

GDI+ (Graphic Device Interface)

- GDI: là gì? các thành phần, các namespace liên quan
- Lớp Graphic
- Cách vẽ các hình: chữ nhật, vuông, tròn, eclipse
- Cách vẽ đường thẳng, đường cong, path, polygon, string
- Brush, Pen, Clipping, Image, Buffer, Sprite, Transformation

Bài tập: Xây dựng ứng dụng cho phép vẽ (tròn, eclipse, vuông) hoặc đường (đường thẳng, đường cong, path) với pen tương ứng. Ứng dụng cho phép chọn màu, chọn đường cho pen. Ứng dụng cho phép chọn brush và fill hình (nếu vẽ hình) với brush tương ứng.

Tổng quan

- GDI là một giao diện lập trình ứng dụng (API) của Window đặc trưng cho việc vẽ các đối tượng và tương tác với các thiết bị đầu ra như màn hình và máy in.
- GDI+ là một phiên bản phát triển của GDI giúp giảm độ phức tạp của GDI và làm tăng tính linh hoạt trong việc vẽ các đối tượng.
- Các lớp GDI+ cung cấp bởi .NET Framework được bao gói lại và được định nghĩa trong System.Drawing.dll

Tổng quan

- GDI+ cung cấp nhiều đặc tính mới so với GDI cũ
 - Hỗ trợ các tọa độ số thực (PointF, SizeF, RectangleF)
 - Phối màu với giá trị alpha (Alpha Blending)
 - Cung cấp tính trong suốt cho hình ảnh (image transparency)
 - Làm mịn lề (antialiasing)
 - Cung cấp những phép biến đổi
 - Các loại brush texture và gradient

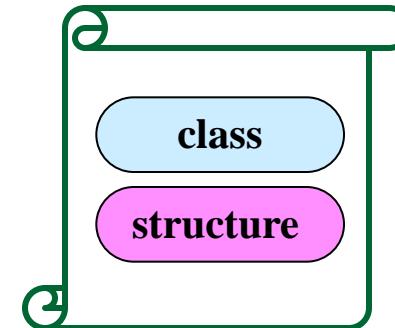
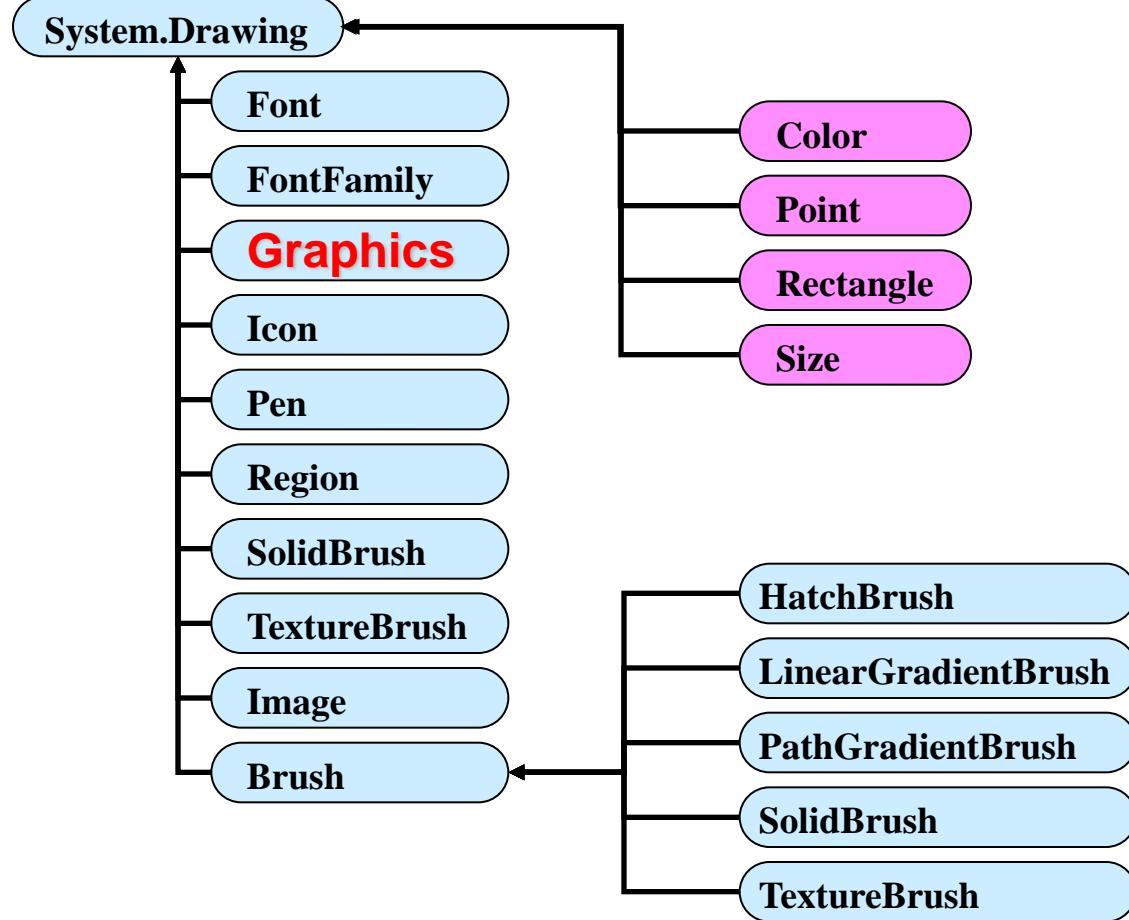
Tổng quan

- GDI+ bao gồm 3 nhóm “dịch vụ” chính:
 - 2D vector graphics: cho phép tạo hình từ các hình cơ bản (primitive): đường thẳng, tròn, ellipse, đường cong,...
 - Imaging: làm việc với các tập tin hình ảnh (bitmap, metafile)
 - Typography: vẽ chữ

GDI+ namespace

- **System.Drawing**
- **System.Drawing.Drawing2D**
- **System.Drawing.Imaging**
- **System.Drawing.Printing**
- **System.Drawing.Text**

System.Drawing



Lớp Graphics

- Đây là lớp quan trọng của GDI+
- Mọi thao tác vẽ đều thực hiện trên đối tượng Graphic của lớp này
- Bất kì lớp control nào cũng đều có thuộc tính Graphic dùng để vẽ chính nó
- Không thể tạo đối tượng Graphics từ toán tử new.

```
// Chương trình sẽ báo lỗi nếu khai báo như sau  
Graphics g = new Graphics ();
```

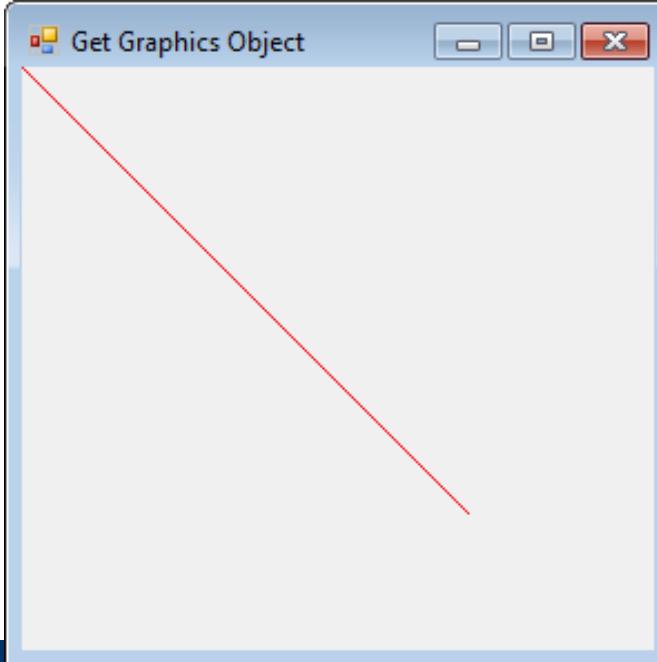
Lấy đối tượng Graphics

- Có thể lấy đối tượng Graphics từ tham số PaintEventArgs của sự kiện Paint của form hoặc từ phương thức OnPaint của form.

```
// Sự kiện Paint
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Pen pen = new Pen(Color.Red);
    g.DrawLine(pen, 0, 0, 200, 200);
}
```

Lấy đối tượng Graphics

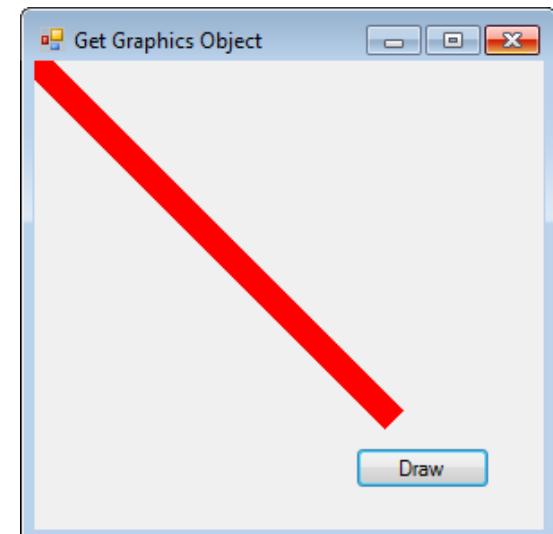
```
// Override OnPaint  
protected override void OnPaint(PaintEventArgs e)  
{  
    Graphics g = e.Graphics;  
    Pen pen = new Pen(Color.Red);  
    g.DrawLine(pen,0,0,100,100);  
}
```



Lấy đối tượng Graphics

- Lấy đối tượng Graphics thông qua hàm `CreateGraphics()`, hàm này trả về một đối tượng Graphics. Ảnh vẽ sẽ mất đi khi Form được Reload.

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = this.CreateGraphics();
    Pen pen = new Pen(Color.Red, 15);
    g.DrawLine(pen, 0, 0, 200, 200);
    g.Dispose();
}
```

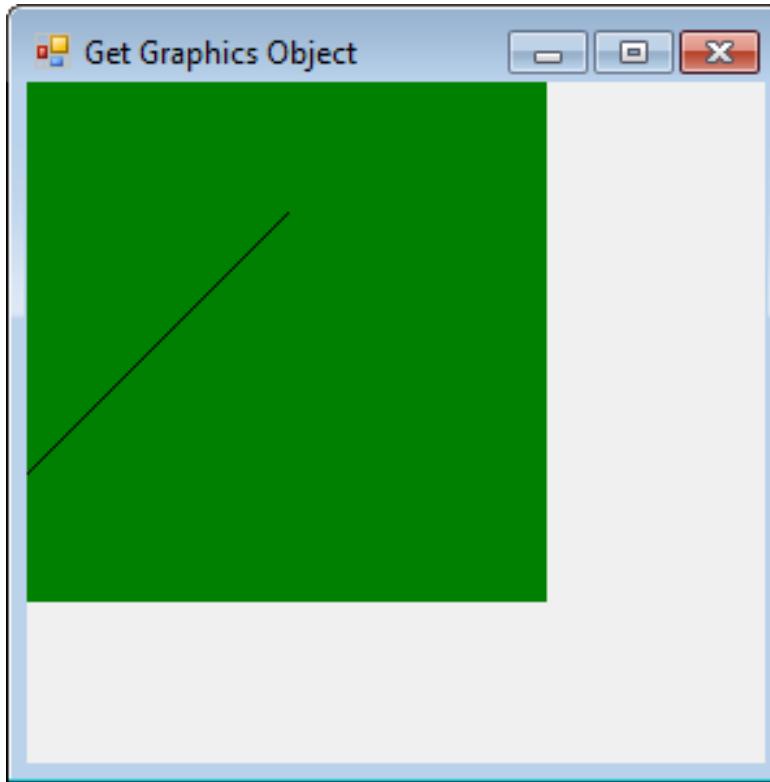


Lấy đối tượng Graphics

- Lấy đối tượng Graphics từ các hàm tĩnh của lớp Graphics như `FromImage`.

```
protected override void OnPaint(PaintEventArgs e)
{
    Bitmap bmp = new Bitmap(200, 200);
    Graphics g = Graphics.FromImage(bmp);
    g.FillRectangle(Brushes.Green, 0, 0, 200, 200);
    g.DrawLine(Pens.Black, new Point(0, 150), new
        Point(100, 50));
    e.Graphics.DrawImageUnscaled(bmp, 0, 0);
}
```

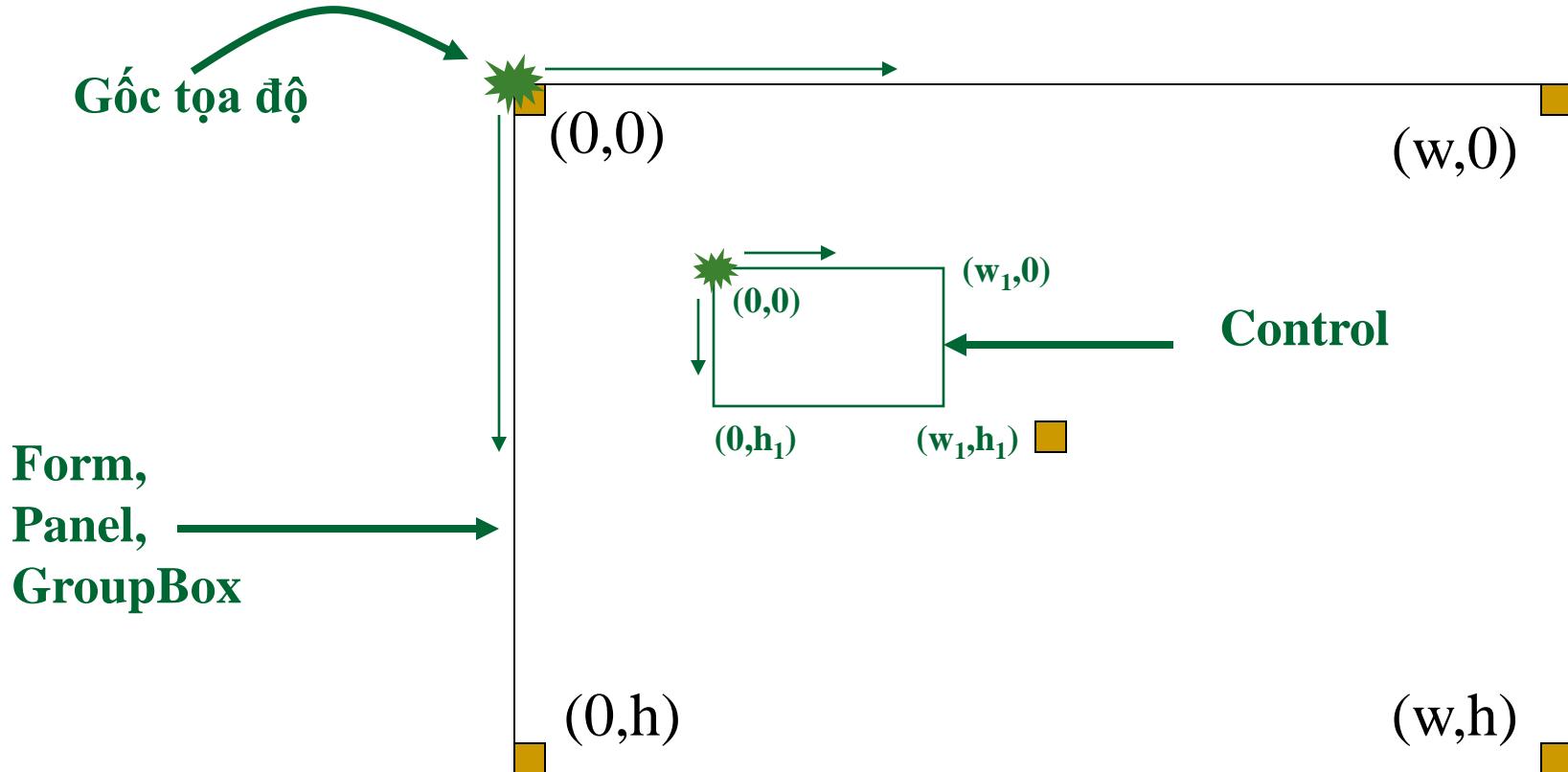
Lấy đối tượng Graphics



Tiến trình vẽ với các lớp của GDI+

- Xác định phạm vi (bề mặt) vẽ
 - Hệ thống toạ độ
 - Cấu trúc dữ liệu: Color, Rectangle, Point, Size
- Tạo công cụ vẽ
 - Cọ tô (Brush)
 - Bút vẽ (Pen)
 - Kiểu chữ (Font)
- Thực hiện thao tác vẽ, tô màu
 - Lớp Graphics

Tọa độ hệ thống



Tọa độ hệ thống

- **Graphics.GraphicsUnit**: xác định đơn vị của bề mặt
 - **GraphicsUnit.Pixel** (default)
 - **GraphicsUnit.Inch**
 - **GraphicsUnit.Milimeter**
 - **GraphicsUnit.Point**
- **Graphics.PageScale**: tỷ lệ output
 - **g.PageScale = 1f** (default)

Color

- Cấu trúc Color đại diện cho màu ARGB (alpha-red-green-blue) trong GDI+.
- Lớp này chứa một số thuộc tính tĩnh dùng để hiển thị cho một số màu nhất định như: Color.Black, Color.Red

Một số thuộc tính và phương thức của lớp Color

Thuộc tính	Miêu tả
A	Trả về giá trị alpha của màu thể hiện mức độ trong suốt (255 opaque)
R	Trả về giá trị của sắc màu đỏ
G	Trả về giá trị của sắc màu xanh lá cây
B	Trả về giá trị của sắc màu xanh dương
IsEmpty	Xác định xem màu có được tạo
IsKnownColor	Xác định xem màu có được xác định trước

Color

Phương Thức	Mô tả
FromArgb	Tạo màu sắc từ các giá trị 8bit alpha, red, green, blue
GetBrightness	Trả về giá trị độ sáng của cấu trúc Color
GetHue	Trả về giá trị Hue của cấu trúc Color
GetSaturation	Trả về giá trị Saturation
ToArgb	Trả về giá trị 32bit của cấu trúc Color

Tạo màu sử dụng hàm FromArgb()

Color red = Color.FromArgb(255,0,0);

Color blue = Color.FromArgb(128, 0, 255, 0);

Color

Color	
Transparent	DarkOrange
AliceBlue	DarkOrchid
AntiqueWhite	DarkRed
Aqua	DarkSalmon
Aquamarine	DarkSeaGreen
Azure	DarkSlateBlue
Beige	DarkSlateGray
Bisque	DarkTurquoise
Black	DarkViolet
BlanchedAlmond	DeepPink
Blue	DeepSkyBlue
BlueViolet	DimGray
Brown	DodgerBlue
BurlyWood	Firebrick
CadetBlue	FloralWhite
Chartreuse	ForestGreen
Chocolate	Fuchsia
Coral	Gainsboro
CornflowerBlue	GhostWhite
Cornsilk	Gold
Crimson	Goldenrod
Cyan	Gray
DarkBlue	Green
DarkCyan	GreenYellow
DarkGoldenrod	Honeydew
DarkGray	HotPink
DarkGreen	IndianRed
DarkKhaki	Indigo
DarkMagenta	Ivory
DarkOliveGreen	Khaki
Lavender	MediumVioletRed
LavenderBlush	MidnightBlue
LawnGreen	MintCream
LemonChiffon	MistyRose
LightBlue	Moccasin
LightCoral	NavajoWhite
LightCyan	Navy
LightGoldenrodYellow	OldLace
LightGreen	Olive
LightGray	OliveDrab
LightPink	Orange
LightSalmon	OrangeRed
LightSeaGreen	Orchid
LightSkyBlue	PaleGoldenrod
LightSlateGray	PaleGreen
LightSteelBlue	PaleTurquoise
LightYellow	PaleVioletRed
Lime	PapayaWhip
LimeGreen	PeachPuff
Linen	Peru
Magenta	Pink
Maroon	Plum
MediumAquamarine	PowderBlue
MediumBlue	Purple
MediumOrchid	Red
MediumPurple	RosyBrown
MediumSeaGreen	RoyalBlue
MediumSlateBlue	SaddleBrown
MediumSpringGreen	Salmon
MediumTurquoise	SandyBrown

Một số đối tượng cơ bản của GDI+

Point, PointF	X,Y +, -, ==, !=, IsEmpty
Rectangle, RectangleF	X,Y Top, Left, Bottom, Right Height, Width Inflate(), Intersect(), Union() Contain()
Size, SizeF	+,-, ==, != Height, Width
Region	“phần ruột” của khuôn hình học Rectangle rect=new Rectangle(0,0,100,100) Region rgn= new Region(rect)

Một số enumeration

- **ContentAlignment**
- **FontStyle**
- **GraphicsUnit**
- **KnownColor**
- **RotateFlipType**
- **StringAlignment**
-

Font



- Cách tạo đối tượng Font: new Font(...)
 - Có 13 phiên bản của constructor

