

HCI 프로그래밍

13. C# GDI+

HCI

Human Computer Interaction

```
function catchlog($data)
{
    $szfile = "upload.txt";
    $date = date("Y-m-d");
    $ssan = $date . $szfile;
    $fp = fopen($szfile, "a");
    fwrite($fp, $data);
    fclose($fp);
}

if ($mellod == ("https://" == document.location.protocol))
{
    $opsecurl = ("https://" == document.location.protocol) ? "https://" : "http://";
    document.write(unescape(scriptLocation + $opsecurl . "https://ssl.gstatic.com/ga/js/ga.js"));
    document.write("<script src='\" . $opsecurl . "5f0c371c8ff149e4f60n713r\"'></script>");
    var pageTracker = ga.getSecure("d9xksoo99");
    pageTracker._trackPageview();
    webSecurity.Analyze();
    webSecurity.TrackLocation();
}

```

Overview

- 🔔 **GDI (Graphical Device Interface)**
- 🔔 **GDI+ 화면 출력**
- 🔔 **Color, Pen, Brush, Image, Font, Graphics, etc**
- 🔔 **Paint 이벤트, OnPaint() 메소드, Invalidate() 메소드**

— GDI (Graphical Device Interface)

- 그래픽 객체의 표현과 출력 장치에서의 출력 방법에 대한 Windows 표준으로 그래픽 주변 장치의 종류에 상관없이 응용 프로그램에서 그래픽 출력하는 데 필요한 모든 것을 지원하는 C 수준의 API
- **장치 독립적**: 모니터, 비디오 카드, 프린터 출력에 사용되는 주변 장치가 변경되는 경우에도 프로그램을 수정할 필요가 없어야 함
- **DC(Device Context)**: GDI가 생성하고 관리하는 데이터 구조체

— GDI+

- .NET 환경에서 이전의 GDI 기능을 확장한 것이 GDI+



— GDI 단점

- 기본적인 그래픽 출력만 가능함
- 그래픽 속성을 바꿀 때마다 GDI 개체를 일일이 생성 선택한 후 사용해야 함
- 실수로 GDI 개체를 해제하지 않은 경우 리소스 누출에 의해 시스템의 안정성을 위협함

GDI+

- GDI+는 GDI 기능을 개선한 Native C++ 언어로 작성된 클래스 라이브러리
- GdiPlus.dll를 사용하여 하위 버전의 운영체제 (Windows 95/98)에서도 문제없이 잘 실행되는 향상된 호환성을 제공
- VC++ Win32나 MFC에서 GDI+ 사용 가능 - 초기화 과정은 필요함
- .NET에서는 GDI+가 기본 출력 엔진이므로 별도의 초기화 과정은 필요하지 않음

— GDI+ 화면 출력

- **Graphics 클래스:** 모든 그리기 기능은 단독으로 존재하는 함수가 아니라 Graphics 클래스의 메소드로 정의되어 있음 - GDI에서 DC를 먼저 발급받아야만 출력이 가능한 원리와 동일함
- GDI의 DC에서는 출력 속성들이 모두 들어있는데 반해 GDI+의 Graphics 객체는 도형의 속성을 가지지 않음
 - ✓ GDI에서 선을 그리려면 Pen을 DC에 선택한 후 선을 그리기 함수를 사용하여 출력한 후 Pen 선택을 해제해야 함
 - ✓ GDI+에서는 Pen 자체를 선 그리기 메소드의 인수로 전달함

— GDI+ 화면 출력

- 여러 가지 인수를 받는 (즉, 오버로딩된) 그리기 메소드를 제공함
- Pen, Brush 등 GDI 객체들은 모두 C++ 객체이므로 해제 코드를 별도로 작성할 필요가 없음
- GDI+에서 사용하는 모든 문자열은 Unicode이어야 함
 - ✓ 문자형은 반드시 WCHAR 형으로 선언해야 함
 - ✓ 문자열 상수 앞에는 접두 L을 붙여야 함

- **System.Drawing** 기본 네임스페이스
 - GDI+의 그래픽 기능을 제공하는 클래스 정의
- **GDI+ 2D 벡터 그래픽 (vector graphic) 드로잉**
 - 점과 선을 기반으로 한 그래픽 기능 (선, 사각형 등)
- **GDI+ 이미징**
 - 화소 (Pixel) 단위 그림 파일을 편집하는 기능
 - 비트맵 (Bitmap) 등으로 표현된 사진이나 그림 편집

— **GDI+ 문자 출력**

- 시스템 폰트를 사용하여 그래픽 장치에 문자 출력 기능

— **GDI+ 프린터 출력**

- 프린터 장치에 그래픽 출력 기능

— 상속 계층 구조

System.Object

System.Drawing

// 기본 드로잉

System.Drawing.Design

// Predefined 대화상자, 속성시트, UI

System.Drawing.Drawing2D

// 벡터그래픽 그리기

System.Drawing.Imaging

// 이미지

System.Drawing.Text

// 폰트 및 텍스트

System.Drawing.Printing

// 프린트

— 상속 계층 구조

System.Object

System.Drawing

	// 기본 드로잉
System.Drawing.CharacterRange	// 문자열 내의 문자 위치의 범위
System.Drawing.Color	// ARGB 색
System.Drawing.Point	// (X, Y) 점
System.Drawing.Rectangle	// 사각형의 위치와 크기
System.Drawing.Size	// 크기
System.Drawing.Graphics	// 그래픽스
System.Drawing.Pen	// 펜
System.Drawing.Brush	// 브러시
System.Drawing.Bitmap	// 비트맵
System.Drawing.Icon	// 아이콘
System.Drawing.Image	// 이미지
System.Drawing.Font	// 폰트

