ ]);
public void DrawBeziers(Pen, PointF[ ]);
```

Сплайны

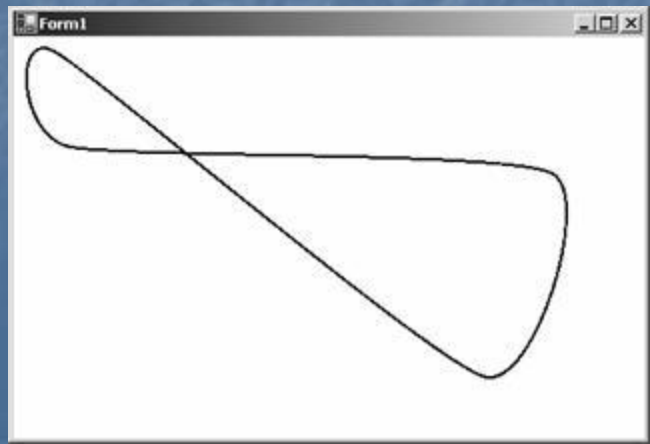
- Сплайн представляет собой линию с плавными изгибами, проходящую через заданные ключевые точки. В отличие от кривой Безье, сплайн проходит через все свои ключевые точки, а не только через первую и последнюю.
- Для рисования сплайнов используется метод **DrawCurve** в одном из двух своих перегруженных вариантов:

```
public void DrawCurve(Pen, Point[ ]);
public void DrawCurve(Pen, PointF[ ]);
```

Замкнутые сплайны

- Метод **DrawCurve** позволял отрисовать сплайн, который начинался в первой своей ключевой точке и заканчивался в последней. Сплайн указанного вида называется открытым.
- В свою очередь, метод **DrawClosedCurve** рисует так называемый закрытый (замкнутый) сплайн, у которого начало и конец соединены. Существует два варианта отрисовки замкнутых сплайнов с использованием данного метода:

```
public void DrawClosedCurve(Pen, Point[ ]);
public void DrawClosedCurve(Pen, PointF[ ]);
```



Литература

- Фролов А.В., Фролов Г.В. Визуальное проектирование приложений C#. – М.: КУДИЦ-Образ, 2003. – 512 с.

- Интернет-версия:

  http://www.frolov-lib.ru/books/msnet/c_sharp2/ch10.html