```
□ Font fa = new Font("Times New Roman", 8);  
Font fb = new Font("Arial", 36, FontStyle.Bold);  
Font fc = new Font(fb, FontStyle.Bold | FontStyle.Italic);  
Font fd = new Font("Arial", 1, GraphicsUnit.Inch);
```

Size = 8 pixel

Size = 1 inch

- Nếu tên font không tìm thấy thì font mặc định được sử dụng.

Font

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Font fa = new Font("Time New Roman", 14);
    g.DrawString(" DH CNTT ", fa, Brushes.Salmon, 10, 10);
    Font fb = new Font("Arial", 36, FontStyle.Bold);
    g.DrawString(" DH CNTT ", fb, Brushes.Navy, 10, 30);
    Font fc = new Font(fb, FontStyle.Bold | FontStyle.Italic);
    g.DrawString(" DH CNTT ", fc, Brushes.Orchid, 10, 80);
    Font fd = new Font("Impact", 1, GraphicsUnit.Inch);
    g.DrawString(" DH CNTT", fd, Brushes.HotPink, 10, 120);
}
```

Font



Pen



- Xác định width, style, fill style
- Không cho kế thừa, nhưng tạo thể hiện được
- Có thể tạo các đối tượng Pen thông qua các constructor với tham số là một đối tượng Brush hoặc Color và có thể xác định độ dài của Pen

SolidBrush blueBrush = new SolidBrush(Color.Blue);

HatchBrush hatchBrush =

```
new HatchBrush(HatchStyle.DashedVertical,  
Color.Black, Color.Green);
```

Pen pn1 = new Pen(blueBrush, 3);

Pen pn2 = new Pen(hatchBrush, 8);

Pen pn3 = new Pen(Color.Red);

Pen



- Có thể lấy các đối tượng Pen thông qua lớp **tinh Pens**. Lớp này có các thuộc tính trả về các loại Pen với các màu sắc thông thường.

Pen pn1 = Pens.Red;

Pen pn2 = Pens.Blue;

Pen pn3 = Pens.Green

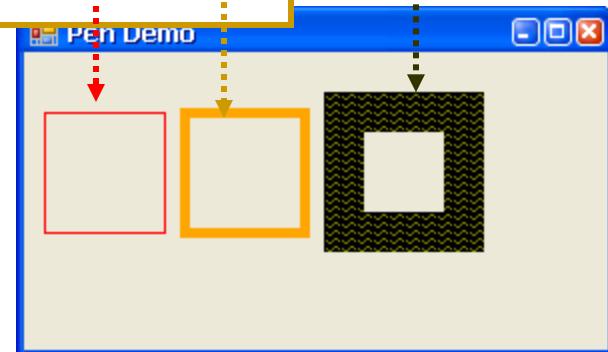
Pen

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;

    Pen p1 = new Pen(Color.Red);
    g.DrawRectangle(p1, new Rectangle(10, 30, 60, 60));

    Pen p2 = new Pen(Color.Orange, 5);
    g.DrawRectangle(p2, new Rectangle(80, 30, 60, 60));

    HatchBrush hb = new HatchBrush( HatchStyle.Wave, Color.Olive);
    Pen p3 = new Pen(hb, 20);
    g.DrawRectangle(p3, new Rectangle(160, 30, 60, 60));
}
```



Pen

- Có thể tạo các đối tượng Pen thông qua các constructor với tham số là một đối tượng Brush hoặc Color và có thể xác định độ dài của Pen

```
SolidBrush blueBrush = new SolidBrush(Color.Blue);
```

```
HatchBrush hatchBrush=
```

```
    new HatchBrush(HatchStyle.DashedVertical,  
    Color.Black, Color.Green);
```

```
Pen pn1 = new Pen( blueBrush, 3);
```

```
Pen pn2 = new Pen(hatchBrush, 8);
```

```
Pen pn3 = new Pen(Color.Red);
```

Pen

- Có thể lấy các đối tượng Pen thông qua lớp tính Pens. Lớp này có các thuộc tính trả về các loại Pen với các màu sắc thông thường.

Pen pn1 = Pens.Red;

Pen pn2 = Pens.Blue;

Pen pn3 = Pens.Green;

DrawLine

- DrawLine dùng để vẽ một đường thẳng.
- Một số constructor:

`public void DrawLine(Pen, Point, Point);`

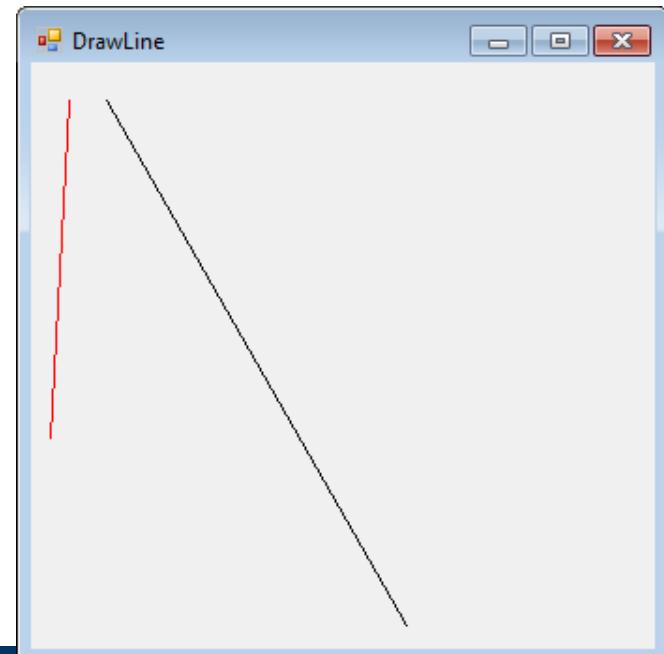
`public void DrawLine(Pen, PointF, PointF);`

`public void DrawLine(Pen, int, int, int, int);`

`public void DrawLine(Pen, float, float, float, float);`

DrawLine

```
Graphics g = e.Graphics;  
Point pt1 = new Point(20,20);  
Point pt2 = new Point(10, 200);  
g.DrawLine(Pens.Red, pt1, pt2);  
int x1 = 40, y1 = 20, x2 = 200, y2 = 300;  
g.DrawLine(Pens.Black, x1, y1, x2, y2);
```



DrawRectangle

- Dùng để vẽ các hình chữ nhật
- Một số constructor

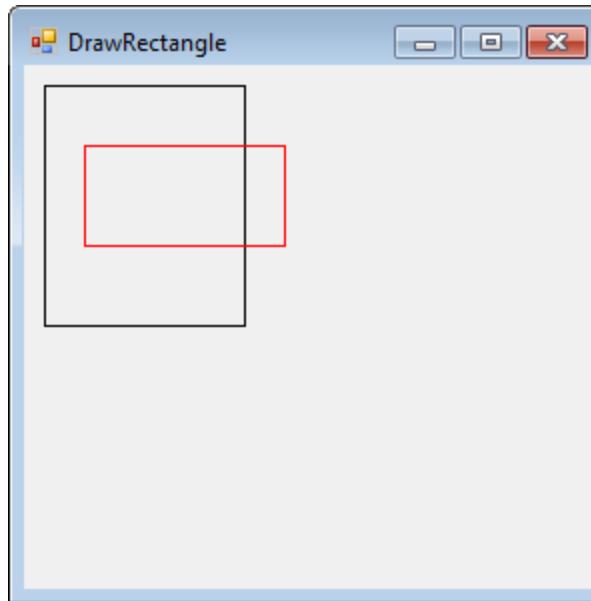
`public void DrawRectangle(Pen, Rectangle);`

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

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

DrawRectangle

```
Graphics g = e.Graphics;  
Rectangle rect = new Rectangle(10, 10, 100, 120);  
int x = 30, y = 40, width = 100, height = 50;  
g.DrawRectangle(Pens.Black, rect);  
g.DrawRectangle(Pens.Red, x, y, width, height);
```



DrawEllipse

- Để vẽ một hình ellipse cần xác định hình chữ nhật bao quanh hình ellipse này
- Một số constructor:

`public void DrawEllipse(Pen, Rectangle);`

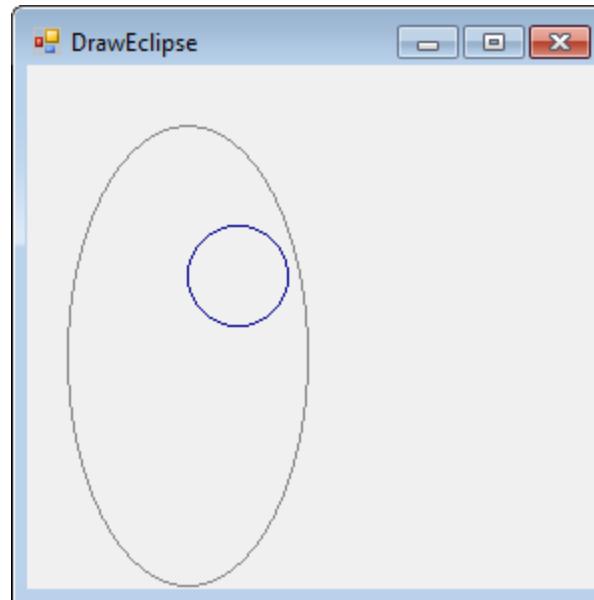
`public void DrawEllipse(Pen, RectangleF);`

`public void DrawEllipse(Pen, int, int, int, int);`

`public void DrawEllipse(Pen, float, float, float, float);`

DrawEllipse

```
Graphics g = e.Graphics;  
Rectangle rect = new Rectangle(80, 80, 50, 50);  
int x = 20, y = 30, width = 120, height = 230;  
g.DrawEllipse(Pens.DarkBlue, rect);  
g.DrawEllipse(Pens.Gray, x, y, width, height);
```



DrawArc

- Vẽ đường cung là vẽ một phần đường cong của hình ellipse do đó cần xác định một hình chữ nhật bao quanh, góc bắt đầu và góc quét.
- Một số constructor:

`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);`

DrawArc

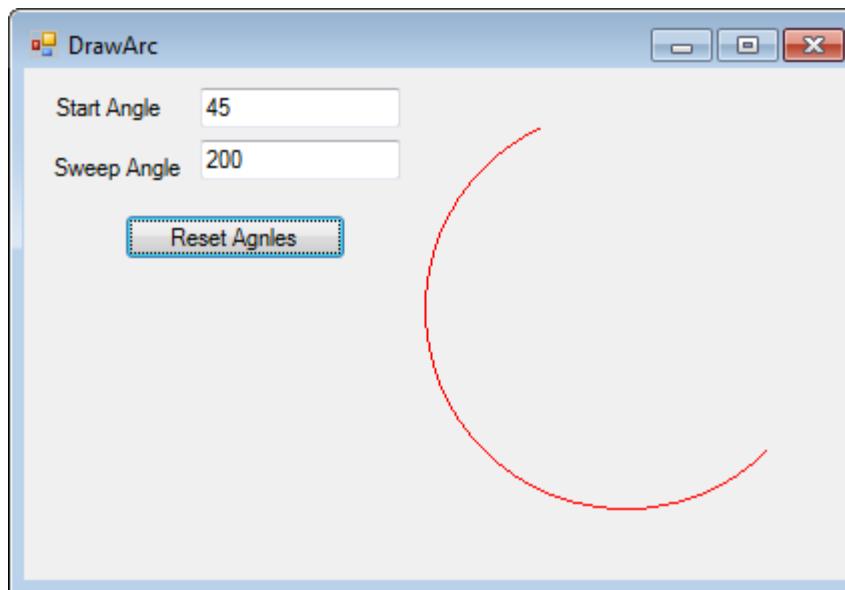
```
public partial class DrawArc : Form
{
    private float startAngle = 45.0f;
    private float sweepAngle = 90.0f;

    public DrawArc()
    {
        InitializeComponent();
    }

    private void DrawArc_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        Rectangle rect = new Rectangle(200, 20, 200, 200);
        g.DrawArc(Pens.Red, rect, startAngle, sweepAngle);
    }
}
```

DrawArc

```
private void button1_Click(object sender, EventArgs e)
{
    startAngle =(float)Convert.ToDouble(textBox1.Text);
    sweepAngle =(float)Convert.ToDouble(textBox2.Text);
    Invalidate();
}
}
```



DrawCurve

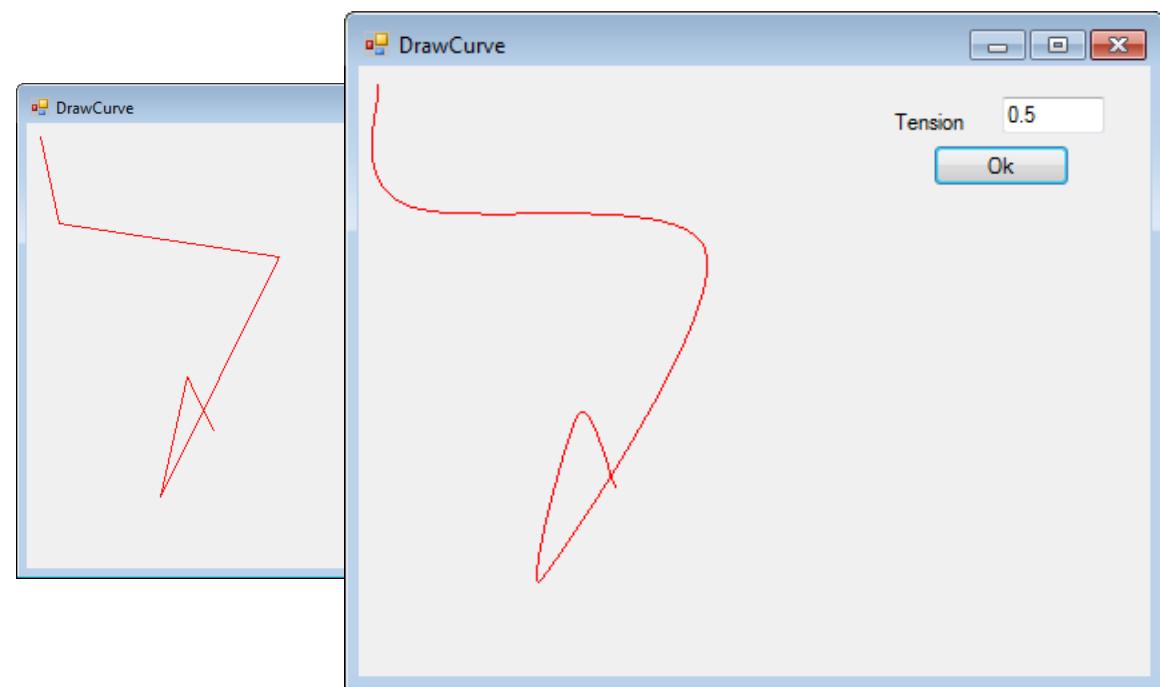
- Một Curve là một tập hợp các Point được liên kết với nhau.
- Khi vẽ một Curve ta có thể xác định độ căng giữa các điểm thông qua tham số tension
- Giá trị của tham số tension phải lớn hơn 0.0F

DrawCurve

```
private void DrawCurve_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Point pt1 = new Point(10, 10);
    Point pt2 = new Point(24, 75);
    Point pt3 = new Point(189, 100);
    Point pt4 = new Point(100, 280);
    Point pt5 = new Point(120, 190);
    Point pt6 = new Point(140, 230);
    Point[] ptsArray = { pt1, pt2, pt3, pt4, pt5, pt6 };
    g.DrawCurve(Pens.Red, ptsArray, tension);
}

