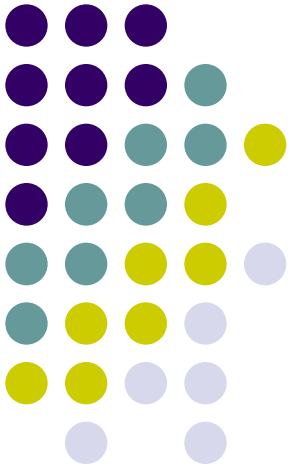
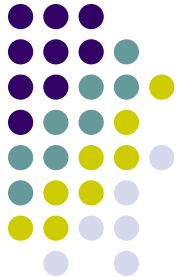
Графика в C#



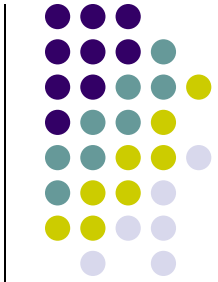
Класс Graphics



Класс Graphics реализует 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.

Создание объекта Graphics



- Получите ссылку на объект Graphics через объект PaintEventArgsПолучите ссылку на объект Graphics через объект PaintEventArgs при обработке события PaintПолучите ссылку на объект Graphics через объект PaintEventArgs при обработке события Paint формы или элемента управления. Это обычный способ получения ссылки на графический объект при создании кода рисования элементов управления. Подобным образом можно получить графический объект как свойство объекта PrintPageEventArgsПолучите ссылку на объект Graphics через объект PaintEventArgs при обработке события Paint формы или элемента управления. Это обычный способ получения ссылки на графический объект при создании кода рисования элементов управления. Подобным образом можно получить графический объект как свойство объекта PrintPageEventArgs при обработке события PrintPageПолучите ссылку на объект Graphics через объект PaintEventArgs при обработке события Paint формы или элемента управления. Это обычный способ получения ссылки на графический объект при создании кода рисования элементов управления. Подобным образом можно получить графический объект как

PaintEventArgs в обработчике события Paint

При создании обработчика события PaintEventHandlerПри создании обработчика события PaintEventHandler для элементов управления графический объект предоставляется как одно из свойств объекта PaintEventArgs.

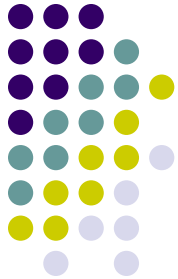
Получение ссылки на объект Graphics из объекта PaintEventArgs в событии Paint:

1. Объявите объект Graphics.
2. Присвойте переменной ссылку на объект GraphicsПрисвойте переменной ссылку на объект Graphics, передаваемый как часть PaintEventArgs.
3. Вставьте код для рисования формы или элемента

Пример

```
private void Form1_Paint(object sender,
                                        PaintEventArgs e)
 {
      Graphics g = e.Graphics;
      // Код рисования
  }
```

Метод CreateGraphics

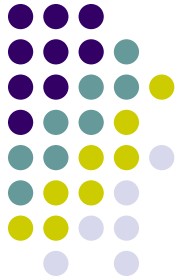


Для получения ссылки на объект GraphicsДля получения ссылки на объект Graphics, который соответствует поверхности рисования формы или элемента управления, можно также использовать метод CreateGraphics этой формы или элемента управления.

Создание объекта Graphics с помощью метода CreateGraphics

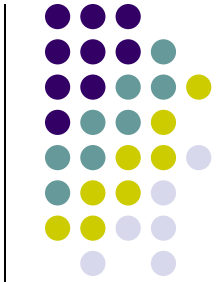
Вызовите метод CreateGraphics формы или элемента управления, на котором необходимо отобразить графику.

Пример



```
Graphics g;

 g = this.CreateGraphics();

//поверхность рисования - форма
```

Создание из объекта Image

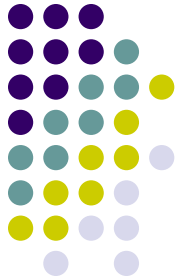


Объект Graphics можно создать из любого объекта, производного от класса Image.

Создание объекта Graphics из объекта Image

Вызовите метод Graphics.FromImageВызовите метод Graphics.FromImage переменной Image, из которой нужно создать объект Graphics.

Пример



```
Bitmap myBitmap = new Bitmap(@"C:\Documents
                    and Settings\Joe\Pics\myPic.bmp");

Graphics g = Graphics.FromImage(myBitmap);
```

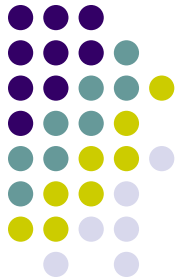
Идентификатор окна Handle и объект Graphics

Приложения Microsoft .NET Framework могут получить идентификатор формы или любого другого элемента управления при помощи свойства Handle. В примере приложение получает идентификатор окна формы Form1 с помощью свойства this.Handle.

Зная идентификатор окна, с помощью метода Graphics.FromHwnd нетрудно получить нужный нам объект класса Graphics:

```
Graphics g = Graphics.FromHwnd(this.Handle);
```

Рисование фигур и изображений и управление ими



После создания объекта GraphicsПосле создания объекта Graphics его можно использовать для рисования линий и фигур, отображения текста или изображения и управления ими. Ниже представлены основные объекты, используемые с объектом Graphics.

- Класс Pen — служит для рисования линий, контуров и отрисовки других геометрических объектов.
- Класс Brush — служит для заливки областей, например фигур, изображений или текста.
- Класс Font — содержит описание фигур, которые должны использоваться при отрисовке текста.
- Структура Color — содержит различные цвета.

Создание объекта Pen (Перо)

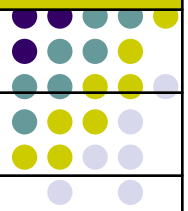
Перья используются для рисования линий и простейших геометрических фигур и создаются как объекты класса Pen. Вот соответствующие конструкторы:

- public Pen(Color); - создает перо заданного цвета. Цвет задается при помощи объекта класса Color.
- public Pen(Color, float); - позволяет дополнительно задать толщину пера.
- public Pen(Brush); - создают перо на основе кисти.
- public Pen(Brush, float); - дополнительно можно указать толщину создаваемого пера.