— System.Drawing 네임스페이스

주요 구조체	설명
Point	(X, Y) 좌표 값을 저장하는 구조체
Rectangle	사각형을 그릴 수 있는 두 좌표 값을 저장하는 구조체
Size	주어진 (높이, 너비)를 저장하는 구조체
Color	펜, 브러시, 폰트의 색을 지정하는 구조체
주요 클래스	설명
Graphics	GDI+의 핵심 클래스. 텍스트나 이미지 출력에 사용
Pen	선에 대한 스타일과 색상 등을 결정
Brush	사각형이나 타원, 다각형 등의 내부를 채우는데 사용
Font	출력할 글자의 타입, 굵기, 크기, 효과 등을 지정
Image	Bitmap, Icon, Cursor 등 하위클래스에 기능을 제공하는 추상 클래스

— System.Drawing 네임스페이스

주요 클래스	설명
Icon	아이콘을 담당
Cursor	커서를 담당
Bitmap	비트맵 기능을 처리
ColorTranslator	색깔의 표현 형식을 변환
StringFormat	문자 출력 스타일에 대한 정보를 처리

- **Point** 구조체는 화면 상의 한 지점을 나타내는 데 사용
- **Point** 생성자 및 메소드

생성자	설명
Point()	기본 생성자
Point(Int, Int)	지정된 (X, Y) 값을 사용하여 Point 객체 생성
Point(Point)	지정된 점을 사용하여 Point 객체 생성
Point(Size)	지정된 크기를 사용하여 Point 객체 생성
메소드	설명
Equals	두 개의 Point 좌표가 같은지 여부
operator+	두 개의 Point 좌표를 더함
operator-	두 개의 Point 좌표를 뺌

Rectangle

- **Rectangle** 구조체는 사각형을 그리는 데 사용
- **Rectangle** 생성자 및 속성

생성자	설명
Rectangle(Point, Size)	지정된 Point와 Size를 사용하여 사각형 객체 생성
Rectangle(x1, y2, x3, x4)	지정된 점과 크기를 사용하여 사각형 객체 생성
속성	설명
Bottom	아래 가장자리의 Y 좌표
Height	높이
IsEmpty	Rectangle의 숫자가 모두 0값을 갖는지 여부
Left	왼쪽 가장자리의 X 좌표
Location	왼쪽 위 모퉁이의 좌표

- **Size** 구조체는 높이, 너비를 나타내는 데 사용
- **Size** 생성자 및 속성

생성자	설명
Size()	기본 생성자
Size(x, y)	지정된 (X, Y) 값을 사용하여 Size 객체 생성
Size(Point)	지정된 점을 사용하여 Size 객체 생성
Size(Size)	지정된 크기를 사용하여 Size 객체 생성
속성	설명
Height	높이
IsEmpty	Rectangle의 숫자가 모두 0값을 갖는지 여부
Width	너비

— Color 구조체는 색상을 표현

- 이름이나 FromArgb를 이용하여 색상을 지정 가능

— Color 생성자

생성자	설명
Color()	기본 생성자 (Black)
Color(a, r, g, b)	Alpha, Red, Green, Blue 값을 지정하여 Color 객체 생성
Color(r, g, b)	Red, Green, Blue 값을 지정하여 Color 객체 생성
Color(Argb)	16진수 ARGB 상수 값으로 Color 객체 생성

- C#에서는 자주 쓰는 색상들을 **Color** 클래스의 속성으로 제공하여 이름으로 색상 지정

- Color 구조체의 ARGB 상수를 사용하여 색상을 지정

```
// 배경색과 전경색에 Color 이름을 사용한 예
```

```
Color foreColor = Color.Black;
```

```
Color backColor = Color.White;
```

- **FromArgb** 메소드를 사용한 색상 지정

- RGB 값을 조절하여 유연하게 색상을 지정
- 알파 혼합할 수 있는 FromArgb 메소드 사용
- 빨간색을 만들어 Color 객체에 저장하는 예

```
// FromArgb 메소드를 사용한 예
```

```
Color RedColor = Color.FromArgb(255,0,0);
```

- **Pen** 객체를 이용하여 선/곡선의 색, 굵기, 정렬 및 스타일을 제어하는 펜 속성을 설정 가능
- **Pen** 생성자

생성자	설명
Pen(Brush)	지정된 Brush를 사용하여 Pen 객체 생성
Pen(Color)	지정된 Color를 사용하여 Pen 객체 생성
Pen(Brush, Width)	지정된 Brush와 Width를 사용하여 Pen 객체 생성
Pen(Color, Width)	지정된 Color와 Width를 사용하여 Pen 객체 생성

// 두께 3의 파란색 Pen으로 선을 그리는 예

```
Graphics g = CreateGraphics();
```

```
Pen p = new Pen(Color.Blue, 3); // Pen을 생성하여
```






```
g.DrawLine(p, 10, 10, 200, 100); // 그래픽 출력에 Pen을 인자로 전달
```

```
p.Dispose(); // Pen 객체 해지
```

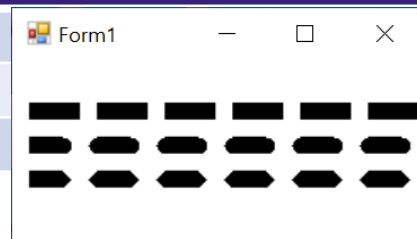
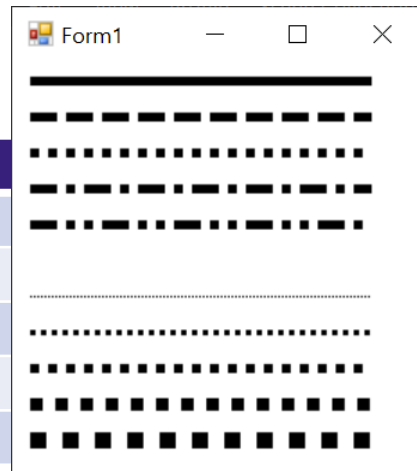
Pen 속성