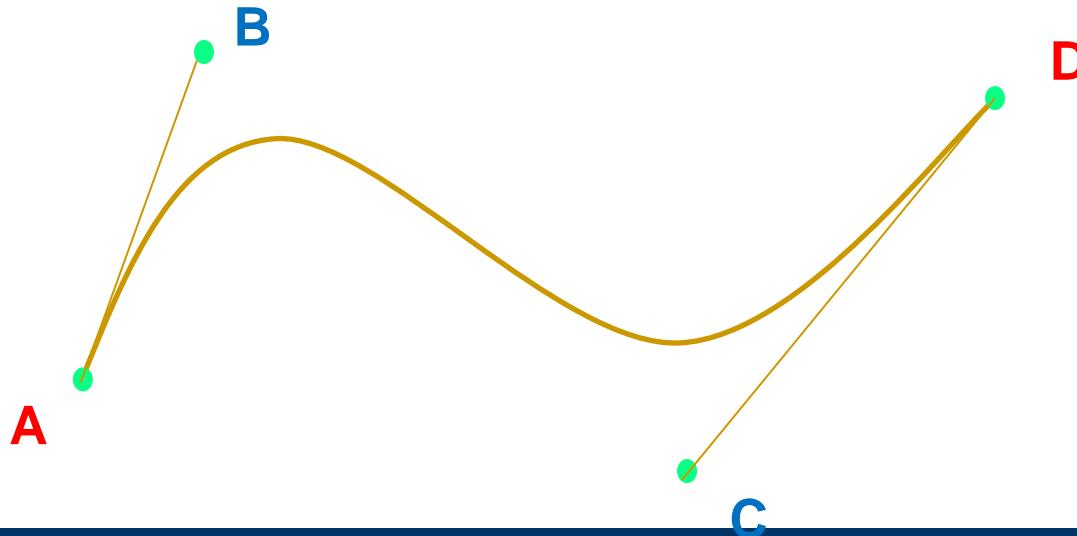
private void button1_Click(object sender, EventArgs e)
{
    tension = (float)Convert.ToDouble(textBox1.Text);
    Invalidate();
}
```

DrawCurve



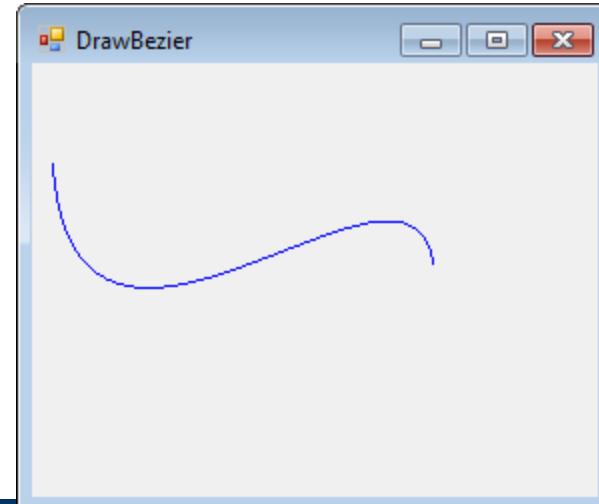
DrawBezier

- Đường cong Bezier được phát triển bởi Pierre Bézier (1960) là một trong những đường được sử dụng nhiều trong việc vẽ.
- Một đường cong Bezier gồm bốn Point: 2 Point đầu và cuối (A, D), 2 Point điều khiển (B, C)



DrawBezier

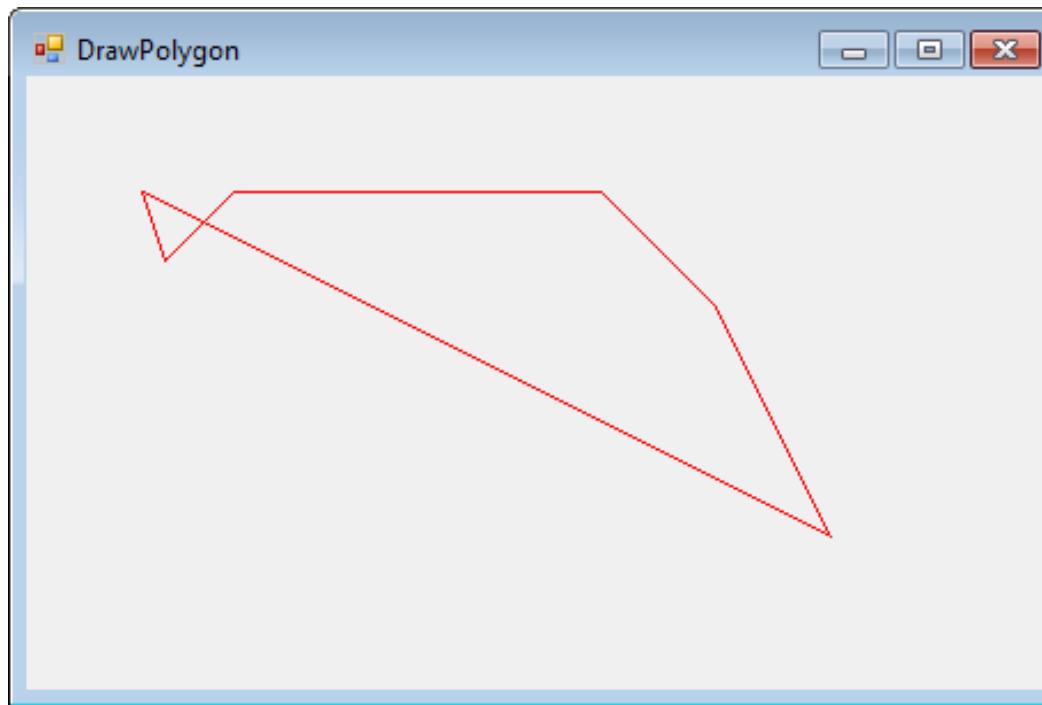
```
Graphics g = e.Graphics;  
Point pt1 = new Point(10, 50);  
Point pt2 = new Point(20, 200);  
Point pt3 = new Point(190, 20);  
Point pt4 = new Point(200, 100);  
g.DrawBezier(Pens.Blue, pt1, pt2, pt3, pt4);
```



DrawPolygon

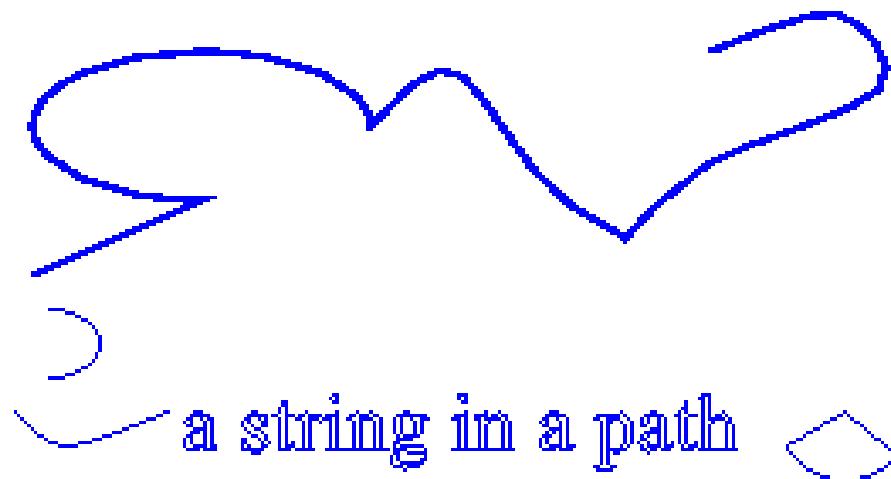
```
Graphics g = e.Graphics;  
Point pt1 = new Point(50, 50);  
Point pt2 = new Point(60, 80);  
Point pt3 = new Point(90, 50);  
Point pt4 = new Point(250, 50);  
Point pt5 = new Point(300, 100);  
Point pt6 = new Point(350, 200);  
Point[] ptsArray = { pt1, pt2, pt3, pt4, pt5, pt6 };  
g.DrawPolygon(Pens.Red, ptsArray);
```

DrawPolygon



DrawPath

- Graphics Path: kết hợp nhiều loại đường nét thành một đối tượng duy nhất. Các “nét” không nhất thiết phải liền nhau.



a string in a path

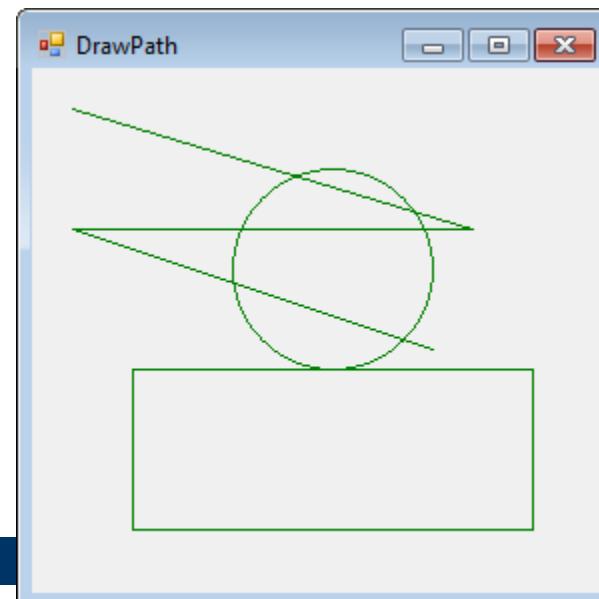
GraphicsPath (AddLine, AddCurve, ...)

Graphics.DrawPath

Graphics.FillPath

DrawPath

```
Graphics g = e.Graphics;  
GraphicsPath path = new GraphicsPath();  
path.AddEllipse(100, 50, 100, 100);  
path.AddLine(20, 20, 220, 80);  
path.AddLine(220, 80, 20, 80);  
path.AddLine(20, 80, 200, 140);  
Rectangle rect = new Rectangle(50, 150, 200, 80);  
path.AddRectangle(rect);  
g.DrawPath(Pens.Green, path);
```



DrawString()

■ Hiển thị text trong Graphics cụ thể

- Có nhiều phiên bản

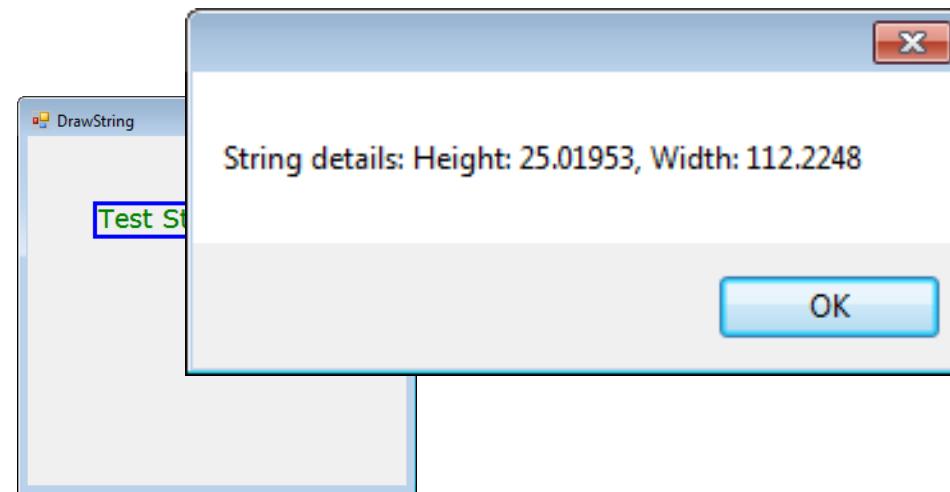
- `DrawString(String text, // Text thẻ hiện
Font f, // Font
Brush b, // Color & texture
Float x, y); // vị trí góc trái trên`

- Tham số Font và Brush không có mặc định nên phải truyền vào.
- Tạo các đối tượng Font chỉ định các thuộc tính chữ (như font, style, ...)
- Tạo pen và brush
- Để “đo” kích thước chuỗi (dài,rộng) , dùng `Graphics.MeasureString`

DrawString

```
Graphics g = this.CreateGraphics();
string testString = "Test String";
Font verdana14 = new Font("Verdana", 14);
SizeF sz = g.MeasureString(testString, verdana14);
string stringDetails = "Height: " + sz.Height.ToString()
                      + ", Width: " + sz.Width.ToString();
MessageBox.Show("String details: " + stringDetails);
g.DrawString(testString, verdana14, Brushes.Green, new
             PointF(50, 50));
g.DrawRectangle(new Pen(Color.Blue, 3), 50, 50, sz.Width,
               sz.Height);
g.Dispose();
```

DrawString

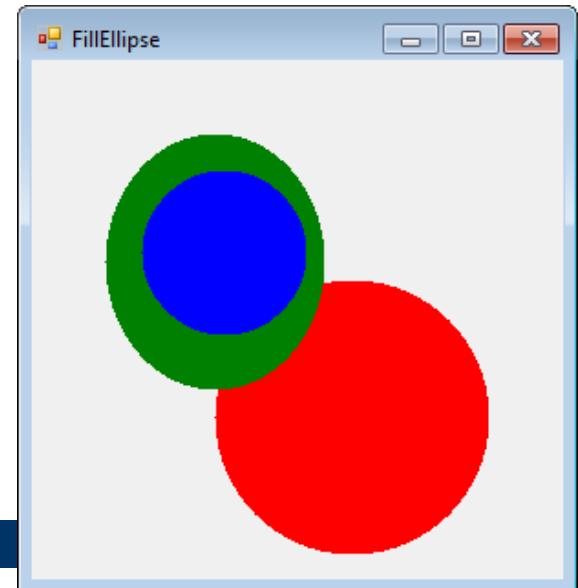


Các Phương thức Fill

- **FillClosedCurve**
- **FillEllipse**
- **FillPath**
- **FillPolygon**
- **FillRectangle**
- **FillRectangles**
- **FillRegion**

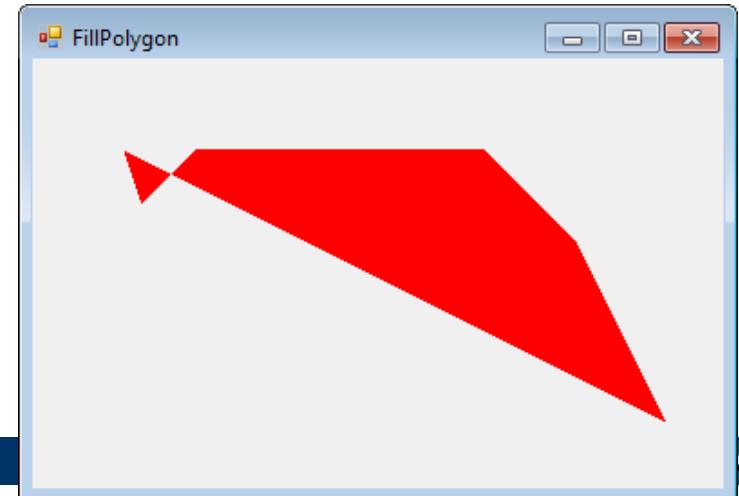
Các Phương thức Fill

```
Graphics g = e.Graphics;  
SolidBrush redBrush = new SolidBrush(Color.Red);  
SolidBrush blueBrush = new SolidBrush(Color.Blue);  
SolidBrush greenBrush = new SolidBrush(Color.Green);  
Rectangle rect = new Rectangle(100, 120, 150, 150);  
g.FillEllipse(redBrush, rect);  
g.FillEllipse(greenBrush, 40, 40, 120, 140);  
g.FillEllipse(blueBrush, 60, 60, 90, 90);
```



Các Phương thức Fill

```
Graphics g = e.Graphics;  
SolidBrush redBrush = new SolidBrush(Color.Red);  
Point pt1 = new Point(50, 50);  
Point pt2 = new Point(60, 80);  
Point pt3 = new Point(90, 50);  
Point pt4 = new Point(250, 50);  
Point pt5 = new Point(300, 100);  
Point pt6 = new Point(350, 200);  
Point[] ptsArray = { pt1, pt2, pt3, pt4, pt5, pt6 };  
g.FillPolygon(redBrush, ptsArray);
```



Brush

- Brush dùng để tô vùng bên trong của một hình
- Lớp Brush là một lớp Abstract
- Các lớp kế thừa từ lớp Brush
 - SolidBrush
 - LinearGradientBrush
 - TextureBrush
 - HatchBrush

SolidBrush

- Một Solid Brush là một brush dùng để tô một vùng với một màu đơn.

```
Graphics g = e.Graphics;
```

```
SolidBrush redBrush = new SolidBrush(Color.Red);
```

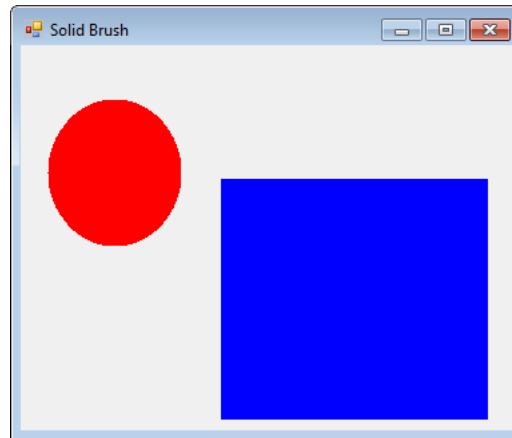
```
SolidBrush blueBrush = new SolidBrush(Color.Blue);
```

```
g.FillEllipse(redBrush, 20, 40, 100, 110);
```

```
Rectangle rect = new Rectangle(150, 100, 200, 180);
```

```
g.FillRectangle(blueBrush, rect);
```

SolidBrush



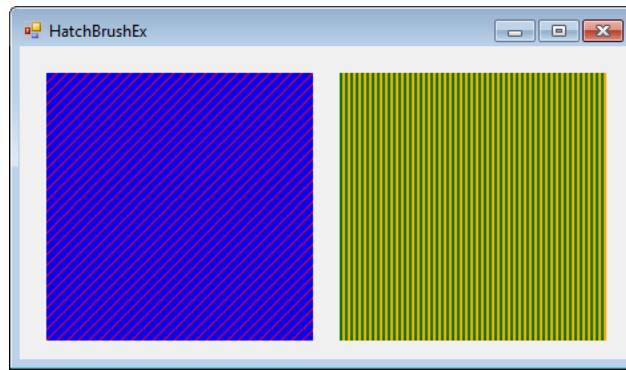
HatchBrush

- Hatch brush là brush được sử dụng dựa trên các kiểu Hatch Style, foreground color và background color.
- Một kiểu trong Hatch Style
 - BackwardDiagonal
 - Cross
 - DiagonalCross
 - HorizontalBrick
 - LightDownwardDiagonal
 - LightUpwardDiagonal
 -

HatchBrush

```
Graphics g = e.Graphics;  
HatchStyle style = HatchStyle.BackwardDiagonal;  
Color forColor = Color.Red;  
Color bgrndColor = Color.Blue;  
HatchBrush brush = new HatchBrush(style, forColor,  
                                  bgrndColor);  
g.FillRectangle(brush, 20, 20, 200, 200);  
style = HatchStyle.DarkVertical;  
forColor = Color.Green;  
bgrndColor = Color.Orange;  
brush = new HatchBrush(style, forColor, bgrndColor);  
g.FillRectangle(brush, 240, 20, 200, 200);
```

HatchBrush



TextureBrush

- Texture Brush cho phép sử dụng ảnh như là một brush để tô các đối tượng

```
private TextureBrush brush = null;
```

```
private void TextureBrushEx_Load(object sender, EventArgs e)
```

```
{
```

```
    Image img = new Bitmap(Brush.Properties.Resources.img);
```

```
    brush = new TextureBrush(img);
```

```
}
```

```
private void TextureBrushEx_Paint(object sender, PaintEventArgs e)
```

```
{
```

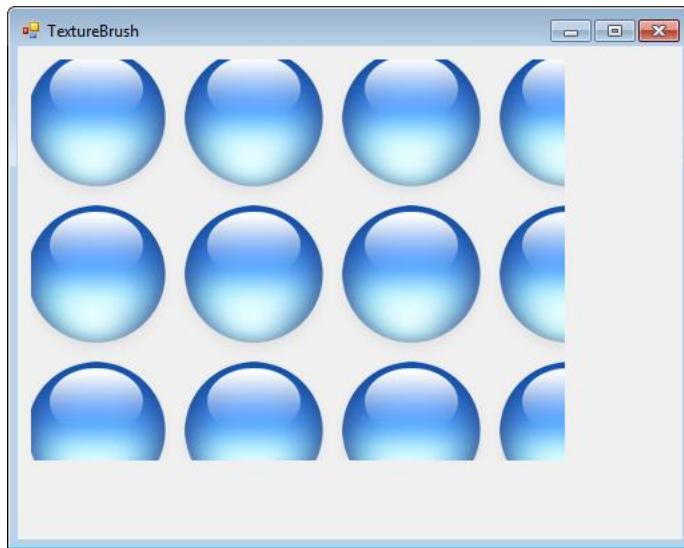
```
    Graphics g = e.Graphics;
```