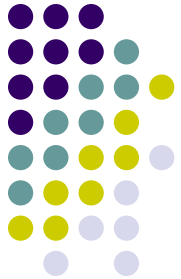
```
Pen myPen = new Pen(Color.Blue, 2);
```

После того как перо создано, программа может определить его атрибуты при помощи свойств класса Pen.

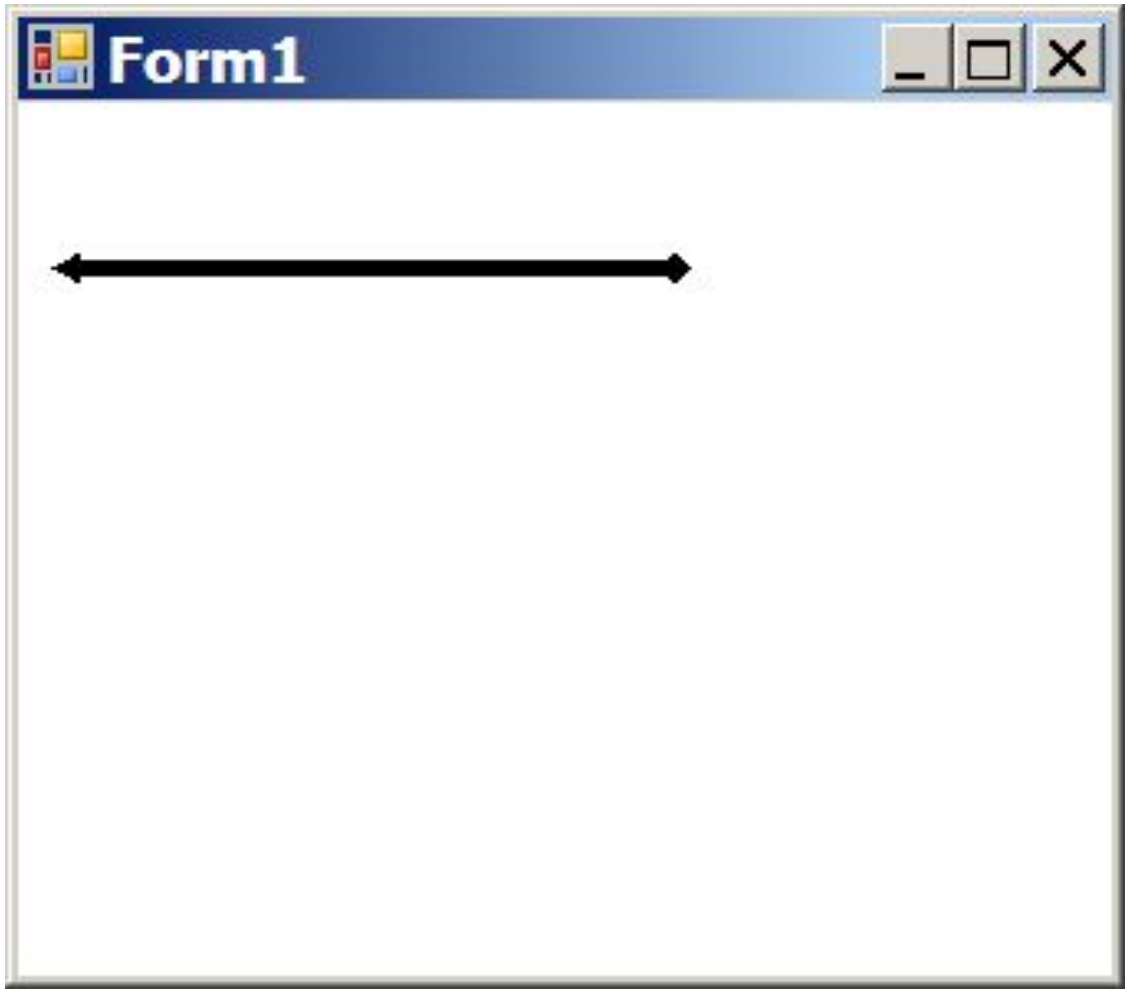
Свойство	Описание
Alignment	Выравнивание пера
Width	Ширина линии
Brush	Кисть, используемая пером
Color	Цвет пера
DashStyle	Стиль пунктирных и штрих-пунктирных линий
DashCup	Вид точек и штрихов пунктирных и штрих-пунктирных линий
DashOffset	Расстояние от начала линии до начала штриха
DashPattern	Массив шаблонов для создания произвольных штрихов и пробелов штриховых и штрих-пунктирных линий
StartCup EndCup	Стиль концов линий
LineCap	Формы концов линий
LineJoin	Стиль соединения концов двух различных линий
MiterLimit	Предельная толщина в области соединения остроконечных линий