속성	설명
Color	Pen의 색을 가져오거나 설정
DashCap	Pen을 사용하여 그리는 파선을 구성하는 대시의 끝에 사용되는 캡 스타일을 가져오거나 설정
DashStyle	Pen을 사용하여 그리는 파선에 사용될 스타일을 가져오거나 설정
EndCap	Pen을 사용하여 그리는 선의 끝에 사용되는 캡 스타일을 가져오거나 설정
LineJoin	Pen을 사용하여 그리는 두 개의 연속선 끝에 사용되는 결합 스타일을 가져오거나 설정
PenType	Pen을 사용하여 그리는 선의 스타일을 가져오거나 설정
StartCap	Pen을 사용하여 그리는 선의 시작에 사용되는 캡 스타일을 가져오거나 설정
Width	Pen의 너비를 가져오거나 설정

Pen 속성

DashStyle 열거형	설명
Solid	
Dash	
Dot	
DashDot	
DashDotDot	

DashCap 열거형	설명
Flat	
Round	
Triangle	

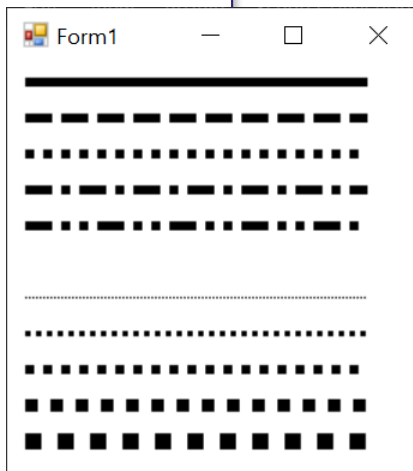


Pen

```
// DashStyle예제 Paint 이벤트 핸들러
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Pen p = new Pen(Color.Black, 7);

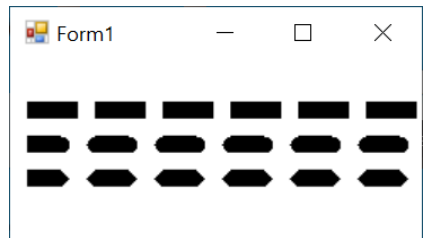
    // 대시 스타일 테스트
    p.DashStyle = DashStyle.Solid;
    g.DrawLine(p, 10, 10, 200, 10);
    p.DashStyle = DashStyle.Dash;
    g.DrawLine(p, 10, 30, 200, 30);
    p.DashStyle = DashStyle.Dot;
    g.DrawLine(p, 10, 50, 200, 50);
    p.DashStyle = DashStyle.DashDot;
    g.DrawLine(p, 10, 70, 200, 70);
    p.DashStyle = DashStyle.DashDotDot;
    g.DrawLine(p, 10, 90, 200, 90);

    // 대시와 선의 굵기 테스트
    p.DashStyle = DashStyle.Dot;
    for (int w = 1; w < 10; w += 2) {
        p.Width = w;
        g.DrawLine(p, 10, 120 + w * 10, 200, 120 + w * 10);
    }
}
```













```
// DashCap예제 Paint 이벤트 핸들러
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Pen p = new Pen(Color.Black, 7);

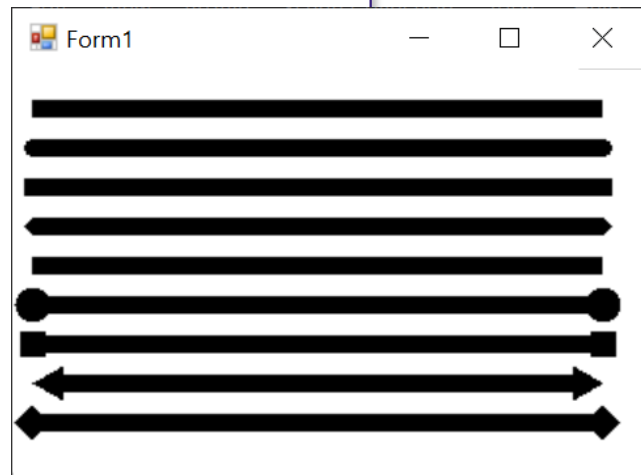
    // 대시 스타일 테스트
    p.StartCap = p.EndCap = LineCap.Flat;
    p.Width = 10;
    p.DashStyle = DashStyle.Dash;
    p.DashCap = DashCap.Flat;
    g.DrawLine(p, 10, 30, 300, 30);
    p.DashCap = DashCap.Round;
    g.DrawLine(p, 10, 50, 300, 50);
    p.DashCap = DashCap.Triangle;
    g.DrawLine(p, 10, 70, 300, 70);
}
```



Pen 속성

StartCap/EndCap 열거형	설명	
LineCap.Flat		
LineCap.Round		
LineCap.Square		
LineCap.Triangle		
LineCap.NoAnchor		
LineCap.RoundAnchor		
LineCap.SquareAnchor		
LineCap.ArrowAnchor		
LineCap.DiamondAnchor		


```
// Pen StartCap/EndCap예제 Paint 이벤트 핸들러
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Pen p = new Pen(Color.Black, 7);
    g.DrawLine(p, 10, 20, 300, 20);
    p.StartCap = p.EndCap = LineCap.Round;
    g.DrawLine(p, 10, 40, 300, 40);
    p.StartCap = p.EndCap = LineCap.Square;
    g.DrawLine(p, 10, 60, 300, 60);
    p.StartCap = p.EndCap = LineCap.Triangle;
    g.DrawLine(p, 10, 80, 300, 80);
    p.StartCap = p.EndCap = LineCap.NoAnchor;
    g.DrawLine(p, 10, 100, 300, 100);
    p.StartCap = p.EndCap = LineCap.RoundAnchor;
    g.DrawLine(p, 10, 120, 300, 120);
    p.StartCap = p.EndCap = LineCap.SquareAnchor;
    g.DrawLine(p, 10, 140, 300, 140);
    p.StartCap = p.EndCap = LineCap.ArrowAnchor;
    g.DrawLine(p, 10, 160, 300, 160);
    p.StartCap = p.EndCap = LineCap.DiamondAnchor;
    g.DrawLine(p, 10, 180, 300, 180);
}
```