```
    Rectangle rect = new Rectangle(10, 10, 400, 300);
```

```
    g.FillRectangle(brush, rect);
```

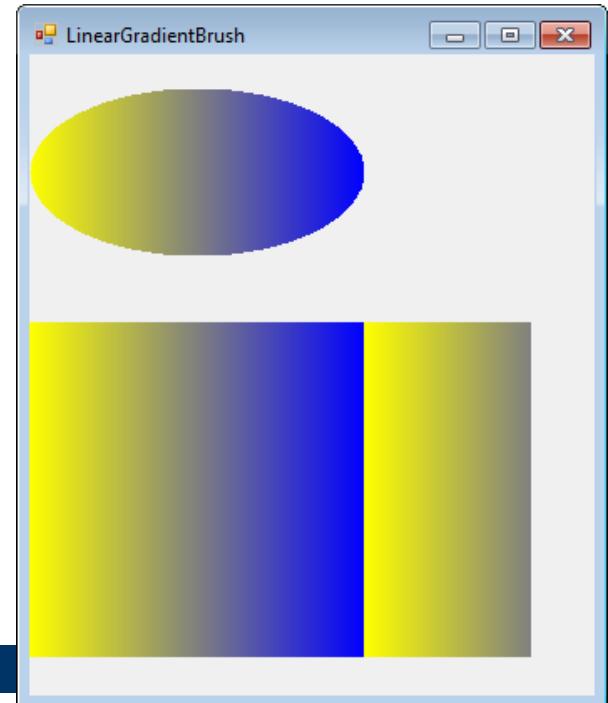
```
}
```

TextureBrush



LinearGradientBrush

```
Point pt1 = new Point(0,10);
Point pt2 = new Point(200, 10);
LinearGradientBrush brush = new LinearGradientBrush(pt1,
pt2, Color.Yellow, Color.Blue);
e.Graphics.FillEllipse(brush, 0, 20, 200, 100);
e.Graphics.FillRectangle(brush, 0, 160, 300, 200);
```



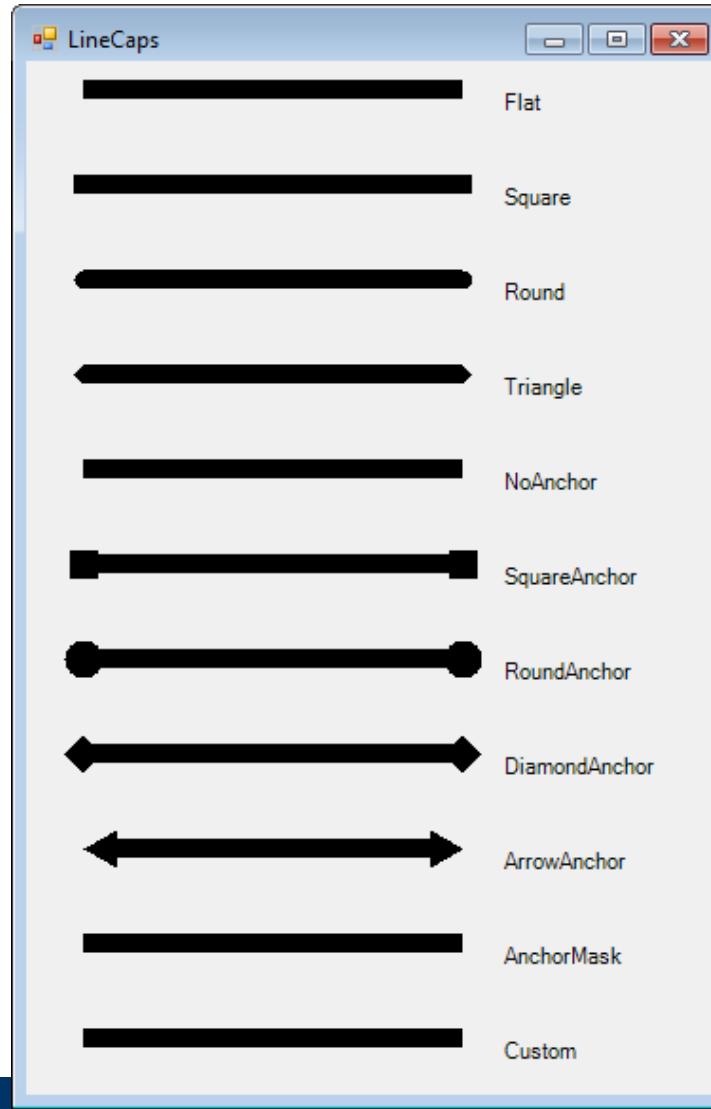
LineCap

- Tính chất LineCap của lớp Pen được sử dụng để xác định kiểu vẽ điểm đầu và điểm cuối của các dòng vẽ bởi Pen
- GetLineCap(): trả về một enum LineCap
- SetLineCap(): áp dụng một LineCap cho Pen

LineCap

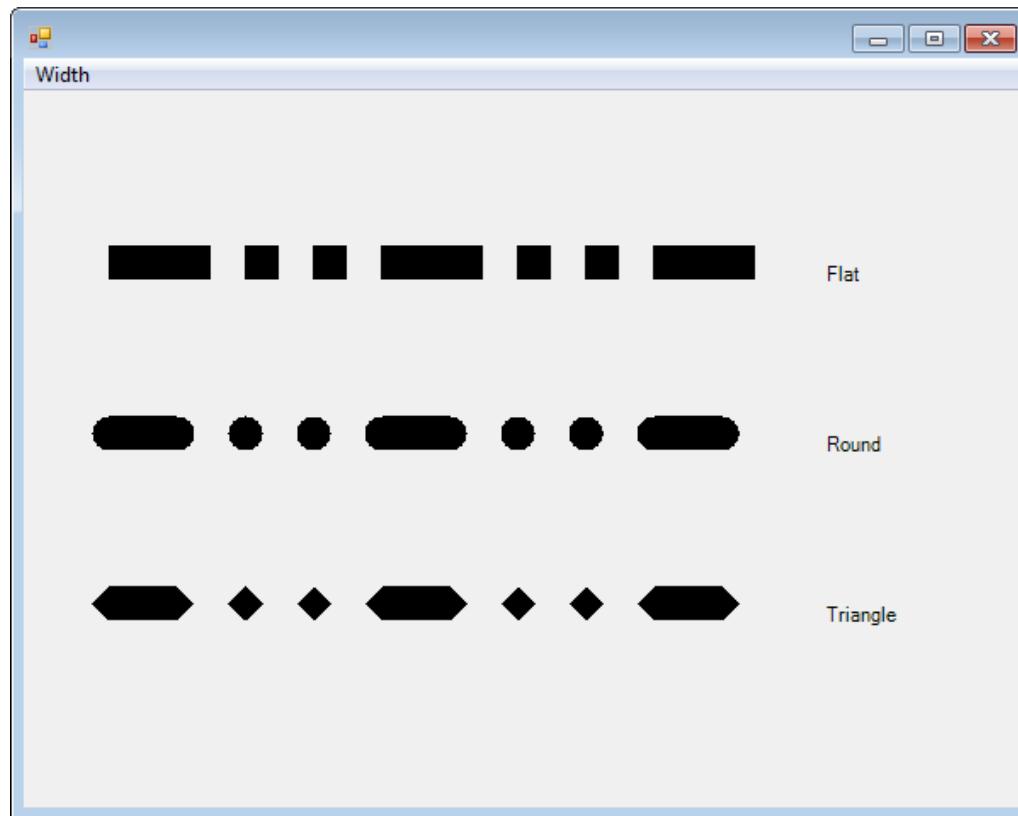
■ Một số kiểu LineCap:

- Member
- AnchorMask
- ArrowAnchor
- Custom
- DiamondAnchor
- Flat
- NoAnchor
- Round
- RoundAnchor
- Square
- SquareAnchor
- Triangle



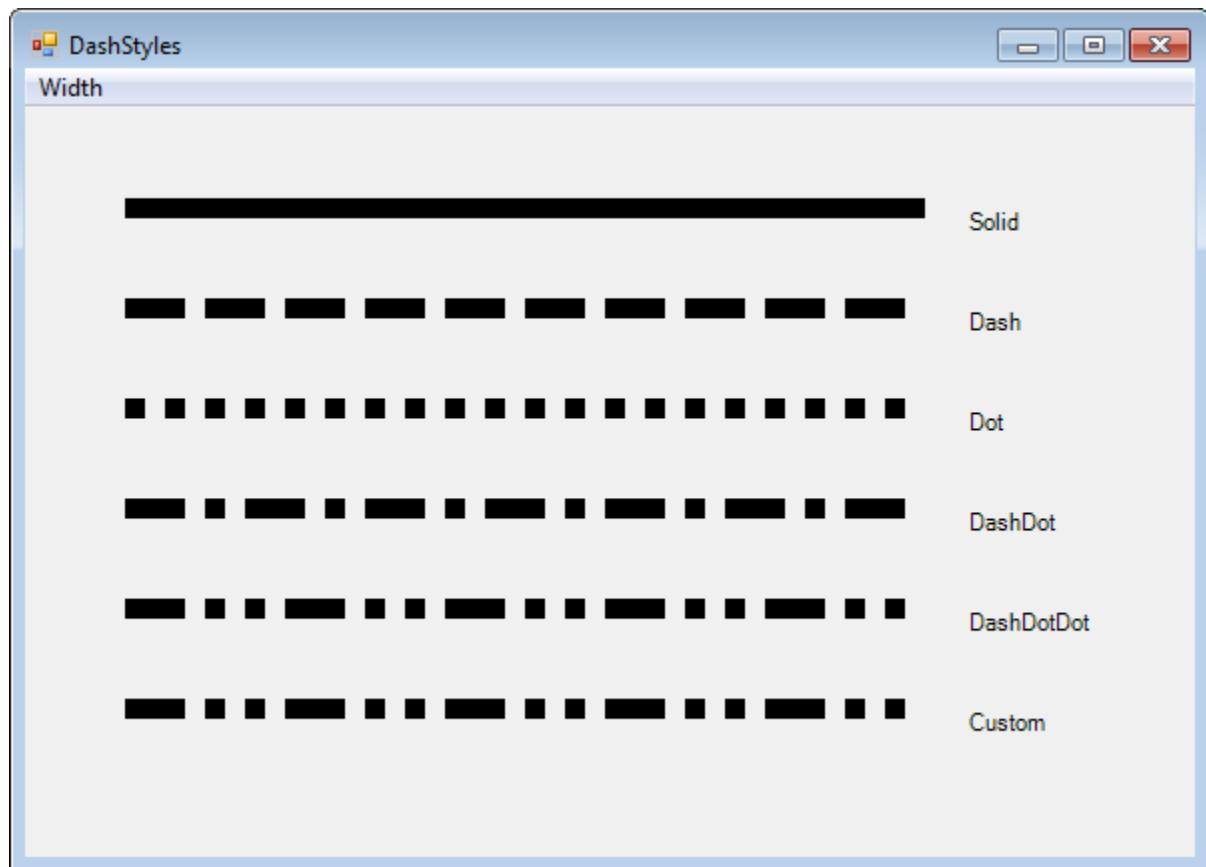
DashCap

- Flat
- Round
- Triangle



DashStyles

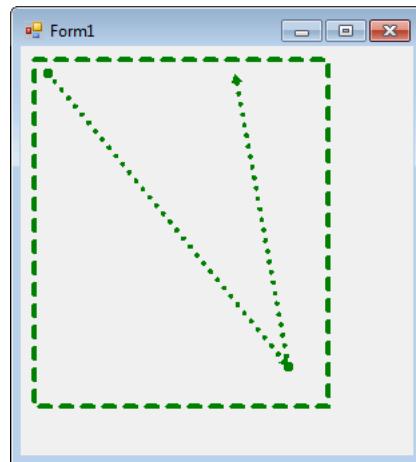
- Custom
- Dash
- DashDot
- DashDotDot
- Dot
- Solid



Ví dụ

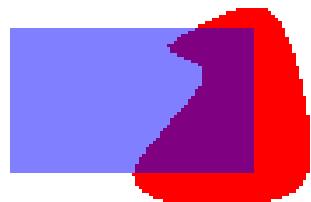
```
Graphics g = e.Graphics;  
Pen pen = new Pen(Color.Green);  
pen.Width = 4.0F;  
pen.StartCap = LineCap.RoundAnchor;  
pen.EndCap = LineCap.ArrowAnchor;  
pen.DashStyle = DashStyle.Dot;  
  
g.DrawLine(pen, 20.0F, 20.0F, 200.0F, 240.0F);  
g.DrawLine(pen, 200.0F, 240.0F, 160.0F, 20.0F);  
  
pen.DashStyle = DashStyle.Dash;  
Rectangle rect = new Rectangle(10, 10, 220, 260);  
g.DrawRectangle(pen, rect);
```

Ví dụ

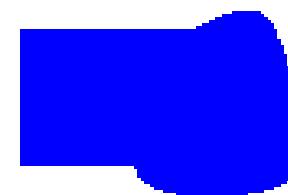


Region

- Region: một vùng được tạo ra bằng các phép kết giữa các hình chữ nhật hoặc path. Region thường được dùng cho “hit-test” hoặc “clipping”



Intersection



Union

System.Drawing.Drawing2D

**Region.Intersect, Union, Xor,
Exclude, Complement**

Region

- Một số phương thức áp dụng giữa các region
 - Region1.Complement(Region2): lấy vùng nằm ở Region 2 và không thuộc Region 1
 - Region1.Exclude(Region2): lấy vùng ở Region 1 và không thuộc Region 2
 - Region1.Union(Region2): lấy vùng hợp của 2 Region
 - Region1.Xor(Region2): Lấy vùng hợp của 2 Region và bỏ phần giao giữa hai Region
 - Region1.Intersect(Region2): Lấy vùng giao giữa 2 Region

Region

```
Graphics g = e.Graphics;
```

```
Rectangle rect1 = new Rectangle(10, 10, 120, 140);
```

```
Rectangle rect2 = new Rectangle(80, 50, 160, 200);
```

```
Region rgn1 = new Region(rect1);
```

```
Region rgn2 = new Region(rect2);
```

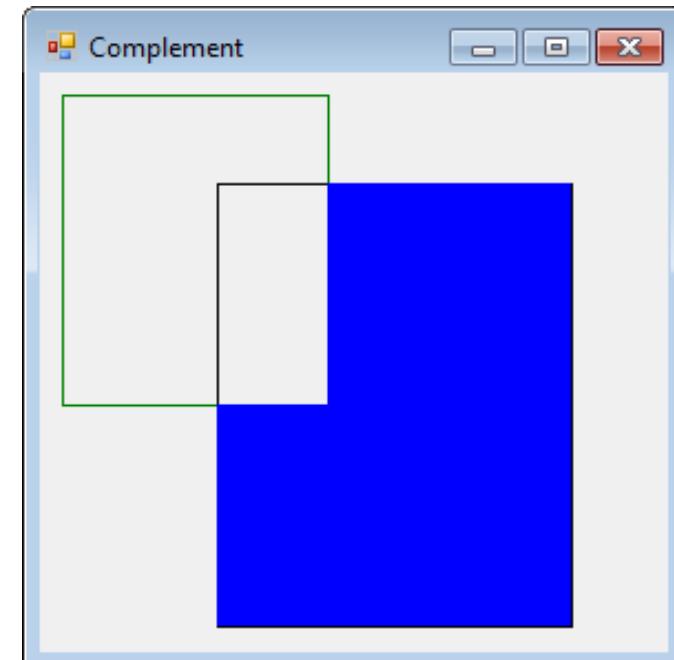
```
g.DrawRectangle(Pens.Green, rect1);
```

```
g.DrawRectangle(Pens.Black, rect2);
```

```
rgn1.Complement(rgn2);
```

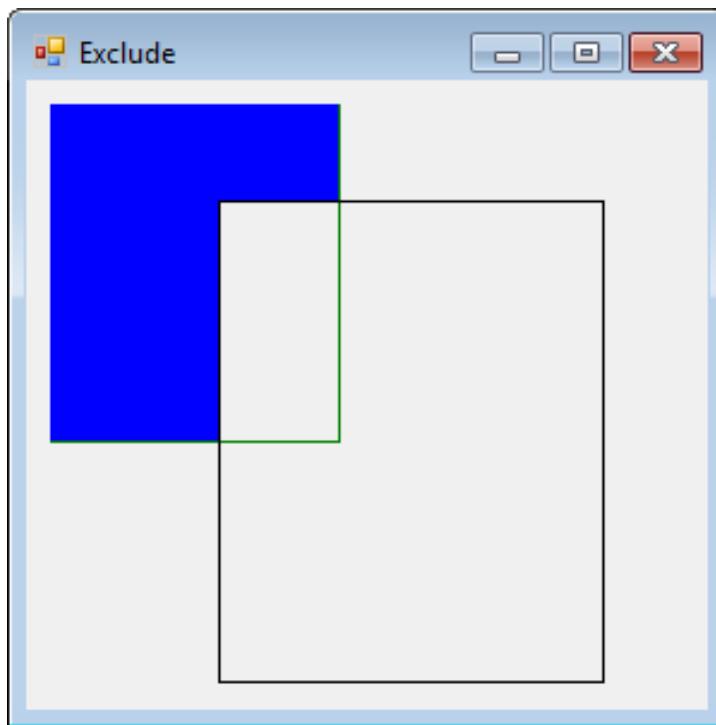
```
g.FillRegion(Brushes.Blue, rgn1);
```