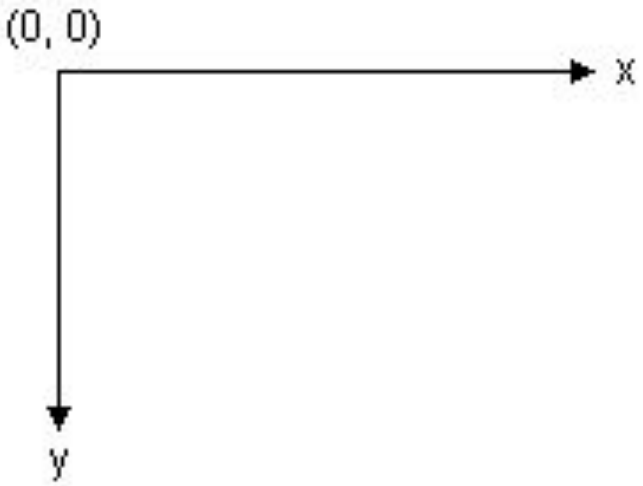
Пример



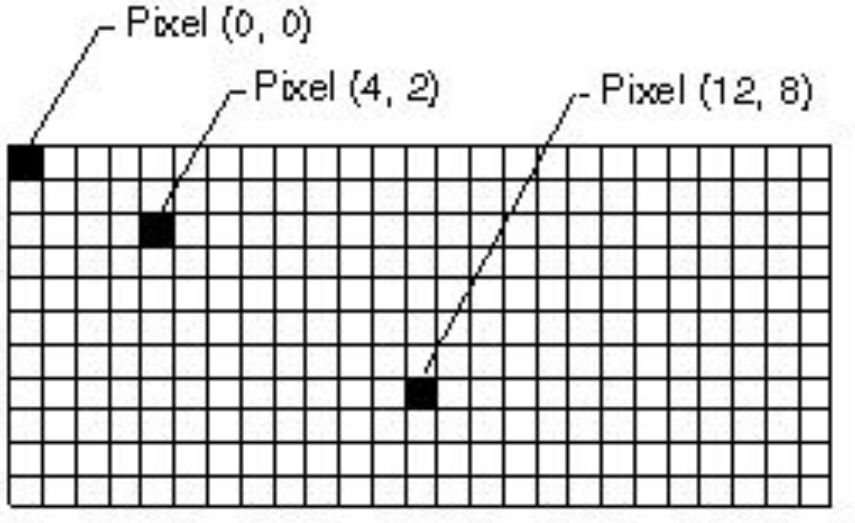
```
…
using System.Drawing.Drawing2D;
…
private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.Clear(Color.White);

            int x = 10;
            int y = 50;
            Pen myPen = new Pen(Color.Black, 1);
            myPen.Width = 5;
            myPen.DashStyle = DashStyle.Solid;
            myPen.StartCap = LineCap.ArrowAnchor;
            myPen.EndCap = LineCap.DiamondAnchor;
            g.DrawLine(myPen, x, y, 200, y);

        }
```


Form1

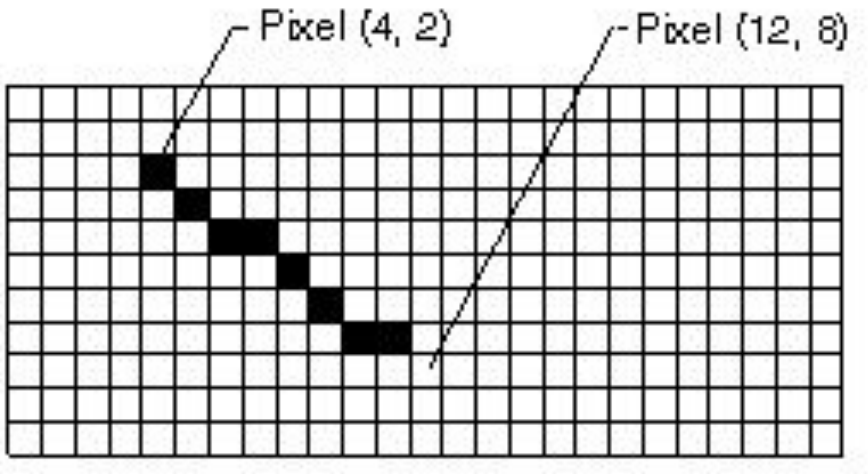
Рисование линий, прямоугольников и других фигур с использованием интерфейса GDI+ происходит в некоторой системе координат. Пользователь может выбрать одну из многих реализованных систем координат, но по умолчанию используется плоская декартова система координат, начало координат которой расположено в верхнем левом углу экрана, ось X направлена вправо, а ось Y — вниз. Единицей измерения в заданной по умолчанию системе координат является пиксель (минимальный элемент изображения).



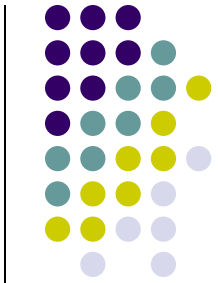
Изображение на мониторе компьютера формируется как прямоугольный массив точек (пикселей), являющихся минимальными элементами изображения. Количество пикселей, отображаемых на экране, зависит от типа монитора. Количество пикселей, отображаемых на экране конкретного монитора, может быть изменено пользователем в определенных пределах.



При использовании интерфейса GDI+ для рисования линии, прямоугольника или формы необходимо указывать определенные ключевые данные, задающие параметры рисуемого элемента. Например, отрезок задается координатами двух точек, а прямоугольник — точкой, шириной и высотой. Интерфейс GDI+ взаимодействует с программным обеспечением драйвера экрана, чтобы определить, какие пиксели экрана должны быть высвечены, чтобы на экране возникло изображение линии, прямоугольника или кривой. На приведенном ниже рисунке показаны пиксели, высвечиваемые, чтобы отобразить отрезок прямой линии от точки с координатами (4, 2) до точки с координатам



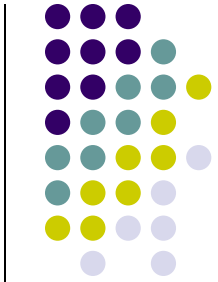
За время развития компьютерной графики были выделены геометрические фигуры, наиболее полезные при создании двухмерных изображений.



- Lines
- Прямоугольники
- Эллипсы
- Дуги
- Многоугольники
- Фундаментальные сплайны
- Сплайны Безье