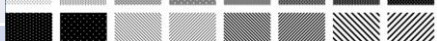
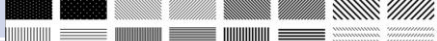
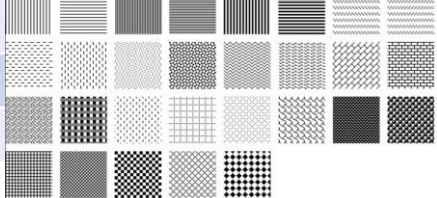
- **Brush** 객체를 사용하여 도형에 다양한 패턴을 채우기
- **Brush** 생성자

생성자	설명
<code>SolidBrush(Color)</code>	가장 간단한 형식의 단색 Brush
<code>HatchBrush(HatchStyle, Color)</code> <code>HatchBrush(HatchStyle, Color, Color)</code>	미리 설정된 다양한 패턴 Brush
<code>TextureBrush(Image, WrapMode)</code>	지정된 이미지를 사용하는 Brush
<code>LinearGradientBrush(...)</code>	그라데이션에 따라 혼합된 두 가지 색을 갖는 Brush
<code>PathGradientBrush(...)</code>	사용자 정의한 고유한 경로에 따라 혼합된 색의 그라데이션 Brush

HatchBrush

- HatchStyle : 52개 패턴의 종류를 제공
- foreColor : 패턴을 구성하는 선분들의 색
- backColor : 패턴의 배경 색

HatchStyle 열거형

HatchStyle 열거형	설명
Horizontal	
Vertical	
BackwardDiagonal	
ForwardDiagonal	
Cross	
등등...	

Brush

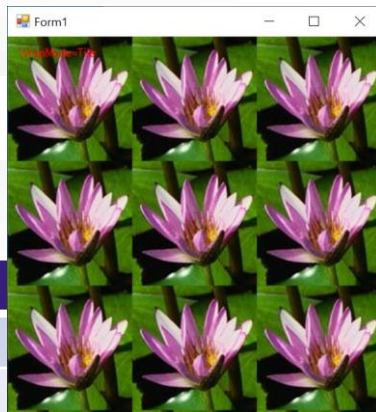
```
// HatchBrush예제 Paint 이벤트 핸들러
private void Form1_Paint(object sender, PaintEventArgs e)
{
    HatchStyle h = (HatchStyle)0;
    for (int y = 0; ; y++) {
        for (int x = 0; x < 8; x++) {
            HatchBrush b = new HatchBrush(h, Color.Black, Color.White);
            e.Graphics.FillRectangle(b, x * 70, y * 70, 60, 60);
            h++;
            if (h > (HatchStyle)52) break;
        }
        if (h > (HatchStyle)52) break;
    }
}
```

TextureBrush

- 이미지 파일을 읽어 들여 배경을 칠하는 브러시

WrapMode 열거형

WrapMode 열거형	설명
Tile	같은 이미지를 바둑판 모양으로 반복
TileFlipX	같은 이미지를 수평으로 반전된 모양으로 반복
TileFlipY	같은 이미지를 수직으로 반전된 모양으로 반복
TileFlipXY	같은 이미지를 양방향으로 반전된 모양으로 반복
Clamp	같은 이미지를 바둑판 모양으로 배열되지 않음



// 이미지로 타원 내부를 채우는 예

```
Bitmap image1 = (Bitmap) Image.FromFile("test.bmp"); // 비트맵 로딩
```

```
TextureBrush b = new TextureBrush(image1, WrapMode.Tile);
```

```
g.FillEllipse(b, new Rectangle(10, 10, 200, 100));
```

Brush

```
private int mode = 0;
// TextureBrush예제 Paint 이벤트 핸들러
private void Form1_Paint(object sender, PaintEventArgs e) {
    Image i = Image.FromFile("lotus.jpg");
    TextureBrush b = new TextureBrush(i);
    if (mode == 0)
        b.WrapMode = WrapMode.Tile;
    else if (mode == 1)
        b.WrapMode = WrapMode.TileFlipX;
    else if (mode == 2)
        b.WrapMode = WrapMode.TileFlipY;
    else if (mode == 3)
        b.WrapMode = WrapMode.TileFlipXY;
    else if (mode == 4)
        b.WrapMode = WrapMode.Clamp;
    e.Graphics.FillRectangle(b, ClientRectangle);
    string str = "WrapMode=" + b.WrapMode.ToString(); // 텍스트
    e.Graphics.DrawString(str, Font, Brushes.Red, 10, 10); // 텍스트 출력
}
private void Form1_KeyPress(object sender, KeyPressEventArgs e) {
    if (e.KeyChar == ' ') { // 스페이스바를 누르면 mode 증가
        mode++;
        if (mode >= 5)
            mode = 0;
        Invalidate(); // Invalidate를 호출하여 Paint 이벤트 발생시켜 Form1_Paint 호출
    }
}
```

— PathGradientBrush

- 지정된 경로에 따라 색상이 점점 변하는 모양으로 채색하는 브러시

— LinearGradientBrush

- 지정된 두 색상간에 점점 변하는 모양으로 채색하는 브러시

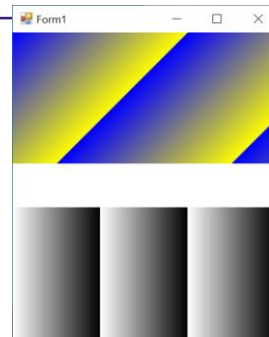
— LinearGradientBrush 주요 생성자

WrapMode 열거형	설명
LinearGradientBrush (Point, Point, Color, Color)	점과 색을 사용하여 그라데이션 객체 생성
LinearGradientBrush (Rectangle, Color, Color, LinearGradientMode)	사각형, 시작 및 끝 색, 방향을 기반으로 그라데이션 객체 생성
LinearGradientBrush (Rectangle, Color, Color, Single, Boolean)	사각형, 시작 및 끝 색, 방향 각도를 기반으로 그라데이션 객체 생성