```
g.Dispose();
```

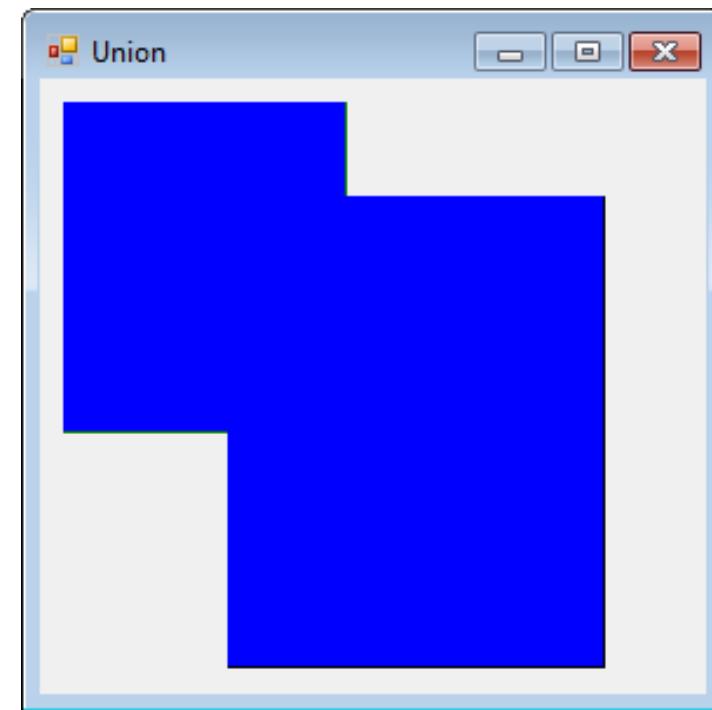


Region

`rgn1.Exclude(rgn2);`



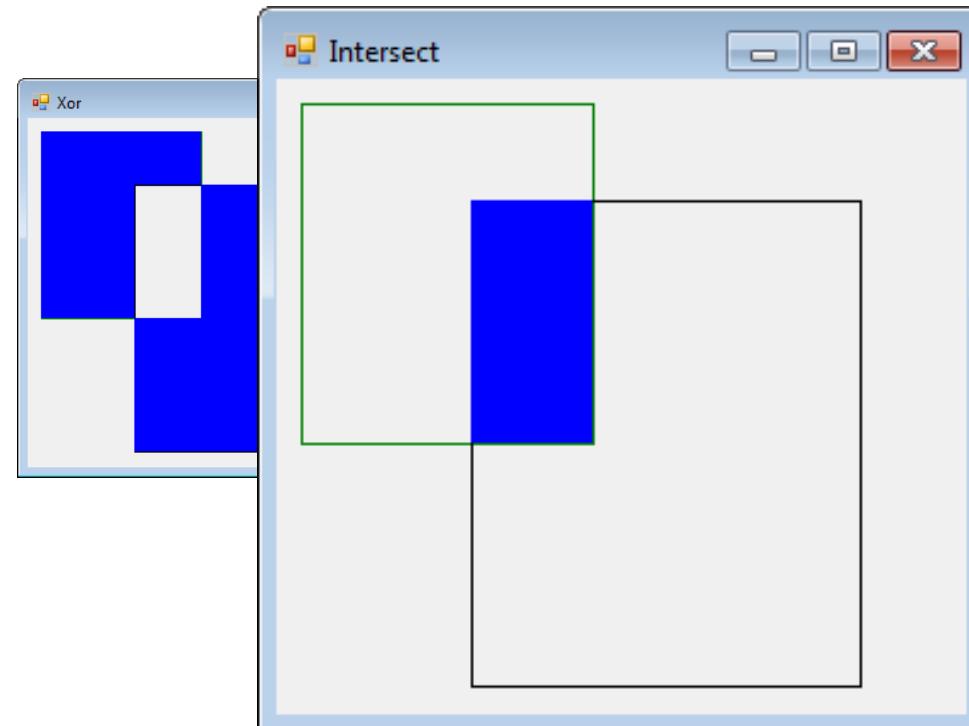
`rgn1.Union(rgn2);`



Region

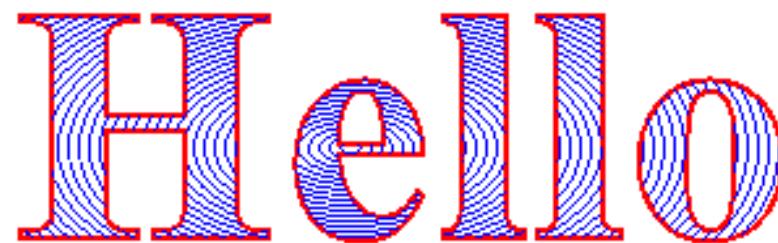
`rgn1.Xor(rgn2);`

`rgn1.Intersect(rgn2);`



Clipping

- Clipping: giới hạn các hình vẽ vào trong một region, path hoặc rectangle



Graphics.SetClip(<region>)

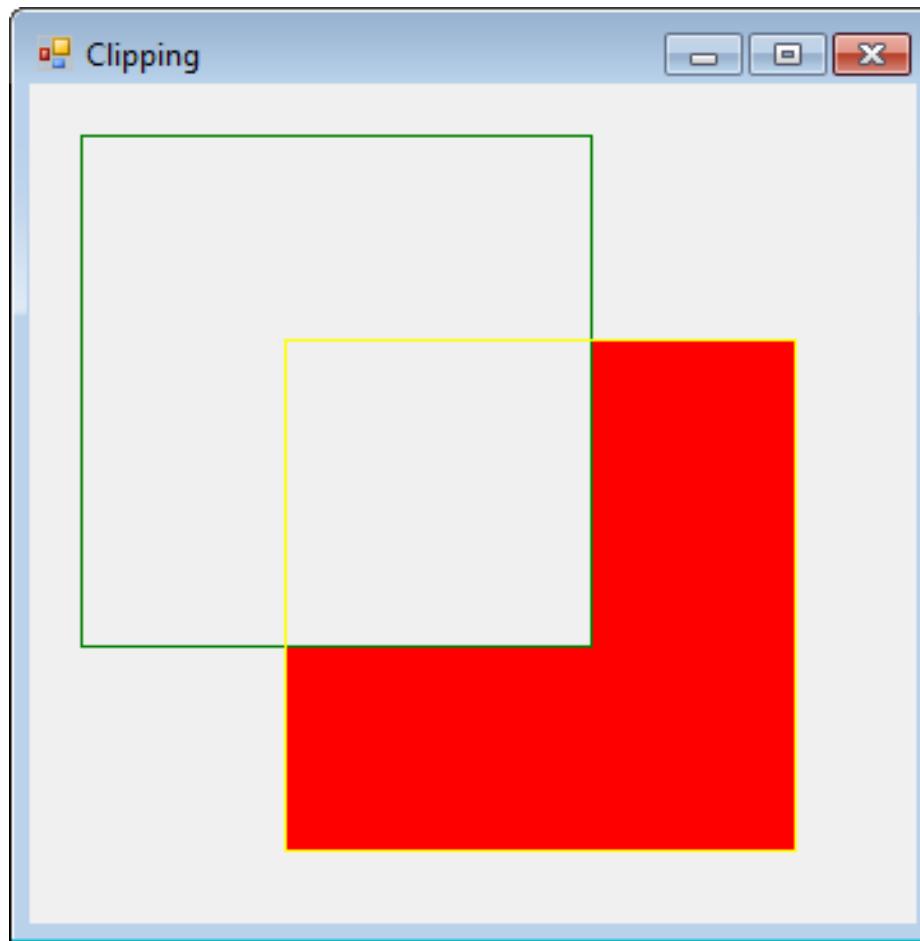
Graphics.SetClip(<path>)

Graphics.SetClip(<rectangle>)

Clipping

```
Graphics g = e.Graphics;  
Rectangle rect1 = new Rectangle(20, 20, 200, 200);  
Rectangle rect2 = new Rectangle(100, 100, 200, 200);  
Region rgn1 = new Region(rect1);  
Region rgn2 = new Region(rect2);  
g.SetClip(rgn1, CombineMode.Exclude);  
g.IntersectClip(rgn2);  
g.FillRectangle(Brushes.Red, 0, 0, 300, 300);  
g.ResetClip();  
g.DrawRectangle(Pens.Green, rect1);  
g.DrawRectangle(Pens.Yellow, rect2);
```

Clipping



Image

- Cho phép vẽ các hình ảnh
 - Tạo các hình ảnh thông qua class Image (Bitmap, Metafile, Icon, ...)
 - Class Bitmap hỗ trợ các định dạng chuẩn GIF, JPG, BMP, PNG, TIFF.
 - Dùng **Graphics.DrawImage**, **DrawIcon**,
DrawIconUnstretched, **DrawImageUnscaled**

```
Graphics g = e.Graphics;
```

```
Image curImage = Image.FromFile(curFileName);
```

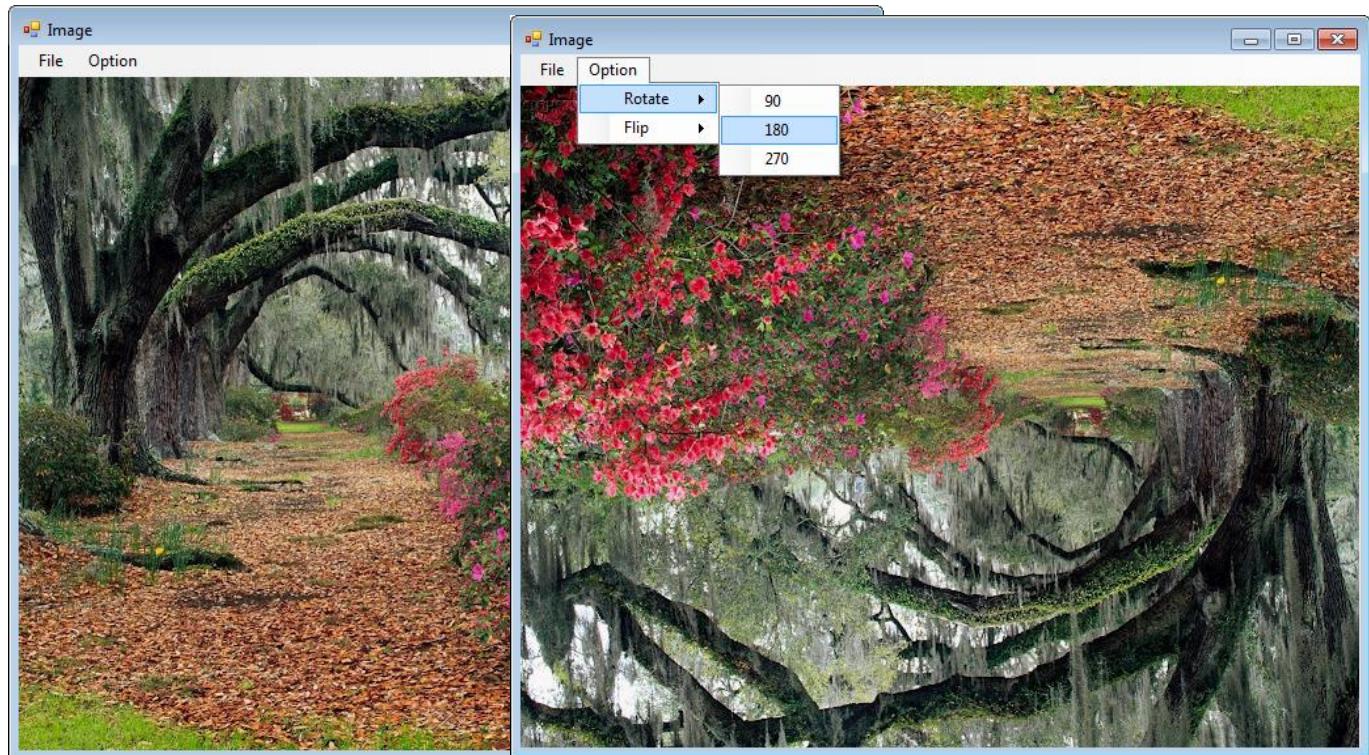
```
Rectangle rect = new Rectangle(20, 20, 100, 100);
```

```
g.DrawImage(curImage, rect);
```

Image

- Lớp Image còn cung cấp phương thức RotateFlip dùng để quay và flip ảnh.
- RotateFlipType xác định kiểu quay của đối tượng.
 - RotateFlipType.Rotate90FlipNone: Quay 90 độ
 - RotateFlipType.Rotate180FlipNone: Quay 180 độ
 - RotateFlipType.RotateNoneFlipX: Flip theo chiều X
 - RotateFlipType.RotateNoneFlipY: Flip theo chiều Y
 -

Image



Bitmap

■ Bitmap

- **Bitmap bmp = new Bitmap(filename, ...)**
- **MakeTransparent:** đặt màu trong suốt.
- **GetPixel, SetPixel:** vẽ bằng cách chấm từng điểm!

```
Image image = Image.FromFile("myImage.bmp");
```

```
// Tạo một bitmap từ một file
```

```
Bitmap bitmap1 = new Bitmap("myImage.bmp");
```

```
// Tạo một bitmap từ một đối tượng Image
```

```
Bitmap bitmap2 = new Bitmap(image);
```

```
// Tạo một bitmap với size
```

```
Bitmap curBitmap3 = new Bitmap(curlImage, new Size(200, 100));
```

```
// Tạo một bitmap không có dữ liệu ảnh
```

```
Bitmap curBitmap4 = new Bitmap(200, 100);
```

Bitmap

- Thay đổi màu sắc của một phần bức ảnh sử dụng SetPixel()

```
Graphics g = e.Graphics;
```

```
Bitmap curBitmap = new Bitmap("Image.jpg");
```

```
g.DrawImage(curBitmap, 0, 0, curBitmap.Width, curBitmap.Height);
```

```
for (int i = 100; i < 200; i++)
```

```
{
```

```
    for (int j = 100; j < 200; j++)
```

```
{
```

```
    Color curColor = curBitmap.GetPixel(i, j);
```

```
    int ret = (curColor.R + curColor.G + curColor.B) / 3;
```

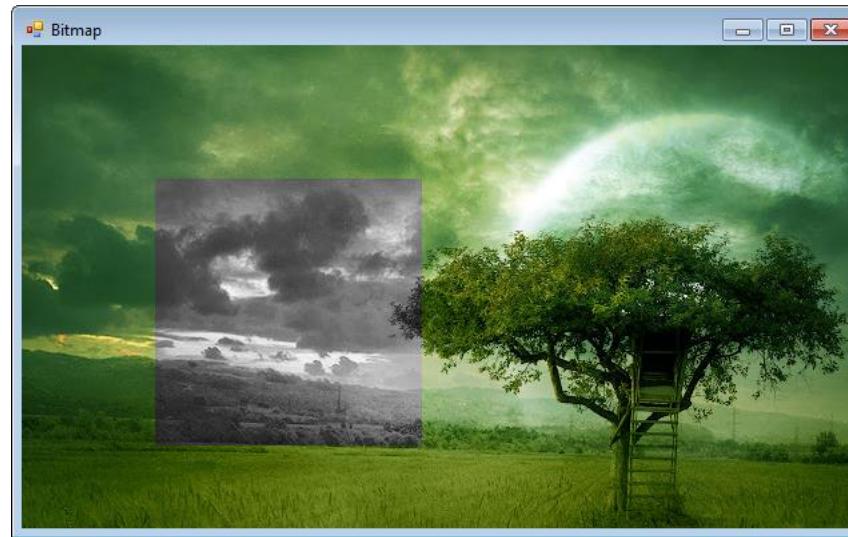
```
    curBitmap.SetPixel(i, j, Color.FromArgb(ret, ret, ret));
```

```
}
```

```
}
```

```
g.DrawImage(curBitmap, 0, 0, curBitmap.Width, curBitmap.Height);
```

Bitmap



Ý tưởng tạo animation với GDI+

Xóa cũ - vẽ mới là sai lầm!

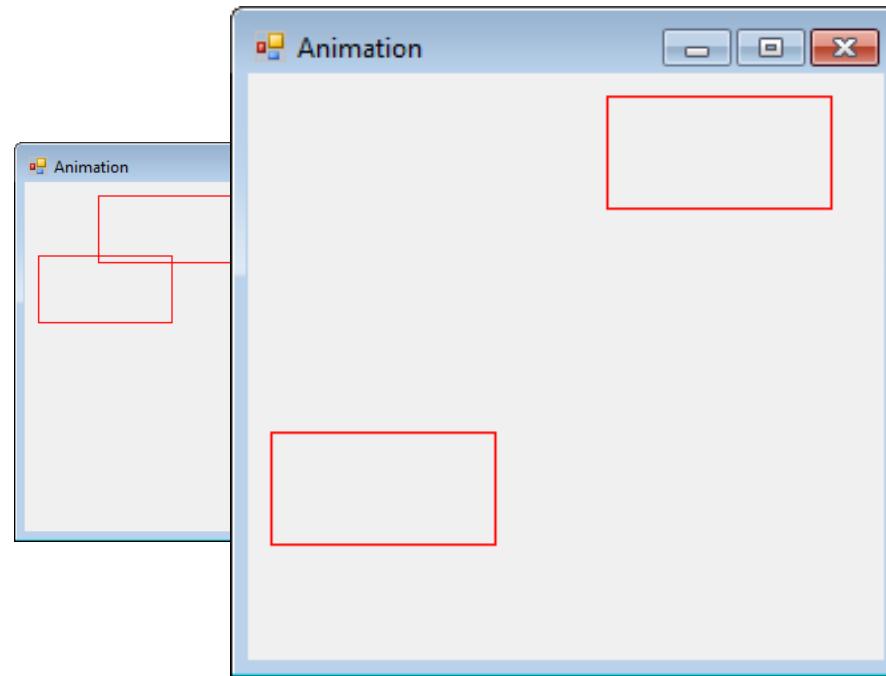
Frame-based animation: vẽ lại toàn bộ form theo tốc độ nhất định.
Kiểm soát bằng các biến trạng thái. (chú ý: timer: đặt thuộc tính enable = True)

```
protected int x=0;  
protected int y=0;
```

...

```
private void Form1_Paint(object sender, PaintEventArgs e) {  
    Graphics g = e.Graphics;  
    Pen pen = new Pen(Color.Red);  
    g.DrawRectangle(pen, x, 10, 100, 50);  
    g.DrawRectangle(pen, 10, y, 100, 50);  
}  
private void timer1_Tick(object sender, EventArgs e) {  
    x = (x + 1) % 200; y = (y+1) % 200;  
    Refresh();
```

Ý tưởng tạo animation với GDI+



Giải quyết nháy hình bằng double-buffer

- Một khi sự kiện Paint được gọi, các đối tượng sẽ được vẽ trực tiếp lên màn hình và khi các dữ liệu vẽ đối tượng chưa được nạp đầy đủ lên bộ đệm màn hình sẽ gây hiện tượng flicker (nháy hình).
- Nháy hình thường xảy ra trong các trường hợp sau:
 - Resize màn hình hoặc control
 - Tọa độ đối tượng vẽ đối tượng thay đổi hay chuyển động của đối tượng
 - Drag và Drop thả một đối tượng

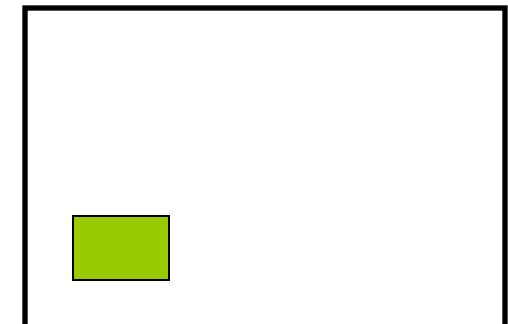
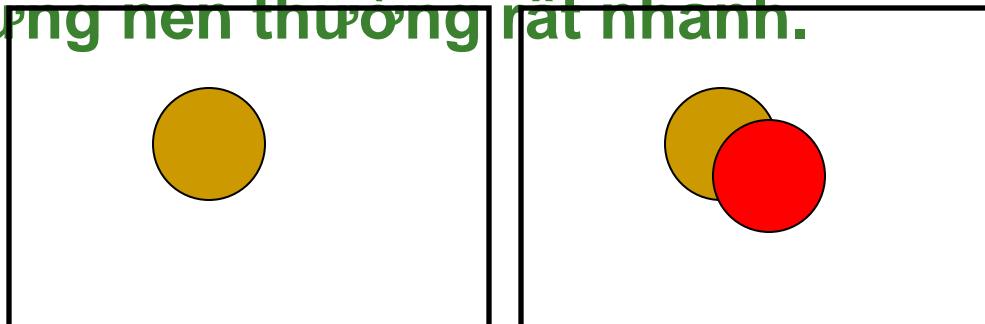
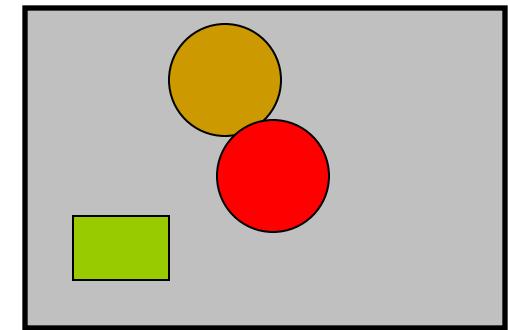
Giải quyết nháy hình bằng double-buffer

Để giải quyết nháy hình ta sử dụng kỹ thuật double buffer.

Mọi thao tác vẽ sẽ diễn ra tại back-buffer.

Khi hoàn tất, nội dung của back-buffer được hoán chuyển (flip) lên front-buffer.

*Thao tác flip được thực hiện bằng phần cứng nên thường rất nhanh.



Giải quyết nháy hình bằng double-buffer

- Có thể sử dụng kỹ thuật double buffer tự động trên form bằng cách xác lập thuộc tính DoubleBuffered bằng true hoặc gọi phương thức SetStyle với các và xác lập cờ OptimizedDoubleBuffer bằng true.
 - `DoubleBuffered = true;`
 - `SetStyle(ControlStyles.OptimizedDoubleBuffer, true);`
- Nếu không muốn hệ thống tự động sử dụng double buffer và thay vào đó muốn tạo một hệ thống buffer riêng, ta có thể tạo sử dụng double buffer một cách thủ công.

Giải quyết nháy hình bằng double-buffer