Линия

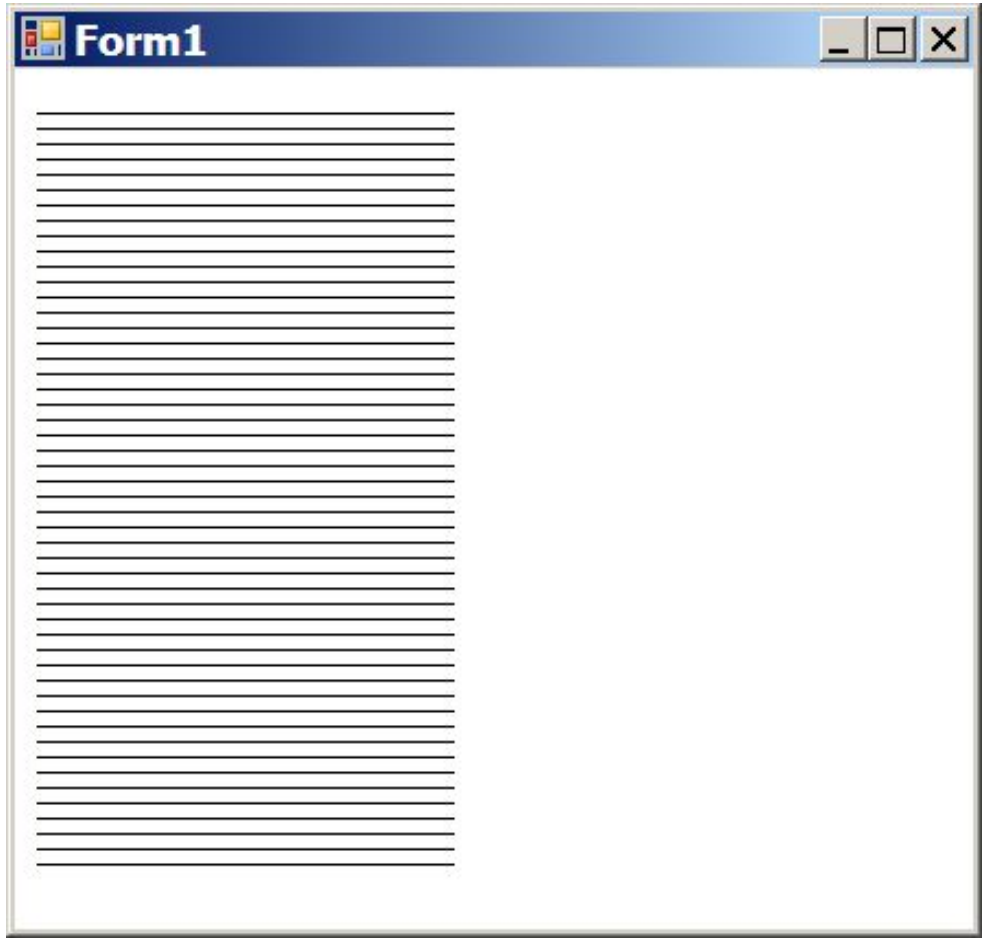
Метод DrawLine рисует линию, соединяющую две точки с заданными координатами.



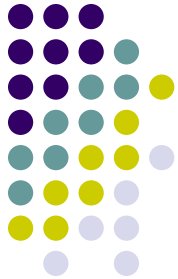
DrawLine(Pen, Point, Point)	Проводит линию, соединяющую две структуры Point.
DrawLine(Pen, PointF, PointF)	Проводит линию, соединяющую две структуры PointF.
DrawLine(Pen, Int32, Int32, Int32, Int32)	Проводит линию, соединяющую две точки, задаваемые парами координат.
DrawLine(Pen, Single, Single, Single, Single)	Проводит линию, соединяющую две точки, задаваемые парами координат.

Пример

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
        Graphics g = e.Graphics;
        g.Clear(Color.White);
        for (int i = 0; i < 50; i++)
        {
            g.DrawLine(new Pen(Brushes.Black, 1), 10, 7 * i
   + 20, 200,
                7 * i + 20);
        }
}
```


Form1

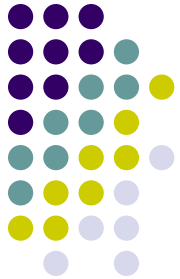
Прямоугольник



Метод DrawRectangle позволяет рисовать прямоугольники, заданные координатой верхнего левого угла, а также шириной и высотой.

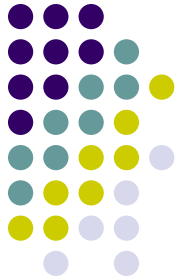
DrawRectangle(Pen, Rectangle)	Рисует прямоугольник, определяемый структурой Rectangle.
DrawRectangle(Pen, Int32, Int32, Int32, Int32)	Рисует прямоугольник, который определен парой координат, шириной и высотой.
DrawRectangle(Pen, Single, Single, Single, Single)	Рисует прямоугольник, который определен парой координат, шириной и высотой.

Наборы



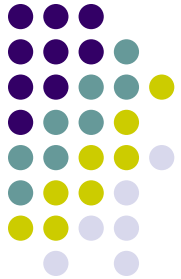
Помимо перечисленных ранее методов для рисования линий, прямоугольников и сплайнов Безье существуют вспомогательные методы, выполняющие рисование нескольких подобных элементов (линий, прямоугольников или сплайнов) за один вызов: DrawLinesПомимо перечисленных ранее методов для рисования линий, прямоугольников и сплайнов Безье существуют вспомогательные методы,

Набор прямоугольников

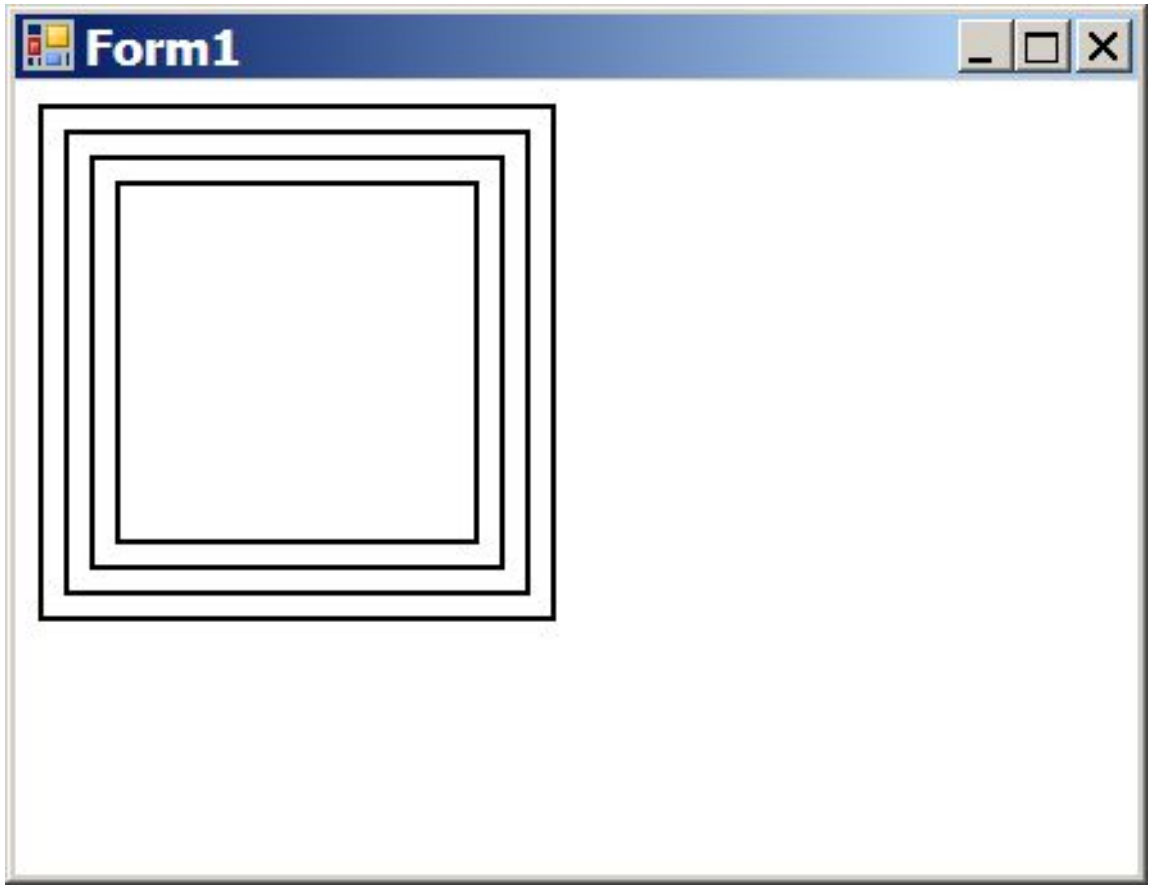


В классе Form1 определено перо myPen и массив вложенных друг в друга прямоугольников myRectsArray:

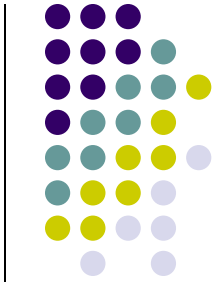
```
Pen myPen = new Pen(Color.Black, 2);
  Rectangle[] myRectsArray =
  {
    new Rectangle(10, 10, 200, 200),
    new Rectangle(20, 20, 180, 180),
    new Rectangle(30, 30, 160, 160),
    new Rectangle(40, 40, 140, 140)
  };
```