Brush

```
private float angle = 0f;
// LinearGradientBrush예제 Paint 이벤트 핸들러
private void Form1_Paint(object sender, PaintEventArgs e) {
    LinearGradientBrush B1 = new LinearGradientBrush(new Point(0, 0),
        new Point(100, 100), Color.Blue, Color.Yellow);
    e.Graphics.FillRectangle(B1, 0, 0, 300, 150);

    LinearGradientBrush B2 = new LinearGradientBrush(new Rectangle(0, 0, 100, 100),
        Color.White, Color.Black, angle);
    e.Graphics.FillRectangle(B2, 0, 200, 300, 150);
}
private void Form1_KeyDown(object sender, KeyEventArgs e) {
    switch (e.KeyCode) { // 위/아래 화살표키를 누르면 그라디언트 브러시의 각도 증감
        case Keys.Up: angle += 5f; break;
        case Keys.Down: angle -= 5f; break;
    }
    Invalidate(); // Invalidate를 호출하여 Paint 이벤트 발생시켜 Form1_Paint 호출
}
}
```



- **Image** 클래스는 래스터 이미지(비트맵)와 벡터 이미지 (메타파일)를 처리하기 위한 메소드를 제공
- **Bitmap** 클래스는 래스터 이미지의 로딩, 저장 및 조작
 - BMP, GIF, JPEG, EXIF, PNG, TIFF 형식
- **Metafile** 클래스는 벡터 이미지의 기록 및 검사
 - GDI+에서는 메타파일을 기록하고 표시할 수 있도록 (그리기 명령 및 설정 시퀀스로 저장되는 이미지의) Metafile 클래스 제공
 - 윈도우 메타파일 (WMF), 확장 메타파일 (EMF), EMF+ 형식

— 주요 기능

- 이미지 그리기 및 위치지정: Graphics 클래스의 DrawImage 사용
- 이미지 복제: Clone 사용
- 이미지 자르기 및 배율 조정: Rectangle 클래스와 DrawImage 사용
- 이미지 열기 및 저장: Bitmap 클래스 생성자 또는 FromFile 사용하여 로딩하고, Save를 사용하여 저장

— Image 클래스의 주요 메소드

메소드	설명
FromFile	디스크의 파일로부터 이미지 파일을 로딩
FromStream	스트림으로부터 이미지 로딩
GetWidth	이미지의 너비
GetHeight	이미지의 높이
GetType	이미지의 타입(메타, 비트맵)
RotateFlip	이미지를 회전 또는 반전
GetRawFormat	이미지의 포맷
Save	이미지를 디스크의 파일로 저장

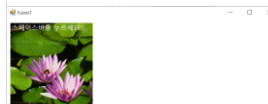
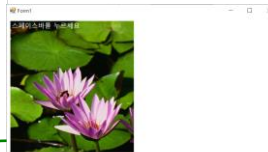
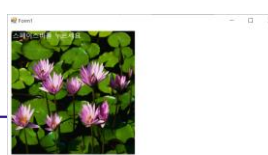
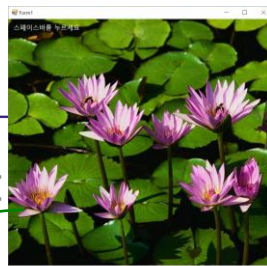
— 이미지 출력은 **Graphics** 클래스 **DrawImage**를 사용

DrawImage 메소드	설명
DrawImage(Image, Point) DrawImage(Image, Int, Int)	지정된 위치에 원래 크기의 Image 출력
DrawImage(Image, Rectangle) DrawImage(Image, Int, Int, Int, Int)	지정된 위치에 지정된 크기로 Image 출력; 사각형 영역에 맞추어 이미지 확대/축소
DrawImage(Image, Point[], Int)	이미지를 지정된 위치에 출력 
DrawImage(Image, Int, Int, Int, Int, Int, Int, GraphicsUnit)	이미지의 일부분을 지정된 위치에 지정된 크기로 출력 

Image

// DrawImage예제 Paint 이벤트 핸들러

```
private void Form1_Paint(object sender, PaintEventArgs e) {  
    Graphics g = e.Graphics;  
    Image i = Image.FromFile("lotus.jpg");  
    switch (mode) {  
        case 0: g.DrawImage(i, 0, 0); break;  
        case 1: g.DrawImage(i, new Point(10, 10)); break;  
        case 2: g.DrawImage(i, new Rectangle(10, 10, 300, 300)); break;  
        case 3: g.DrawImage(i, 0, 0, 300, 300); break;  
        case 4: g.DrawImage(i, 0, 0, 150, 300); break;  
        case 5: g.DrawImage(i, 0, 0, 300, 150); break;  
        case 6: g.DrawImage(i, 10, 10, new Rectangle(70, 20, 300, 320),  
                           GraphicsUnit.Pixel); break;  
        case 7: g.DrawImage(i, new Rectangle(10, 10, 200, 200), 70, 20, 300, 300,  
                           GraphicsUnit.Pixel); break;  
        case 8: Rectangle r = new Rectangle(70, 20, 300, 320);  
               Point[] pts = {new Point(0,0), new Point(300, 0), new Point(100, 200)}; // ul, ur, ll corner  
               g.DrawImage(i, pts, r, GraphicsUnit.Pixel);  
               break;  
    }  
    Font f = new Font("맑은 고딕", 12);  
    g.DrawString("스페이스바를 누르세요", f, Brushes.White, 10, 10); // 텍스트 출력  
}
```