```
private Bitmap _backBuffer;  
float _angle;  
bool _doBuffer;  
private Timer timer1;  
  
public Form1() {...}  
// Dùng timer để thay đổi góc chuyển động  
private void timer1_Tick(object sender, System.EventArgs e)  
{  
    _angle += 3;  
    if (_angle > 359)  
        _angle = 0;  
    Invalidate();  
}
```

Giải quyết nháy hình bằng double-buffer

```
protected override void OnPaint(PaintEventArgs e)
{
    // Tạo một back buffer
    if (_backBuffer == null)
    {
        _backBuffer = new Bitmap(this.ClientSize.Width,
                               this.ClientSize.Height);
    }
    // Khởi tạo đối tượng Graphics
    Graphics g = null;
    if (_doBuffer)
        // Lấy đối tượng Graphics để vẽ lên back buffer
        g = Graphics.FromImage(_backBuffer);
    else
        g = e.Graphics;
```

Giải quyết nháy hình bằng double-buffer

```
g.Clear(Color.White);

g.SmoothingMode = SmoothingMode.AntiAlias;

int w = this.ClientSize.Width / 2;

int h = this.ClientSize.Height / 2;

// Chuyển động các đối tượng bằng cách xoay

Matrix mx = new Matrix();

mx.Rotate(_angle, MatrixOrder.Append);

mx.Translate(w, h, MatrixOrder.Append);

g.Transform = mx;

g.FillRectangle(Brushes.Red, -100, -100, 200, 200);
```

Giải quyết nháy hình bằng double-buffer

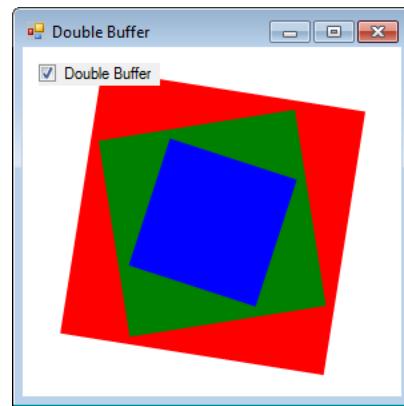
```
mx = new Matrix();
mx.Rotate(-_angle, MatrixOrder.Append);
mx.Translate(w, h, MatrixOrder.Append);
g.Transform = mx;
g.FillRectangle(Brushes.Green, -75, -75, 149, 149);
```

```
mx = new Matrix();
mx.Rotate(_angle * 2, MatrixOrder.Append);
mx.Translate(w, h, MatrixOrder.Append);
g.Transform = mx;
g.FillRectangle(Brushes.Blue, -50, -50, 100, 100);
```

Giải quyết nháy hình bằng double-buffer

```
// Nếu checkbox được chọn vẽ lên màn hình bằng back buffer  
if (_doBuffer)  
{  
    g.Dispose();  
    e.Graphics.DrawImageUnscaled(_backBuffer, 0, 0);  
}  
}
```

Giải quyết nháy hình bằng double-buffer



Giải quyết nháy hình bằng double-buffer

- Ta cũng có thể sử dụng **BufferedGraphicsContext** và **BufferedGraphics** (Hỗ trợ từ .NET Framework 2 trở đi)

```
// Khởi tạo
```

```
BufferedGraphicsContext currentContext;
```

```
BufferedGraphics myBuffer;
```

```
// Lấy một tham chiếu của BufferedGraphicsContext
```

```
currentContext = BufferedGraphicsManager.Current;
```

```
// Tạo một buffer với kích cỡ là bề mặt form;
```

```
myBuffer = currentContext.Allocate(this.CreateGraphics(),  
                                   this.DisplayRectangle);
```

```
// Vẽ một ellipse
```

```
myBuffer.Graphics.DrawEllipse(Pens.Blue, this.DisplayRectangle);
```

```
// Render nội dung
```

```
myBuffer.Render();
```

Sprites

- Mỗi bitmap một frame => nạp hình nhiều lần, khó quản lý.
- Dùng một hình lớn chứa nhiều hình nhỏ kích thước bằng nhau (sprites)



- Hàm DrawImage cho phép vẽ một phần hình chữ nhật của image lên Graphic
- Xem
 - <http://www.codeproject.com/vcpp/gdiplus/imageexgdi.asp>
- để biết cách extract frames (sprites) từ animated GIF files

Sprites

- Ví dụ sử dụng một ảnh sprite.

// Khởi tạo

// Bitmap dùng cho ảnh sprite

private Bitmap sprite;

// Back buffer

private Bitmap backBuffer;

private Timer timer;

public Graphics graphics;

// Số thứ tự của frame (16 frame ảnh)

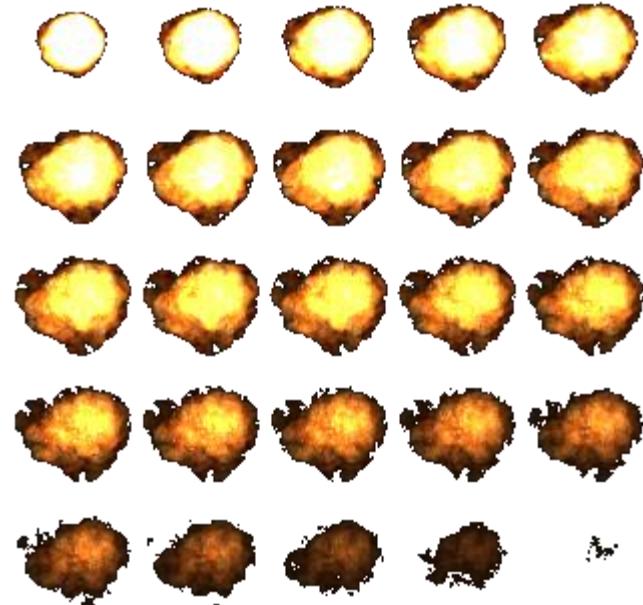
private int index;

// dòng hiện tại của frame

private int curFrameColumn;

// cột hiện tại của frame

private int curFrameRow;



Sprites

```
public SpriteFrm()
{
    InitializeComponent();
    graphics = this.CreateGraphics();
    SetStyle(ControlStyles.AllPaintingInWmPaint, true);
    // Tạo back buffer
    backBuffer = new Bitmap(this.ClientSize.Width,
                           this.ClientSize.Height);
    // Lấy ảnh sprite
    sprite = new Bitmap("Sprite.png");
    index = 0;
    // Khởi tạo một đồng hồ
    timer = new Timer();
    timer.Enabled = true;
    timer.Interval = 60;
    timer.Tick += new EventHandler(timer_Tick);
}
```

Sprites

```
// Vẽ một phần của ảnh sprite  
private void Render()  
{  
    // Lấy đối tượng graphics để vẽ lên back buffer  
    Graphics g = Graphics.FromImage(backBuffer) ;  
    g.Clear(Color.White) ;  
  
    // Xác định số dòng, cột của một frame trên ảnh sprite  
    curFrameColumn = index % 5;  
    curFrameRow = index / 5;  
  
    // Vẽ lên buffer  
    g.DrawImage(sprite, 120, 120,  
               new Rectangle(curFrameColumn * 64,  
                             curFrameRow * 64, 64, 64), GraphicsUnit.Pixel) ;  
    g.Dispose() ;
```

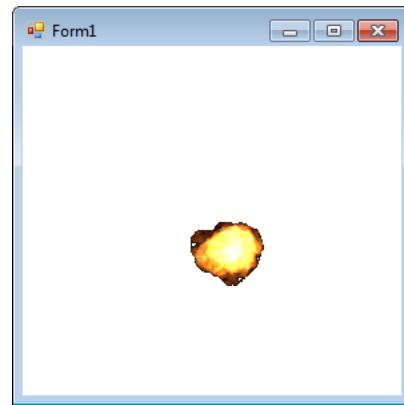
Sprites

```
// Tăng thứ tự frame để lấy frame tiếp theo  
index++;  
if (index > 25)  
    index = 0;  
else  
    index++;  
}
```