```
private void Form1_Paint(object sender,
   PaintEventArgs e)
{
        Graphics g = e.Graphics;
        g.Clear(Color.White);
        g.DrawRectangles(myPen, myRectsArray);
}
```


Form1

Многоугольник



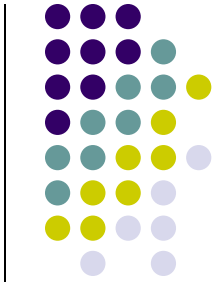
Метод DrawPolygon поможет нарисовать многоугольник, заданный своими вершинами.

Предусмотрено два варианта этого метода:

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

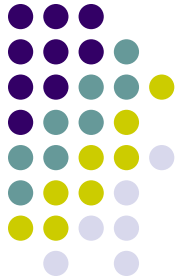
В первом случае методу DrawPolygon через второй параметр передается массив точек класса Point, в котором координаты точек заданы целыми числами, а во втором — массив класса PointF, где координаты соединяемых точек задаются в виде числе с плавающей десятичной точкой.

Пример

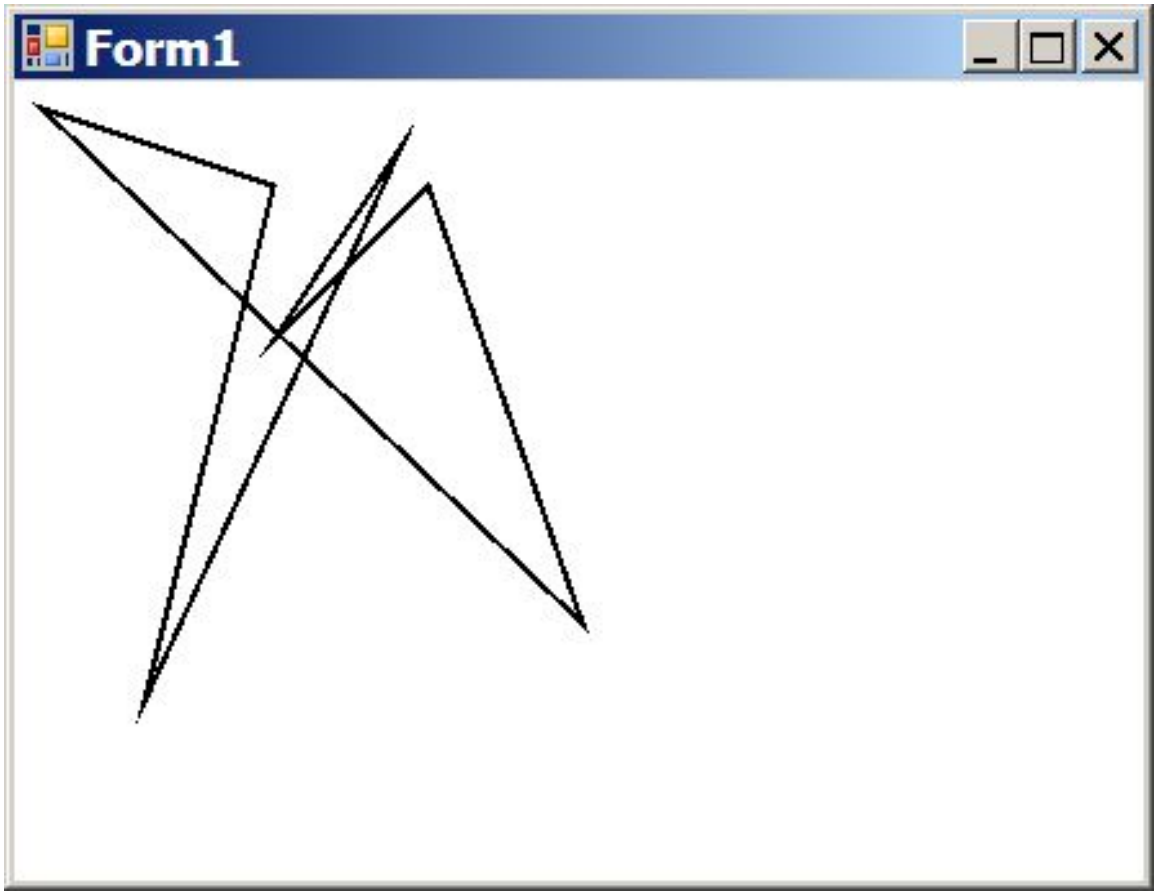


Определим перо myPen и массив точек myPoints:

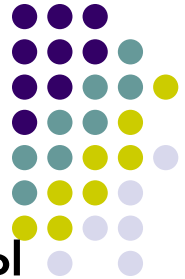
```
Pen myPen = new Pen(Color.Black, 2);