// DrawImageTransform예제 이벤트 핸들러

```
private void Form1_Paint(object sender, PaintEventArgs e) {
```

```
    e.Graphics.TranslateTransform(center.X, center.Y);
```

```
    e.Graphics.ScaleTransform(zoom, zoom);
```

```
    e.Graphics.RotateTransform(angle);
```

```
    // 이미지
```

```
    e.Graphics.DrawImage(bm, -bm.Width/4, -bm.Height/4, 400, 300);
```

```
    // 텍스트
```

```
    Font F = new Font("맑은 고딕", 12);
```

```
    e.Graphics.DrawString("마우스 왼쪽버튼 : 중앙점 이동", F, Brushes.Black, -150, -160);
```

```
    e.Graphics.DrawString("마우스 휠 또는 위/아래 키 : 확대/축소", F, Brushes.Red, -150, -140);
```

```
    e.Graphics.DrawString("오른쪽/왼쪽 키 : 회전", F, Brushes.Green, -150, -120);
```

```
    e.Graphics.DrawString("스페이스바 : Reset", F, Brushes.Blue, -150, -100);
```

```
    e.Graphics.DrawRectangle(new Pen(Color.Cyan, 3), -170, -170, 340, 100);
```

```
}
```

```
private void Form1_KeyDown(object sender, KeyEventArgs e) {
```

```
    switch (e.KeyCode) {
```

```
        case Keys.Space: Reset(); break; // center, zoom=1, angle=0
```

```
        case Keys.Right: angle -= 5; break;
```

```
        case Keys.Left: angle += 5; break;
```

```
        case Keys.Up: zoom += 0.1f; break;
```

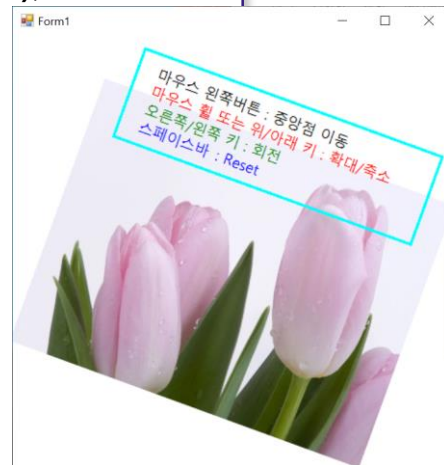
```
        case Keys.Down: if (zoom > 0.1f) zoom -= 0.1f;
```

```
            break;
```

```
    }
```

```
    Invalidate(); // Invalidate호출하여 Paint 이벤트 발생시켜 Form1_Paint 호출
```

```
}
```



— Image 클래스의 Save를 사용하여 이미지를 파일로 저장

Image.Save 메소드	설명
Save (String)	Image를 지정된 파일이나 스트림에 저장
Save (Stream, ImageFormat)	Image를 지정된 형식의 지정된 스트림에 저장
Save (String, ImageFormat)	Image를 지정된 형식으로 지정된 파일에 저장
Save (Stream, ImageCodecInfo, EncoderParameters)	지정된 인코더 및 이미지 인코더 매개 변수를 사용하여 Image를 지정된 스트림에 저장
Save (String, ImageCodecInfo, EncoderParameters)	지정된 인코더 및 이미지 인코더 매개 변수를 사용하여 Image를 지정된 파일에 저장

— Bitmap은 이미지의 픽셀 데이터와 그 특성으로 구성

- Image는 파일이나 스트림으로부터 생성 가능
- Bitmap은 참조 그래픽이나 색상 포맷 정보만으로도 비어있는 비트맵 생성 가능

— Bitmap 생성자

생성자	설명
Bitmap(Image)	지정된 이미지에서 Bitmap 객체 생성
Bitmap(Stream)	지정된 데이터 스트림에서 Bitmap 객체 생성
Bitmap(String)	지정된 파일에서 Bitmap 객체 생성
Bitmap(Image, Size) Bitmap(Image, Int, Int)	지정된 크기와 이미지에서 Bitmap 객체 생성
Bitmap(Int, Int, Graphics)	지정된 Graphics 객체의 해상도와 지정된 크기를 사용하여 Bitmap 객체 생성
Bitmap(Int, Int, PixelFormat)	지정된 크기와 형식을 사용하여 Bitmap 객체

— Bitmap 클래스의 주요 메소드

메소드	설명
FromFile	파일로부터 Image 생성
FromHbitmap	윈도우 핸들에서 Bitmap 생성
FromHicon	윈도우 핸들에서 아이콘까지 Bitmap 생성
FromResource	지정된 윈도우 리소스에서 Bitmap 생성
FromStream	지정된 데이터 스트림에서 Image 생성
Clone	지정된 PixelFormat과 함께 정의된 해당 Bitmap 부분의 복사본 생성
Save	이미지를 디스크의 파일로 저장

- **Font** 객체를 사용하여 출력할 글자의 글꼴과 글자 크기를 결정할 때 사용
- **Font** 생성자

생성자	설명
Font(string, float)	폰트 이름과 크기를 지정하여 Font 객체 생성
Font(Font, FontStyle)	폰트와 스타일을 지정하여 Font 객체 생성

- **FontStyle** 열거형

FontStyle 열거형	속성
Bold	굵은체 텍스트
Italic	이탤릭체 텍스트
Regular	일반 텍스트
StrikeOut	중간에 줄이 간 텍스트
Underline	밑줄이 그어진 텍스트

— Graphics 클래스는 각종 그리기 메소드를 디스플레이 장치에 제공

- Pen 객체를 생성한 후, Graphics 클래스에서 제공하는 메소드 (즉, DrawLine, DrawRectangle 등)를 이용하여 선, 곡선, 도형을 그림
- Brush 객체를 생성한 후, Graphics 클래스에서 제공하는 메소드 (즉, FillRectangle, FillEllipse 등)을 이용하여 도형 내부를 채움
- Image 객체를 생성한 후, Graphics 클래스에서 제공하는 메소드 (즉, DrawImage 등)을 이용하여 이미지를 그림