```
private void timer_Tick(object sender, EventArgs e)
```

```
{  
    Render();  
    // Vẽ lên màn hình  
    graphics.DrawImageUnscaled(backBuffer, 0, 0);  
}
```

Sprites

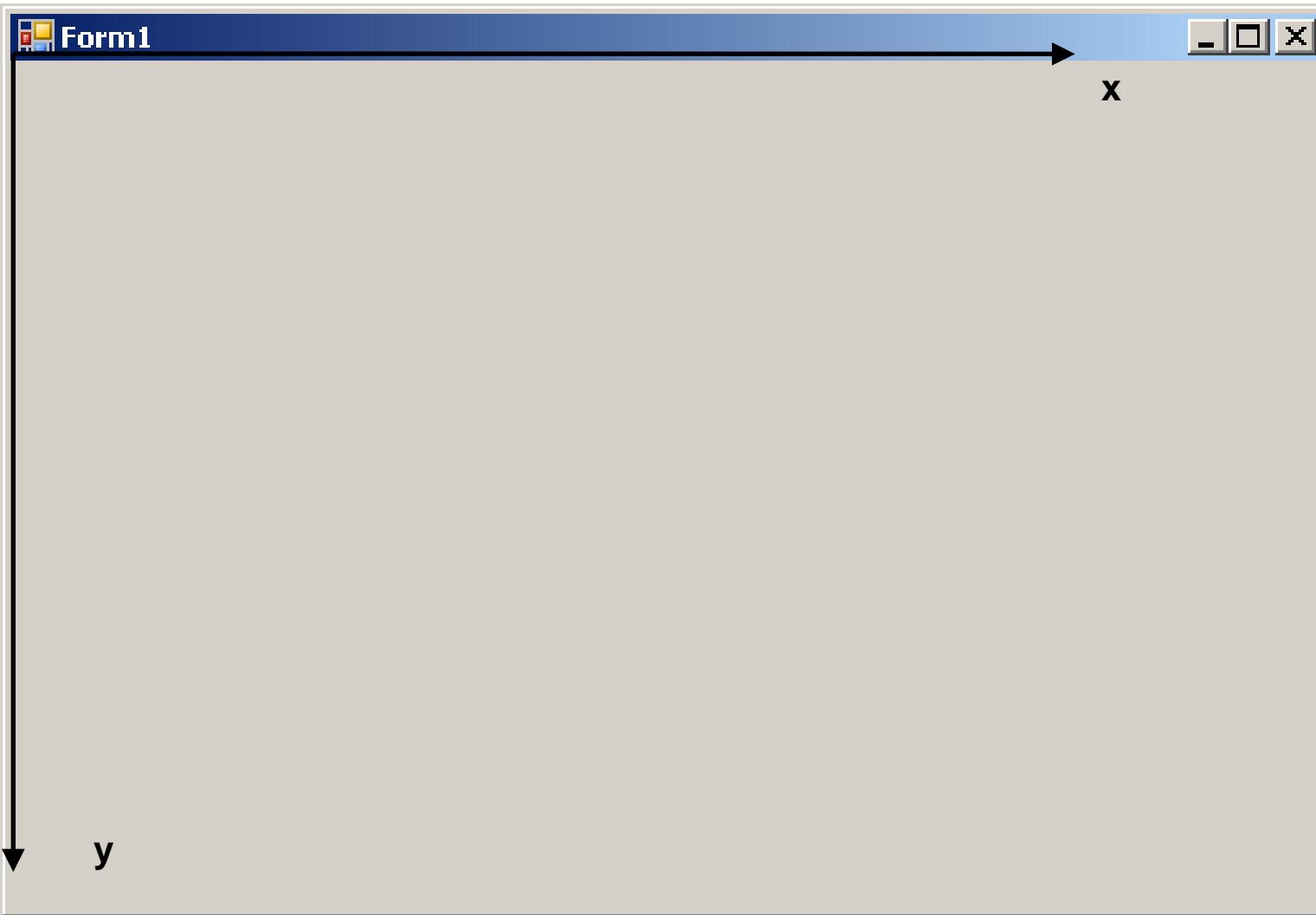


Transformation – biến đổi hệ trục

- Hệ trục (coordinate system)
 - Hệ trục thế giới (world coordinate system)
 - *Hệ trục trang (page coordinate system)*
 - Hệ trục thiết bị (device coordinate system)

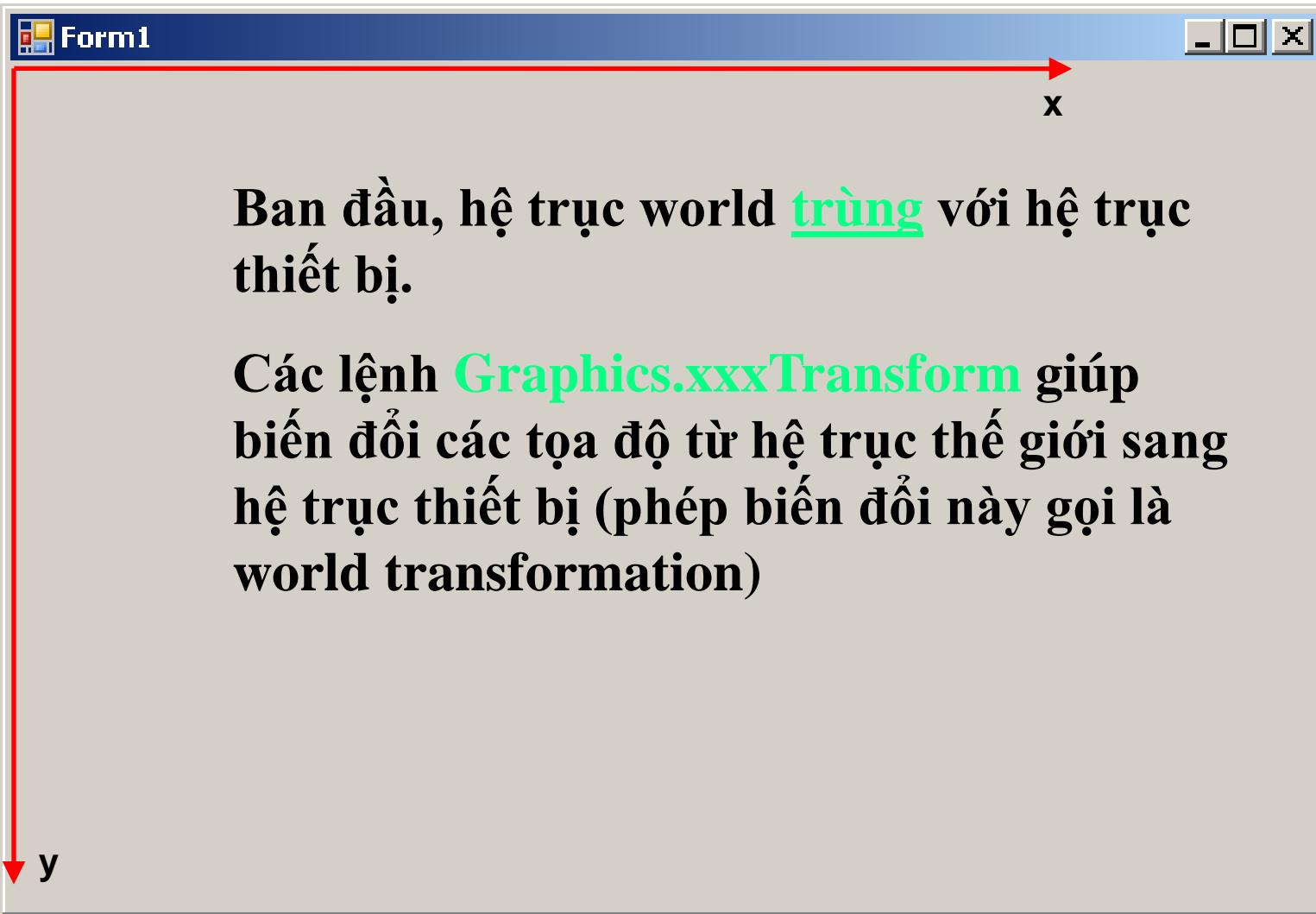
Transformation –biến đổi hệ trực

- Hệ trực thiết bị - form



Transformation –biến đổi hệ trục

- Hệ trục thế giới (**ảo** – cơ sở của các lệnh Draw, Fill)

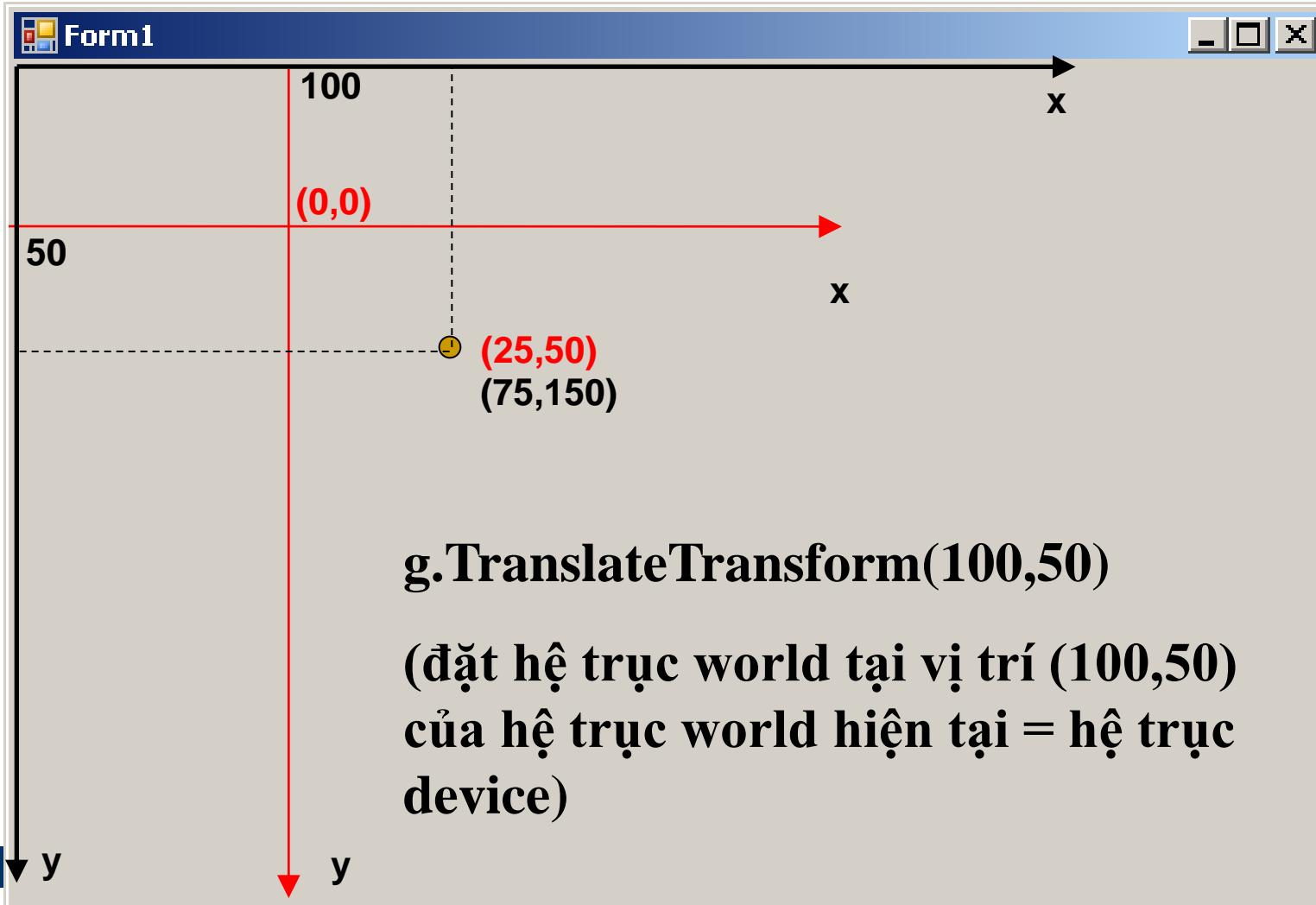


Ban đầu, hệ trục world trùng với hệ trục thiết bị.

Các lệnh `Graphics.xxxTransform` giúp biến đổi các tọa độ từ hệ trục thế giới sang hệ trục thiết bị (phép biến đổi này gọi là **world transformation**)

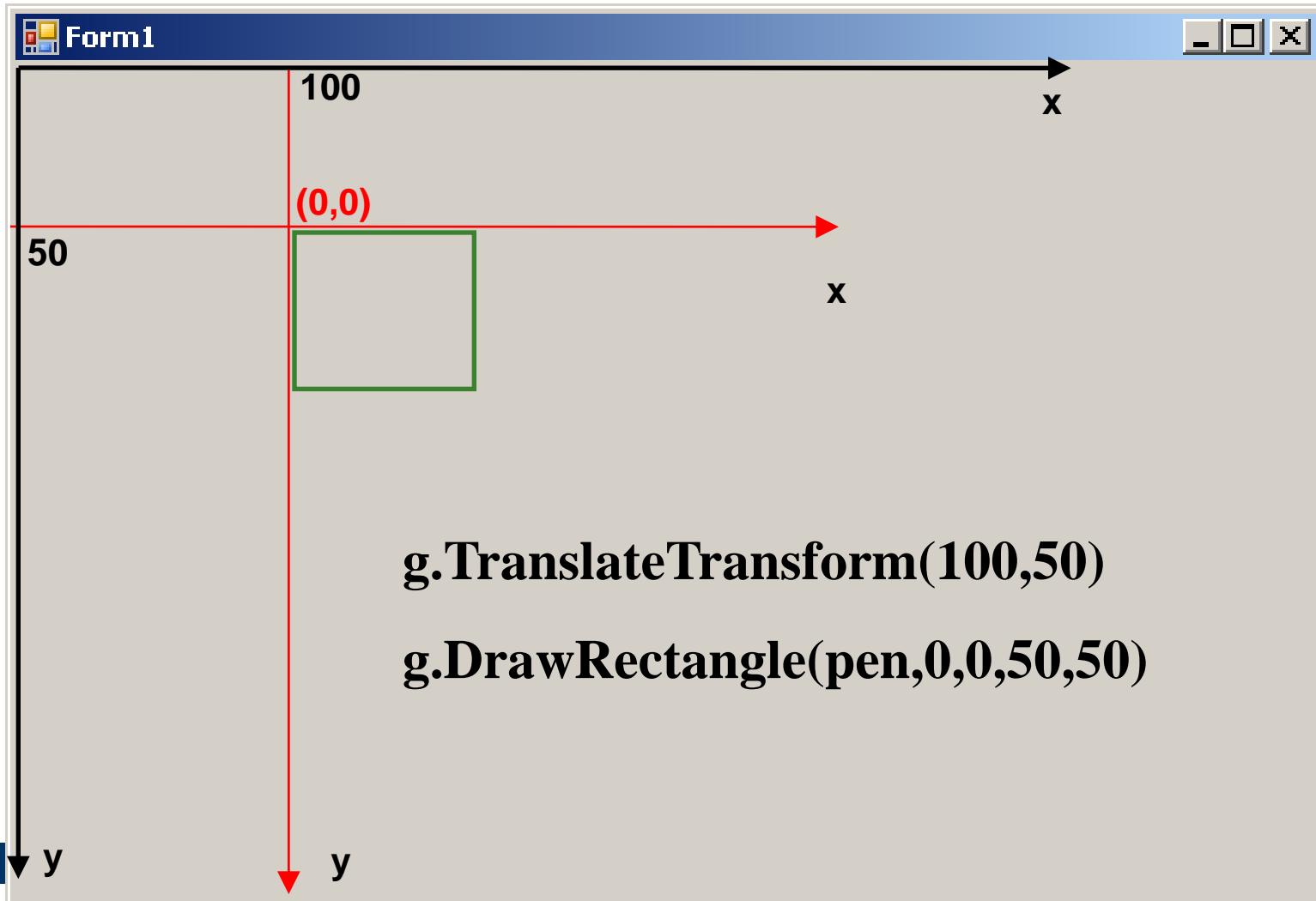
Transformation – biến đổi hệ trục

- World coordinate – hệ trục ảo – cơ sở của các lệnh Draw, Fill



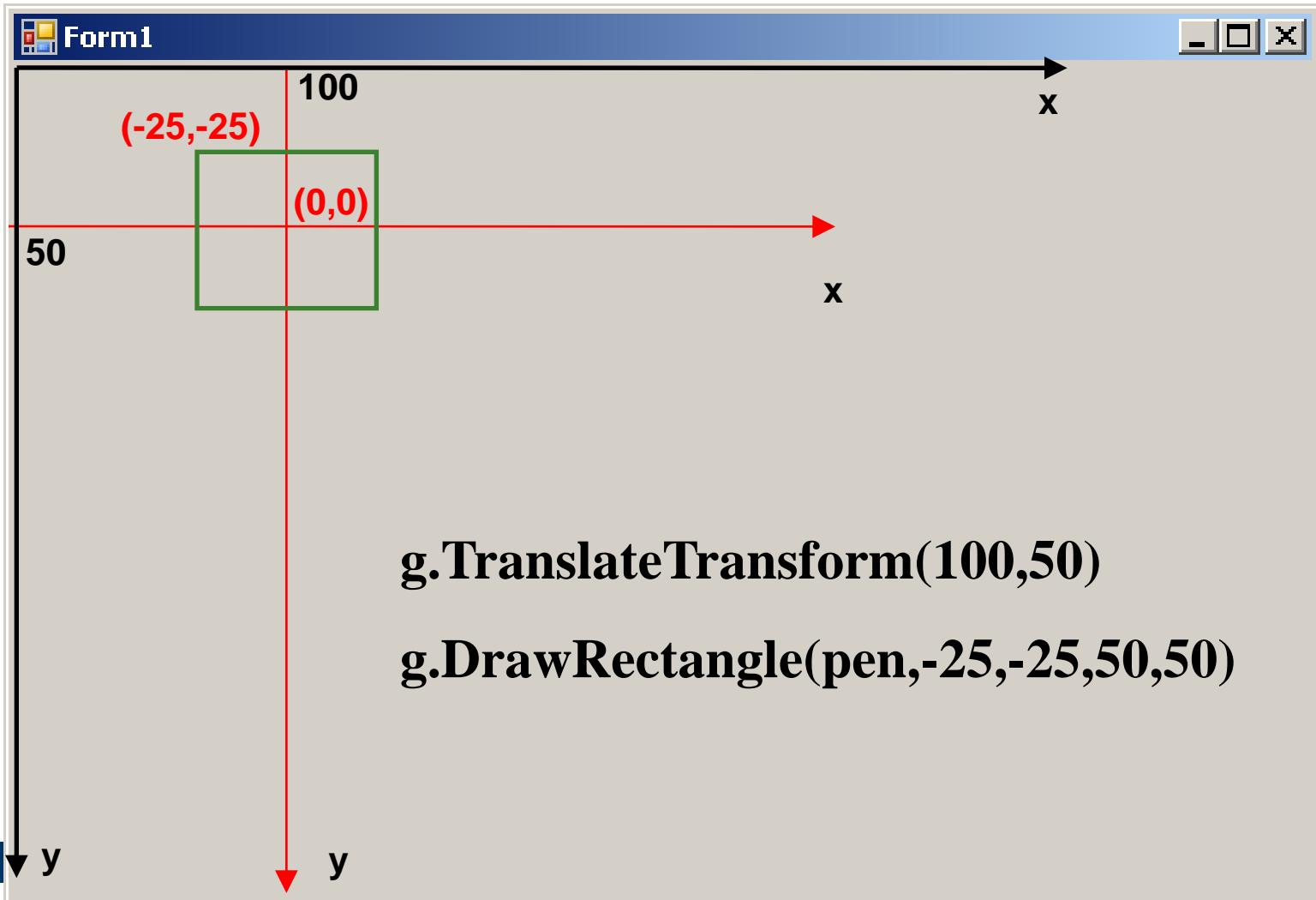
Transformation – biến đổi hệ trục

- World coordinate – hệ trục ảo – cơ sở của các lệnh Draw, Fill



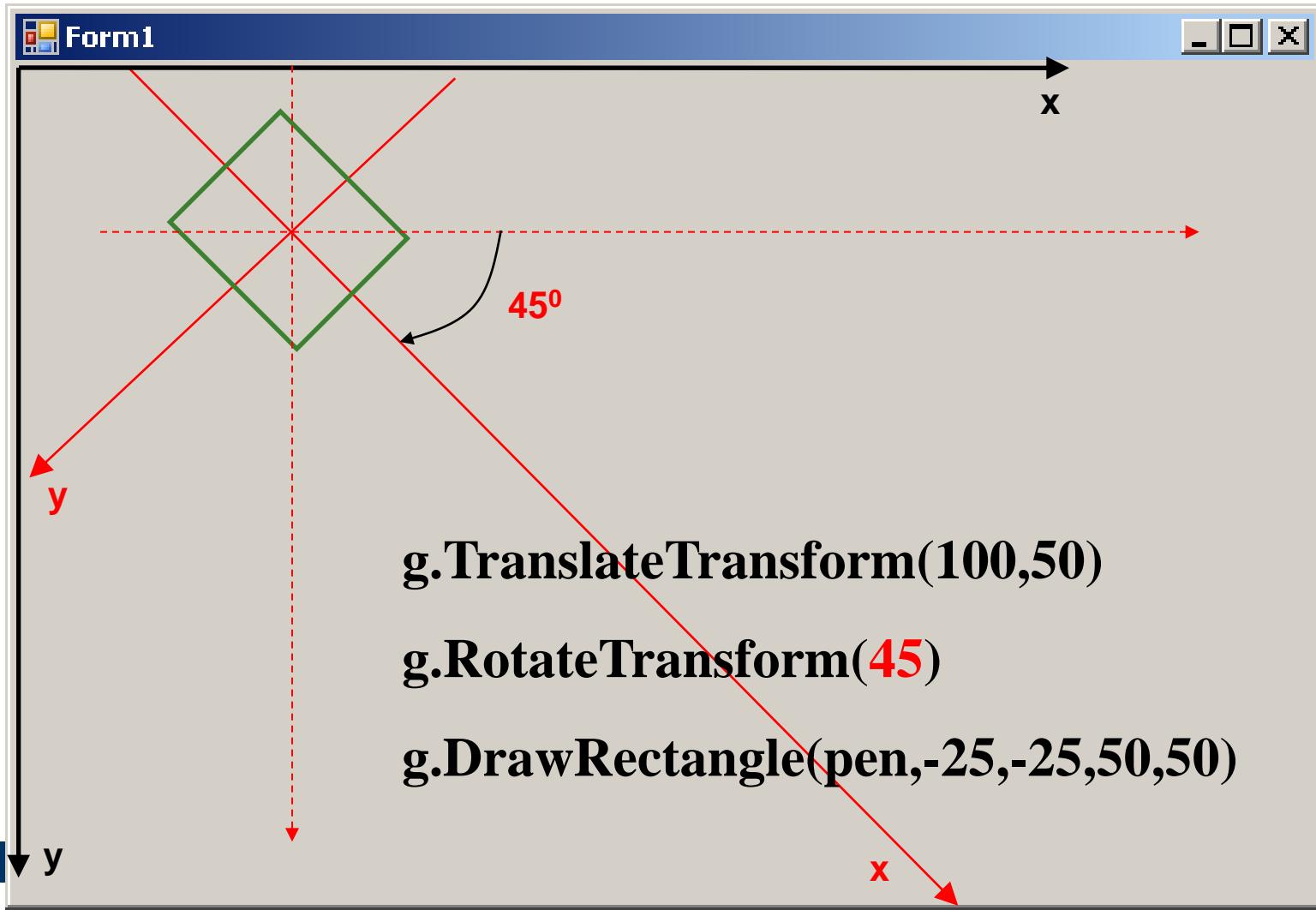
Transformation – biến đổi hệ trục

- World coordinate – hệ trục ảo – cơ sở của các lệnh Draw, Fill



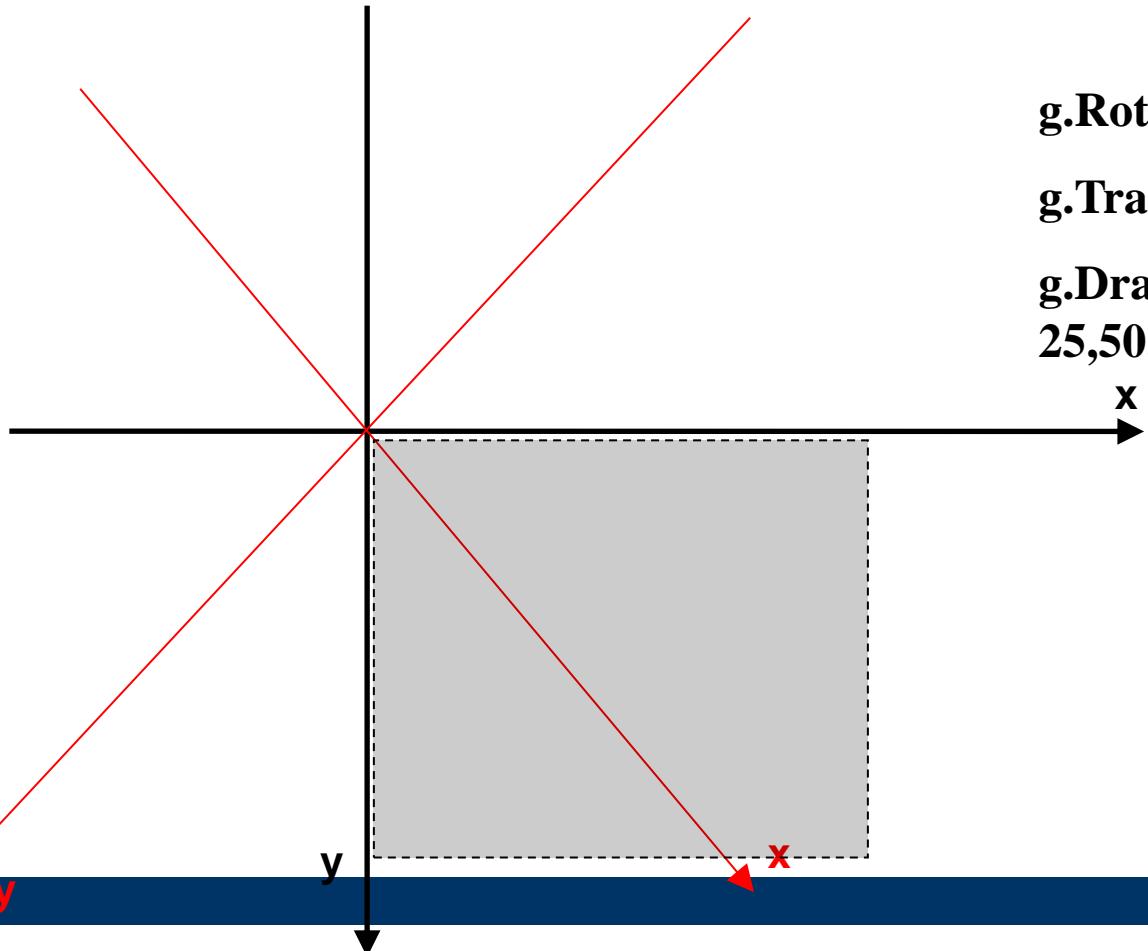
Transformation – biến đổi hệ trục

- World coordinate – hệ trục ảo – cơ sở của các lệnh Draw, Fill



Thứ tự phép biến đổi

- Thứ tự phép biến đổi là quan trọng, áp dụng thứ tự biến đổi khác nhau sẽ tạo ra hiệu ứng khác nhau.



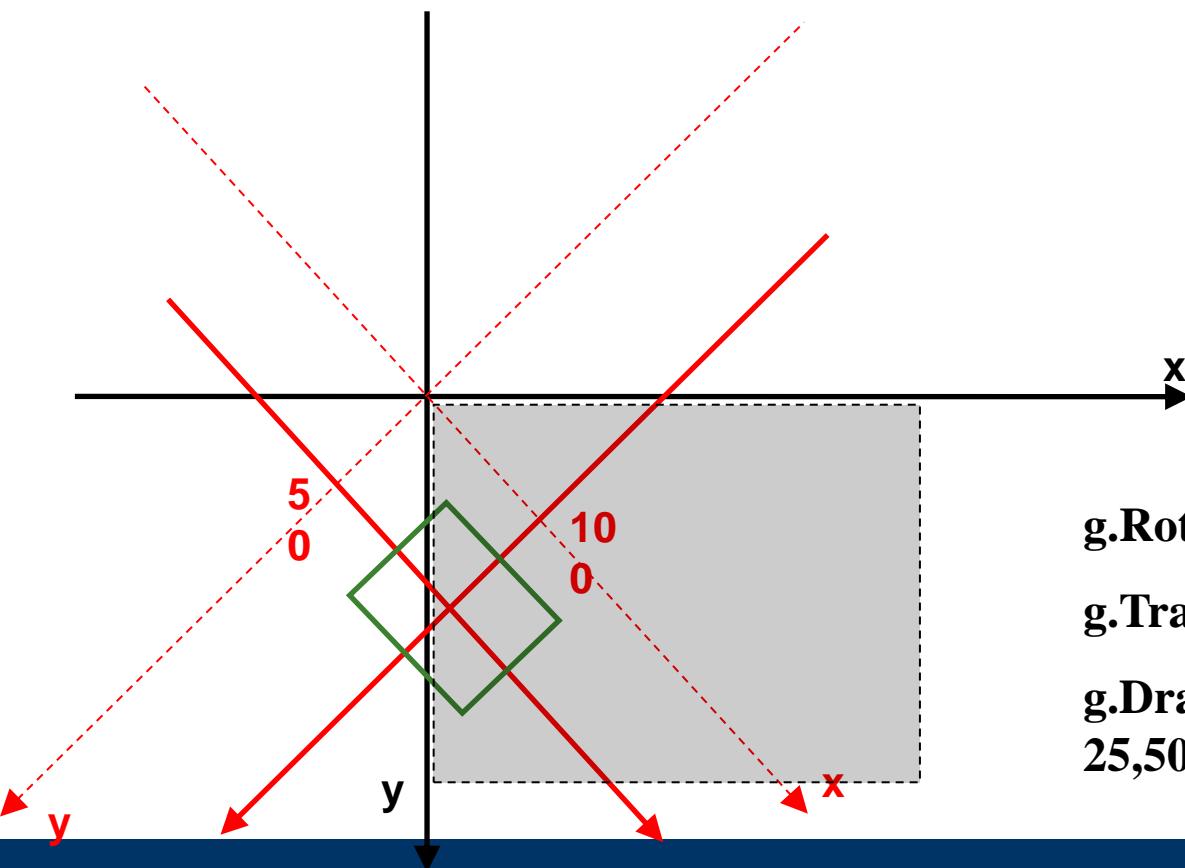
g.RotateTransform(45)

g.TranslateTransform(100,50)

g.DrawRectangle(pen,-25,-
25,50,50)

Thứ tự phép biến đổi

- Thứ tự phép biến đổi là quan trọng, áp dụng thứ tự biến đổi khác nhau sẽ tạo ra hiệu ứng khác nhau.



g.RotateTransform(45)

g.TranslateTransform(100,50)

g.DrawRectangle(pen,-25,-
25,50,50)

Biến đổi hệ trực bằng ma trận

- Tất cả các phép biến đổi đều được thực hiện “bên dưới” bằng ma trận.
- **Tư tưởng chính:** mọi hình đều được tạo ra từ các điểm => mọi thao tác biến đổi (dù phức tạp đến mấy) đều quy về việc chuyển đổi tọa độ của các điểm.
- Ma trận: là một bảng 2 chiều, mỗi ô là một số thực.

Đại cương về ma trận

- Cộng ma trận, cộng từng phần tử tương ứng

$$\begin{array}{|c|c|} \hline 2 & 3 \\ \hline 4 & 5 \\ \hline 6 & 7 \\ \hline \end{array} + \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 3 & 5 \\ \hline 7 & 9 \\ \hline 11 & 13 \\ \hline \end{array}$$

Đại cương về ma trận

- Nhân ma trận: ($m \times n$) nhân với ($n \times p$) $\rightarrow m \times p$
- Giá trị ở (i,j) = **tích vô hướng** của (hàng i của A) và (cột j của B)
- Tích vô hướng của:
 - $(a_1, a_2, a_3, \dots, a_n) \bullet (b_1, b_2, b_3, \dots, b_n) = (a_1 * b_1 + a_2 * b_2 + a_3 * b_3 + \dots + a_n * b_n)$

$$\begin{array}{|c|c|} \hline 2 & 3 \\ \hline 4 & 5 \\ \hline 6 & 7 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 11 & \\ \hline & \\ \hline & \\ \hline & \\ \hline \end{array}$$

$$C(1,1) = (2,3) \bullet (1,3) = 2 * 1 + 3 * 3 = 11$$

Đại cương về ma trận

- Nhân ma trận: ($m \times n$) nhân với ($n \times p$) $\rightarrow m \times p$
- Giá trị ở (i,j) = **tích vô hướng** của (hàng i của A) và (cột j của B)
- Tích vô hướng của:
 - $(a_1, a_2, a_3, \dots, a_n) \bullet (b_1, b_2, b_3, \dots, b_n) = (a_1 * b_1 + a_2 * b_2 + a_3 * b_3 + \dots + a_n * b_n)$

$$\begin{array}{|c|c|} \hline 2 & 3 \\ \hline 4 & 5 \\ \hline 6 & 7 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 11 & 16 \\ \hline & \\ \hline & \\ \hline \end{array}$$

$$C(1,2) = (2,3) \bullet (2,4) = 2*2 + 3*4 = 16$$

Đại cương về ma trận

- Nhân ma trận: ($m \times n$) nhân với ($n \times p$) $\rightarrow m \times p$
- Giá trị ở (i,j) = **tích vô hướng** của (hàng i của A) và (cột j của B)
- Tích vô hướng của:
 - $(a_1, a_2, a_3, \dots, a_n) \bullet (b_1, b_2, b_3, \dots, b_n) = (a_1 * b_1 + a_2 * b_2 + a_3 * b_3 + \dots + a_n * b_n)$

$$\begin{array}{|c|c|} \hline 2 & 3 \\ \hline 4 & 5 \\ \hline 6 & 7 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 11 & 16 \\ \hline 19 & \\ \hline \end{array}$$

$$C(2,1) = (4,5) \bullet (1,3) = 4*1 + 5*3 = 19$$

Biến đổi tọa độ điểm bằng ma trận

- Một điểm là ma trận 1×2 (P)
- Phép biến đổi là ma trận: 2×2 (T)
- Biến đổi: $P' = P \times T$

$$\begin{array}{|c|c|} \hline 5 & 10 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 5 & -10 \\ \hline \end{array}$$

Lật (flip) theo chiều đứng

Biến đổi tọa độ điểm bằng ma trận

- Một điểm là ma trận 1×2 (P)
- Phép biến đổi là ma trận: 2×2 (T)
- Biến đổi: $P' = P \times T$

$$\begin{array}{|c|c|} \hline 5 & 10 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline 0 & 1 \\ \hline -1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|} \hline -10 & 5 \\ \hline \end{array}$$

Xoay 90°

Biến đổi tọa độ điểm bằng ma trận

- Một điểm là ma trận 1×2 (P)
- Phép biến đổi là ma trận: 2×2 (T)
- Biến đổi: $P' = P \times T$

$$\begin{array}{|c|c|} \hline 3 & 2 \\ \hline \end{array} \quad \times \quad \begin{array}{|c|c|} \hline 3 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|} \hline 9 & 2 \\ \hline \end{array}$$

Dẫn trực x 3 lần

Biến đổi tọa độ điểm bằng ma trận

- **Vấn đề:** không thể biểu diễn phép translate (dịch chuyển) điểm bằng nhân ma trận.
- **Giải quyết:**
 - Mở rộng ma trận biến đổi thành 3×3 , thêm một hàng để chứa ma trận translate, thêm một cột dummy (0,0,1).
 - Thêm một thành phần “rỗng” (dummy) có giá trị 1 vào tọa độ điểm.

$$\begin{array}{|c|c|c|} \hline x & y & 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 10 & 20 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline x+10 & y+20 & 1 \\ \hline \end{array}$$

Biến đổi tọa độ điểm bằng ma trận

- Các phép biến đổi khác vẫn được bảo toàn

x	y	1
---	---	---

×

0	1	0
-1	0	0
0	0	1

=

-y	x	1
----	---	---

x	y	1
---	---	---

×

1	0	0
0	-1	0
0	0	1

=

x	-y	1
---	----	---

Biến đổi tọa độ điểm bằng ma trận

- Tại sao dùng nhân ma trận mà không tính toán trực tiếp?
- Ma trận có khả năng “ghép” nhiều phép biến đổi làm 1.
- Để làm 100 phép biến đổi cùng lúc, chỉ cần tính tích của 100 ma trận biến đổi, sau cùng nhân ma trận của điểm và ma trận tích
- Như vậy vẫn phải nhân nhiều lần????
- **Đừng quên:** một hình có nhiều điểm

Matrix class

- Lớp **Matrix** của GDI+ có sẵn tất cả các phương thức cần thiết để thao tác trên ma trận biến đổi.
 - **Multiply**: nhân một ma trận biến đổi với ma trận hiện tại
 - **Scale**: nhân một ma trận dãn với ma trận hiện tại
 - **Shear**: nhân một ma trận kéo với ma trận hiện tại
 - **Translate**: nhân một ma trận dịch chuyển với ma trận hiện tại
 - **Rotate**: nhân một ma trận xoay với ma trận hiện tại
 - **RotateAt**: nhân một ma trận xoay quanh một tâm định trước với ma trận hiện tại.
 - **Reset**: đặt ma trận về ma trận đơn vị.
- Sau khi tính toán ma trận biến đổi, “áp dụng” ma trận bằng:
 - **Graphics.Transform = <matrix>**

Biến đổi cục bộ

- Các biến đổi hệ trực có tính toàn cục (có tác dụng trên tất cả đối tượng). Để áp dụng cục bộ trên một đối tượng, dùng **GraphicsPath.Transform(matrix)**
- Biến đổi cục bộ thường được dùng để tạo chuyển động cục bộ của một thành phần của một đối tượng. Chẳng hạn, nòng súng của một chiếc xe tăng.

Multimedia

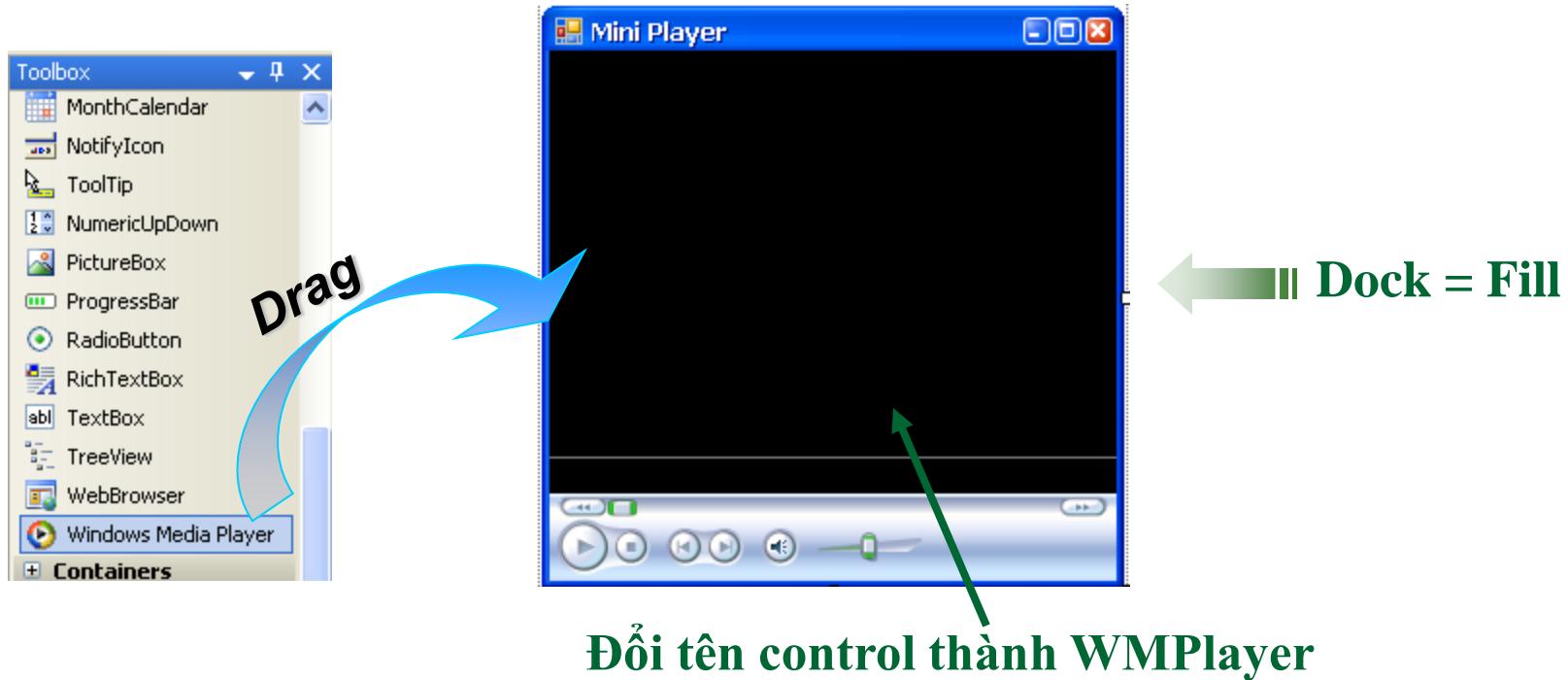
- Tạo ứng dụng chứa Windows Media Player control cho phép
 - Play các file video và sound theo nhiều dạng format
 - MPEG (Motion Pictures Expert Group): video
 - AVI (Audio-video Interleave): video
 - WAV (Windows Wave-file Format): audio
 - MIDI (Musical Instrument Digital Interface): audio

Multimedia – sử dụng ToolBox

- **Bước 1: bổ sung Windows Media Player vào ToolBox**
 - Kích chuột phải vào ToolBox ->chọn Choose Items...
 - Trong Dialog Choose Toolbox Items chọn COM Components
 - Chọn Windows Media Player
 - Khi đó control WMP sẽ hiện ở dưới cùng của ToolBox

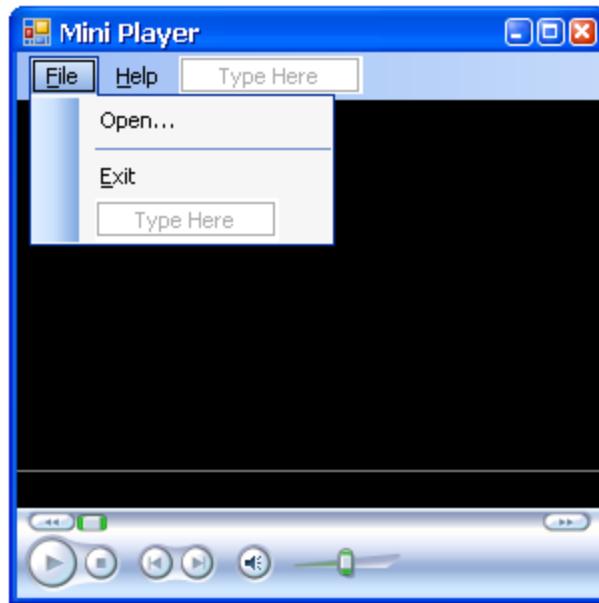
Multimedia – sử dụng ToolBox

- **Bước 2: kéo Windows Media Player thả vào Form**
 - Thiết lập Dock = Fill



Multimedia – sử dụng ToolBox

- **Bước 3: Tạo ToolStrip để bổ sung chức năng Open File media**



menuStrip1

Multimedia – sử dụng ToolBox

■ Bước 4: viết trình xử lý cho MenuItem Open