  Point[] myPoints =
  {
    new Point(10, 10),
    new Point(100, 40),
    new Point(50, 240),
    new Point(150, 24),
    new Point(100, 100),
    new Point(160, 40),
    new Point(220, 210)
  };
```



```
private void Form1_Paint(object sender,
    PaintEventArgs e)
    {
            Graphics g = e.Graphics;
            g.Clear(Color.White);
            g.DrawPolygon(myPen, myPoints);
    }
```


Form1

Эллипс



Метод DrawEllipse рисует эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров.

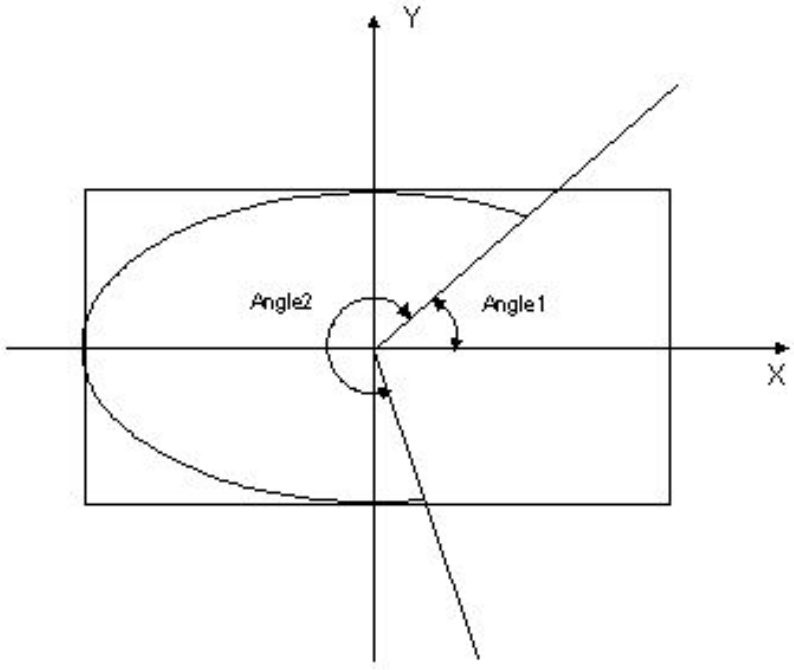
Предусмотрено четыре варианта метода DrawEllipse:

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);

Эти методы отличаются только способом, при помощи которого описывается расположение и размеры прямоугольной области, в которую вписан эллипс. Вы можете задавать расположение и размеры этой области в виде рассмотренных ранее объектов класса Rectangle, RectangleF, а также в виде целых чисел или чисел с плавающей десятичной точкой.

Сегмент эллипса

При помощи метода DrawArc программа может нарисовать сегмент эллипса. Сегмент задается при помощи координат прямоугольной области, в которую вписан эллипс, а также двух углов, отсчитываемых в направлении против часовой стрелки. Первый угол Angle1 задает расположение одного конца сегмента, а второй Angle2 — расположение другого конца сегмента (рисунок).



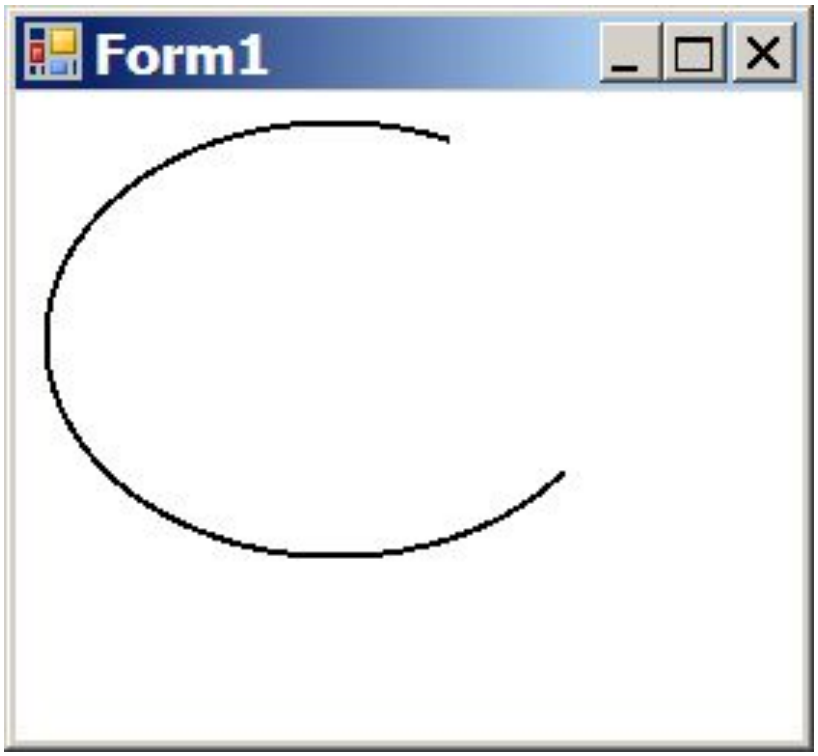
Предусмотрено четыре варианта метода DrawArc:

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);

Первый параметр метода DrawArc определяет перо, с помощью которой будет нарисован сегмент. Последние два параметра задают углы Angle1 и Angle2 в соответствии с рисунком. Расположение и размеры прямоугольной области передаются методу DrawArc аналогично тому, как это делается для рассмотренного выше метода DrawEllipse.

Пример:

```
g.DrawArc(myPen, 10, 10, 200, 150, 30, 270);
```


Form1

Замкнутый сегмент эллипса

Для рисования замкнутого сегмента эллипса (pie) можно воспользоваться методом DrawPie.

Имеется 4 варианта этого метода:

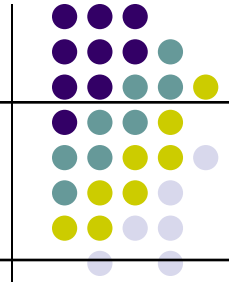
public void DrawPie(Pen, Rectangle, float, float);
public void DrawPie(Pen, RectangleF, float, float);
public void DrawPie(Pen, int, int, int, int, int, int);
public void DrawPie(Pen, float, float, float, float, float, float);

В качестве первого параметра методу нужно передать перо для рисования. Последние два параметра определяют углы, ограничивающие сегмент эллипса. Эти углы используются таким же образом, как и при рисовании незамкнутого сегмента эллипса методом DrawArc, рассмотренным выше. Остальные параметра задают расположение и размеры прямоугольника, в который вписывается сегмент эллипса.

Закрашенные фигуры

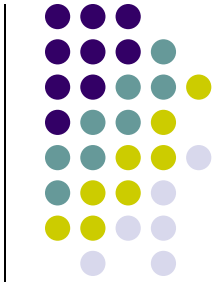
В классе Graphics определен ряд методов, предназначенных для рисования закрашенных фигур. Имена некоторых из этих методов, имеющих префикс Fill, перечислены в таблице далее.

Есть два отличия методов с префиксом Fill от одноименных методов с префиксом Draw. Прежде всего, методы с префиксом Fill рисуют закрашенные фигуры, а методы с префиксом Draw — не закрашенные. Кроме этого, в качестве первого параметра методам с префиксом Fill передается не перо класса Pen, а кисть класса Brush.



Метод	Описание
FillRectangle	Рисование закрашенного прямоугольника
FillRectangles	Рисование множества закрашенных многоугольников
FillPolygon	Рисование закрашенного многоугольника
FillEllipse	Рисование закрашенного эллипса
FillPie	Рисование закрашенного сегмента эллипса
FillClosedCurve	Рисование закрашенного сплайна
FillRegion	Рисование закрашенной области типа Region

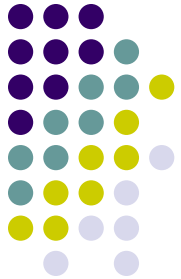
Кисти



Внутренняя область окна и замкнутых геометрических фигур может быть закрашена при помощи кисти. В приложениях Microsoft .NET Frameworks кисти создаются на базе классов, производных от абстрактного класса Brush. Это следующие классы:

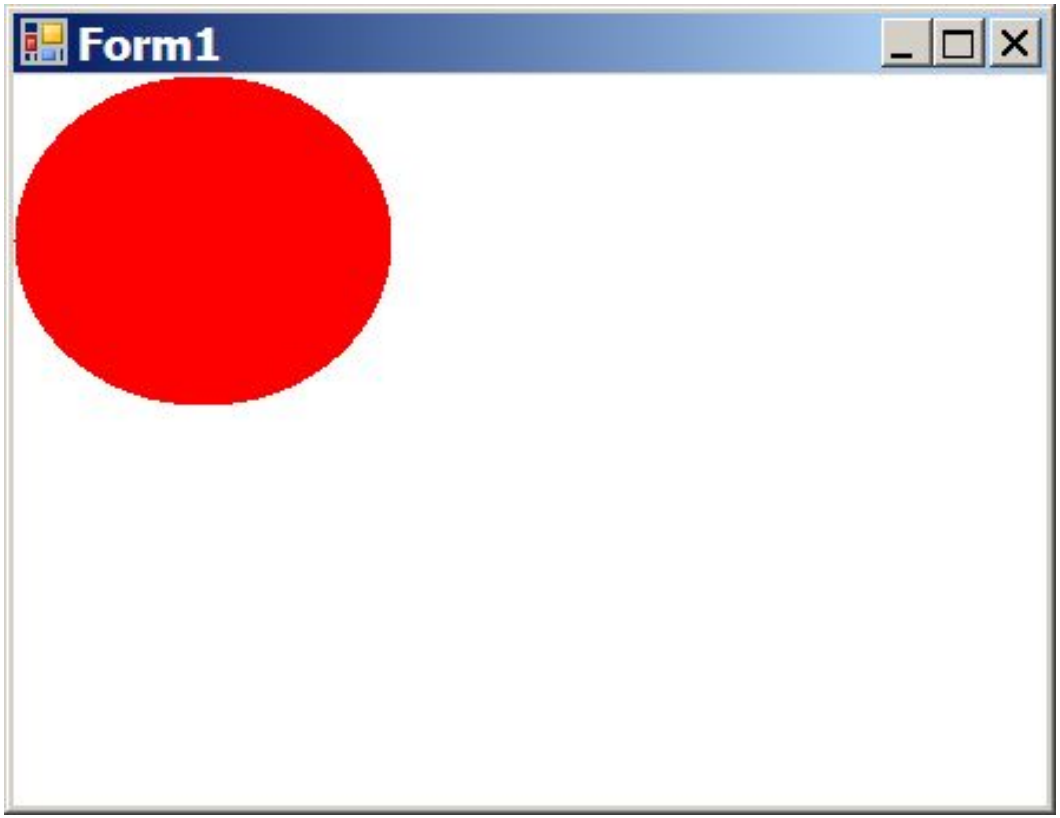
- Brushes
- SolidBrush;
- HatchBrush;
- TextureBrush;
- LinearGradientBrush;
- PathGradientBrush

Кисть для сплошной закраски

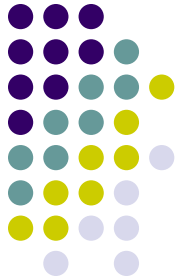


Простейшие из кистей — это кисти Brushes и SolidBrush, предназначенные для сплошной закраски фигур. Эти кисти создается при помощи конструктора с одним параметром, задающим цвет в виде объекта класса Color.