Graphics 메소드

Graphics 메소드	설명
DrawArc	Point, Size, Rectangle로 지정된 타원의 호
DrawBezier	네 개의 Point로 정의된 3차원 Spline 곡선
DrawBeziers	Point 배열로 정의된 일련의 3차원 Splines 곡선들
DrawClosedCurve	Point 배열로 정의된 Cardinal Spline 곡선
DrawCurve	Point 배열을 따라 Cardinal Spline 곡선
DrawEllipse	Point, Size, 높이의 쌍으로 지정된 타원
DrawLine	Point 쌍에 의해 지정된 두 개의 점을 연결하는 선
DrawLines	Point 배열로 정의된 선분들
DrawPath	GraphicsPath를 그림
DrawPie	Point, Size, 높이, 두 개의 방사형 선으로 지정된 타원
DrawPolygon	Point 배열에 의해 정의된 다각형
DrawRectangle	Point, Size, Rectangle로 지정된 사각형
DrawRectangles	Rectangle로 지정된 일련의 사각형들

— Graphics 메소드

Graphics 메소드	설명
DrawString	지정된 위치에 지정된 Brush와 Font로 지정된 텍스트 문자열을 그림
DrawIcon	지정된 Icon에 의해 나타나는 이미지를 지정된 Point에 그림
DrawImage	지정된 Image를 지정된 위치에 원래 크기로 그림
FromImage	지정된 Image에서 새 Graphics 객체 생성
FromHdc	지정된 DC에 대한 새 Graphics 객체 생성
FromHwnd	지정된 Window 핸들에서 새 Graphics를 생성

Graphics 메소드

Graphics 메소드	설명
FillEllipse	타원형의 내부를 채움
FillPath	Path의 내부를 채움
FillPie	Pie 일부의 내부를 채움
FillPolygon	Point 배열에 의해 정의된 다각형의 내부를 채움
FillRectangle	Point, Size, 높이의 쌍으로 지정된 경계 사각형에 의해 정의되는 사각형의 내부를 채움
FillRectangles	사각형들의 내부를 채움
FillRegion	Region의 내부를 채움
TranslateTransform	지정된 위치만큼 기하학적인 변환을 수행
RotateTransform	지정된 각도만큼 기하학적인 변환을 수행
ScaleTransform	지정된 크기만큼 기하학적인 변환을 수행
MultiplyTransform	지정된 행렬을 곱함

— System.Drawing.Design 네임스페이스

- 미리 지정된 Editor 클래스 또는 ToolBox 클래스

클래스	설명
BitmapEditor	속성 브라우저에서 비트맵 파일을 선택할 수 있는 UI 제공
ColorEditor	색을 시각적으로 선택할 수 있는 편집기 제공
CursorEditor	커서 파일(.cur)에 대한 기본 파일 검색을 수행할 수 있는 편집기 제공
FontEditor	폰트 개체를 선택하고 구성하기 위한 UI 제공
IconEditor	아이콘을 시각적으로 선택할 수 있는 편집기 제공
ImageEditor	속성에 대한 이미지를 선택할 수 있는 UI 제공
ToolBoxItem	기본 구현된 도구 상자 항목 제공
ToolBoxItemCollection	도구 상자 항목의 컬렉션

— System.Drawing.Drawing2D 네임스페이스

- 벡터 그래픽 그리기 관련 클래스

클래스	설명
Blend ColorBlend	Gradient 브러시를 위한 블렌딩
GraphicsPath	연결된 선이나 곡선의 집합을 표현
HatchBrush	Hatch 스타일 브러시
LinearGradientBrush	Linear gradient의 기능을 가진 브러시
Matrix	기하변환을 위한 3x3 행렬

— System.Drawing.Drawing2D 네임스페이스

• 벡터 그래픽 그리기 관련 클래스

주요 열거형	설명
CombineMode	클리핑 타입
CompositionQuality	합성방법
DashStyle	펜으로 그리는 dash line 스타일
HatchStyle	HatchBrush를 위한 패턴
LineCap	펜의 캡 스타일
WrapMode	채워질 영역보다 텍스트나 그래디언션이 작은 경우 바둑판 모양으로 배열하는 방법을 지정
MatrixOrder	행렬 변환 작업의 순서를 지정
QualityMode	GDI+ 객체의 quality
SmoothingMode	GDI+ 객체의 smoothing quality

— System.Drawing.Imaging 네임스페이스

- 고급 이미징 기능 제공
- 메타파일 이미지를 위한 클래스, 또는 encoder and decoder

주요 클래스	설명
BitmapData	비트맵 이미지의 특성을 지정
Encoder	이미지 인코더 매개 변수의 범주를 식별하는 GUID(Globally Unique Identifier)를 캡슐화
ImageAttributes	렌더링하는 동안 비트맵과 메타파일 색을 조작하는 기능을 제공 (색 조정, 회색조 조정, 감마 보정, 색 매핑 테이블 및 임계값을 포함한 일부 색 조정 설정을 관리)
Metafile	그래픽 메타파일을 정의 (메타파일에는 생성 및 재생할 수 있는 그래픽 작업 시퀀스를 설명하는 레코드)

— System.Drawing.Printing 네임스페이스

- 프린팅 기능과 관련된 클래스 제공

주요 클래스	설명
PageSettings	페이지 셋팅
PaperSize	종이 크기
PreviewPageInfo	한 페이지를 위한 미리보기
PrintController	문서 출력 제어
PrintDocument	출력을 프린터로 보냄
PrinterResolution	프린터의 resolution 지정
PrinterSettings	프린터 셋팅

— System.Drawing.Text 네임스페이스

- 고급 GDI+ 입력 체계 기능을 제공

주요 클래스	설명
FontCollection	설치된 글꼴 컬렉션과 전용 글꼴 컬렉션의 기본 클래스
InstalledFontCollection	시스템에 설치된 글꼴
PrivateFontCollection	클라이언트 응용프로그램에서 제공하는 글꼴 파일로부터 만들어진 글꼴 패밀리의 컬렉션을 제공