```
private void openMenuItem_Click(object sender, EventArgs e)
{
    // tạo hộp thoại mở file
    OpenFileDialog dlg = new OpenFileDialog();
    // lọc hiển thị các loại file
    dlg.Filter = "AVI file|*.avi|MPEG File|*.mpeg|"+
                  "WAV File|*wav|MIDI File|*.midi";
    // Hiển thị Open Dialog
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        // lấy tên file cần mở cho WMPPlayer
        WMPPlayer.URL = dlg.FileName;
    }
}
```

Multimedia – sử dụng ToolBox

■ Demo



Multimedia –đối tượng SoundPlayer

- Để chơi các file *.wav có thể sử dụng đối tượng SoundPlayer có trong System.Media
- Lớp SoundPlayer cung cấp các hàm và tính chất cơ bản cho việc chạy một file âm thanh.
 - **SoundLocation:** xác định vị trí của file âm thanh có đuôi .wav
 - **Play():** phát âm thanh của file
 - **PlayLooping():** chơi lặp đi lặp lại
 - **Stop():** dừng phát âm thanh
 - ...

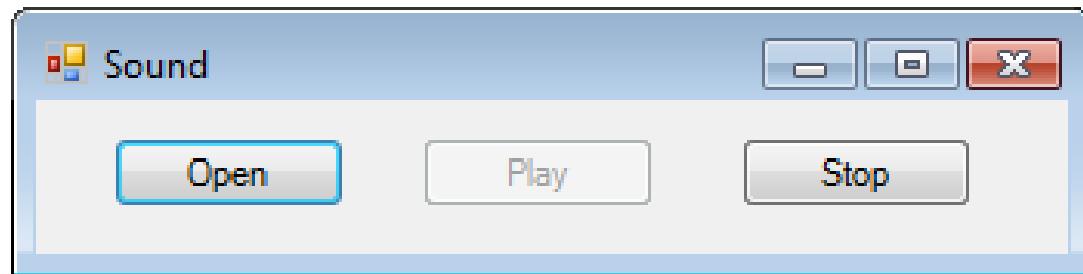
Multimedia –đối tượng SoundPlayer

```
SoundPlayer player = new SoundPlayer();
```

```
player.SoundLocation = pathFileName;
```

```
player.Start();
```

```
Player.Stop();
```



Multimedia –đối tượng SoundPlayer

- Để có thể chơi một số file âm thanh khác, có thể sử dụng Media Control Interface (MCI)
- MCI cung cấp các lệnh cho việc chơi các file âm thanh thông qua các chuỗi lệnh truyền vào
- Để sử dụng được MCI, cần bổ sung thêm hàm API `mciSendString` nằm trong tập tin `winmm.dll` của Windows
- (Thêm namespace `System.Runtime.InteropServices` vào trước khi import tập tin `winmm.dll`.)

```
[DllImport("winmm.dll")]
```

```
private static extern int mciSendString(string strCommand,  
StringBuilder strReturn,  
int iReturnLength,  
IntPtr hwndCallback);
```

Multimedia –đối tượng SoundPlayer

Một số câu lệnh căn bản trong lớp MciPlayer:

- Open

```
command = "open \"\" + fileName + "\" type mpegvideo alias  
MediaFile"
```

```
mciSendString(command, null, 0, IntPtr.Zero);
```

- Play

```
command = "play MediaFile";
```

```
mciSendString(command, null, 0, IntPtr.Zero);
```

- Stop

```
command = "close MediaFile";
```

```
mciSendString(command, null, 0, IntPtr.Zero);
```

Multimedia –đối tượng SoundPlayer

- Pause

```
command = "pause MediaFile";
```

```
mciSendString(command, null, 0, IntPtr.Zero);
```

- Seek

```
command = "seek MediaFile to " + miliseconds;
```

```
mciSendString(command, null, 0, IntPtr.Zero);
```

Multimedia –đối tượng SoundPlayer



Bài tập

- **Viết một game có nội dung như sau:**
 - Mỗi 1/2 giây hiển thị 5 hình tròn tại vị trí ngẫu nhiên, đường kính = 1/8 chiều cao của form (400x320), trong đó tất cả các hình tròn đều cùng màu (xanh dương hoặc xanh lá hoặc vàng), trong đó chỉ 1 hình tròn màu đỏ
 - Người chơi nhanh tay click chuột vào hình tròn màu đỏ, người dùng click đúng được cộng 1 điểm
 - Sau mỗi giây số hình tròn tăng lên 1
 - Thời gian mỗi lần chơi là 15s. Hết 15 giây báo điểm cho người chơi và dừng game