```
SolidBrush mySolidBrush = new SolidBrush(Color.Red);
g.FillEllipse(mySolidBrush, 0, 0, 160, 140);
```


Form1

Кисти типа HatchBrush



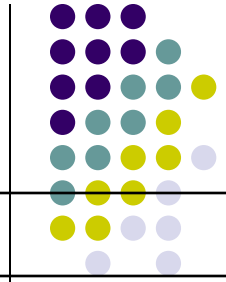
При помощи класса HatchBrush можно создать прямоугольную кисть заданного стиля, с заданным цветом изображения и фона.

Для создания кистей этого типа предусмотрено два конструктора:

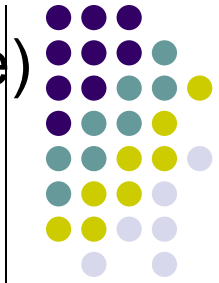
public HatchBrush(HatchStyle, Color);
public HatchBrush(HatchStyle, Color, Color);

Первый из этих конструкторов позволяет создать кисть заданного стиля и цвета, а второй дополнительно позволяет указать цвет фона.

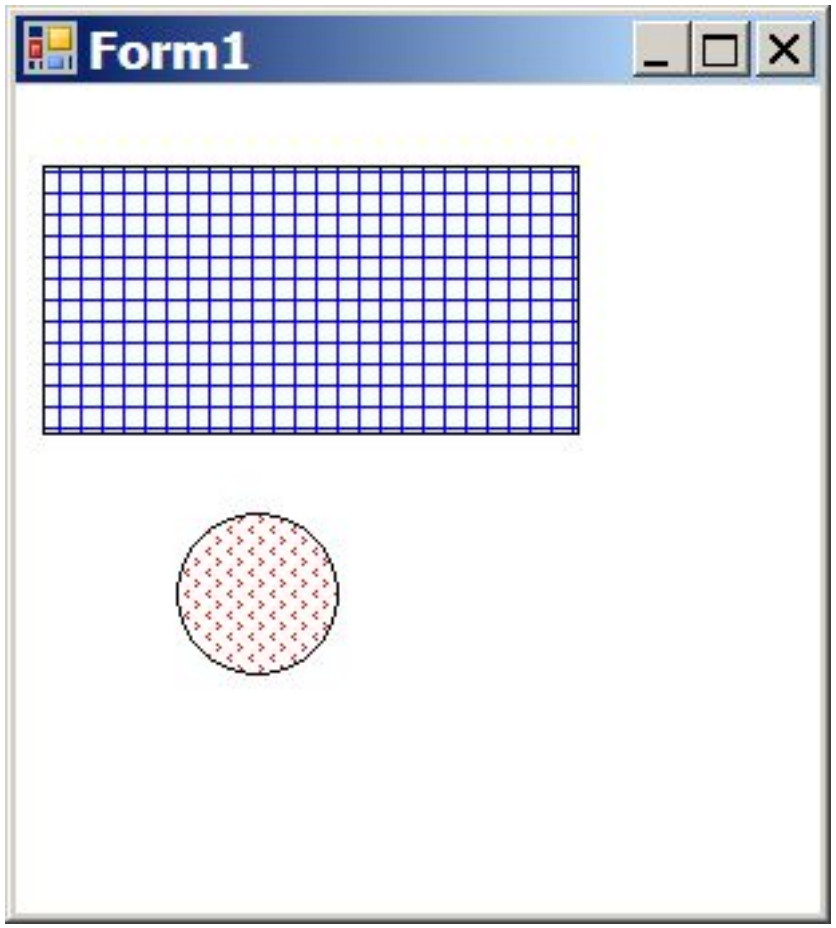
Стили кисти типа HatchBrush



Константа	Описание
BackwardDiagonal	Линии штриховки располагаются в обратном направлении (от верхнего правого угла к нижнему левому углу кисти)
Cross	Пересекающиеся горизонтальные и вертикальные линии
DarkHorizontal	Горизонтальные линии, которые на 50% плотнее, чем при использовании константы Horizontal (темная штриховка)
DarkUpwardDiagonal	Диагональные линии, плотнее на 50% чем при использовании константы BackwardDiagonal (темная штриховка)
DarkVertical	Вертикальные линии, которые на 50% плотнее, чем при использовании константы Vertical (темная штриховка)
DashedHorizontal	Штриховые горизонтальные линии
. . .	. . .



```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.Clear(Color.White);
    HatchBrush hb = new HatchBrush(HatchStyle.Cross,
  Color.Blue,
  Color.White);
    g.FillRectangle(hb, 10, 30, 200, 100);
    g.DrawRectangle(new Pen(Brushes.Black, 1), 10, 30, 200,
  100);
    HatchBrush hb2 = new HatchBrush(HatchStyle.Divot,
  Color.Red,
  Color.White);
    g.FillEllipse(hb2, 60, 160, 60, 60);
    g.DrawEllipse(new Pen(Brushes.Black, 1), 60, 160, 60, 60);
}
```


Form1

Кисти типа TextureBrush

Можно создать собственную кисть на базе класса TextureBrush, в виде произвольного изображения. Такая кисть, называемая текстурной, может иметь любой внешний вид и любой цвет.

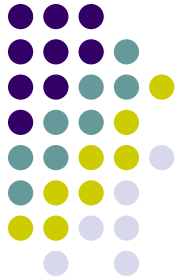
Для создания кисти класса TextureBrush Ваше приложение может воспользоваться одним из следующих конструкторов:

public TextureBrush(Image);
public TextureBrush(Image, Rectangle);
public TextureBrush(Image, RectangleF);
public TextureBrush(Image, WrapMode);
public TextureBrush(Image, Rectangle, ImageAttributes);
public TextureBrush(Image, WrapMode, Rectangle);
public TextureBrush(Image, WrapMode, RectangleF);

Самому простому из этих конструкторов нужно передать изображение, загруженное из ресурсов приложения или из внешнего файла (с помощью рассмотренного ранее метода Image.FromFile).

Структуры Rectangle и RectangleF позволяют задать границы прямоугольной области, ограничивающие изображение кисти.

Градиентные кисти



Приложениям GDI+ доступен еще один вид кистей — так называемая градиентная кисть. Линейная градиентная кисть LinearGradientBrush позволяет задать в кисти переход от одного цвета к другому. Кисти с множественным градиентом PathGradientBrush позволяют задать внутри кисти область, которая будет закрашена с использованием цветового градиента.