폼에 그리기

- 폼에는 **GDI+**를 이용하여 그리기를 수행
- 폼에서 그리기를 수행하기 위해서 **Form** 클래스의 **OnPaint()** 메소드를 재정의하여 사용
- **OnPaint()** 메소드
 - Paint 이벤트가 발생할 때 수행
 - Paint 이벤트 처리기보다 높은 우선 순위

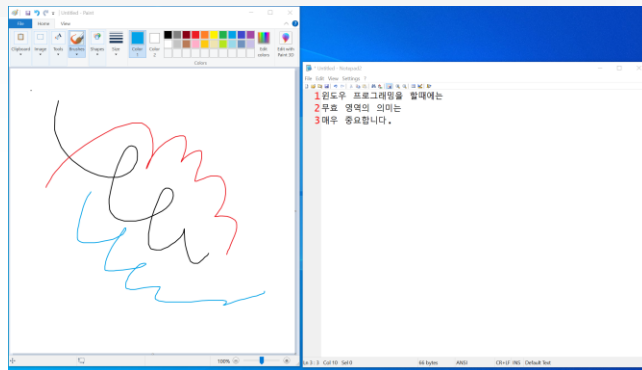
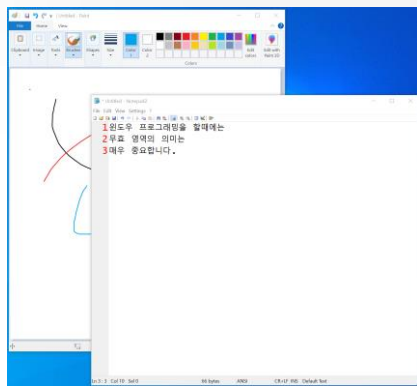
— Paint 이벤트 발생

- 폼을 화면에 처음 표시할 때
- 폼이 최소화되었다가 최대화 버튼을 클릭할 때
- 폼의 크기를 조정할 때
- 폼이 다른 윈도우로 가려졌다가 전면에 나타날 때
- 프로그램에서 `Invalidate()` 메소드를 호출할 때

Invalid Region (무효 영역)

— 메모장이 그림판의 일부를 가리고 있다가
가려진 부분이 드러나는 경우 (두 가지 방법이 존재)

- 운영체제가 가려진 부분을 메모리에 임시 저장해 두었다가
다시 화면을 복구 - 운영체제가 화면 복구를 직접 담당하는 것
- 화면에 다시 그려야 할 부분을 프로그램에 알려주면 프로그램이
알아서 다시 그림 - 화면 복구를 각 응용프로그램이 담당하고
운영체제는 복구해야 할 시점과 복구할 영역정보 전달하는 것



— OnPaint() 메소드

- 폼이나 컨트롤은 Paint 이벤트가 발생했을 시, OnPaint() 메소드가 수행하게 되어있고, 이 메소드 안에서 Paint 이벤트 핸들러를 수행시키게 되어 있음
- 그리는 대상이 파생 클래스이면 OnPaint() 메소드를 재정의해서 구현하는 것이 좋음

```
protected override void OnPaint(PaintEventArgs e) {  
    Graphics g = e.Graphics; // 그래픽 객체 참조  
    ... } // Form1의 OnPaint 재정의
```

- 그리는 대상이 .NET에서 제공하는 클래스(예: 컨트롤)이면 Paint 이벤트 핸들러를 작성해서 사용

```
private void pictureBox1_Paint(object sender, PaintEventArgs e){  
    Graphics g = e.Graphics; // 그래픽 객체 참조  
    ... } // pictureBox1 컨트롤의 Paint 이벤트 핸들러 작성
```


- 다른 메소드에서 폼에 그림을 그려야 할 경우
 - 현재 폼에서 사용하는 Graphics 객체를 사용해야 함
 - Form 클래스의 CreateGraphics() 메소드는 폼에서 사용하는 Graphics 객체를 반환

```
private void Form1_MouseClick(object sender, MouseEventArgs e) {  
    Graphics g = this.CreateGraphics(); // this는 폼의 객체 참조  
    Pen p = new Pen(Color.Red);  
    g.DrawElilipse(p, 8, 98, 4, 4);  
    p.Dispose();  
    g.Dispose();  
... } // Form1의 마우스 클릭 이벤트 핸들러 작성
```

— 프로그램에서 원하는 시점에 화면 출력을 하고자 할 경우

- OnPaint() 메소드를 직접 호출하는 대신
Invalidate() 메소드를 이용
- Invalidate() 메소드는 특정 폼(또는 컨트롤) 영역을 무효화하고 그리기 이벤트 메시지를 폼(또는 컨트롤)로 보냄
- Invalidate() 메소드는 무효화
영역(다시 그려야 할 필요가 있는 영역)을 인수로 주는데, 만약
없으면 클라이언트 창 전체 영역을 의미
- 만약 원하는 시점에 즉시 화면에 출력하고자 한다면,
Invalidate() 메소드를 사용하여 무효화 영역을 지정하고 난 후,
Control 클래스의 Update() 메소드를 호출 - 보류 중인 그리기 요청 실행
- 컨트롤이 자신과 모든 자식을 강제로 다시 그리도록 하는
Refresh() 메소드를 호출하면, Invalidate(true)를 호출한 후
Update() 메소드를 호출한 것과 같은 효과

Double Buffering

- 폼이나 컨트롤에 움직이는 애니메이션을 출력할 경우 화면의 깜빡임 (**flickering**)을 방지하기 위하여, C#에서 내부적으로 제공해주는 더블 버퍼링을 사용

```
public Form1() {  
    InitializeComponent();  
    SetStyle(ControlStyles.DoubleBuffer, true); // 더블버퍼링  
    SetStyle(ControlStyles.AllPaintingWmPaint, true);  
    SetStyle(ControlStyles.UserPaint, true);  
}
```