07장 캔버스

캔버스(Canvas) 패널은 전통적 그래픽 환경에 가장 가까운 레이아웃이다. 엘리먼트가 어디에 위치할지를 좌표로 지정하는 방식이다. WPF의 다른 부분과 마찬가지로 이 좌표계도 좌측 상단을 기준으로 하는 1/96인치의 장치 독립적 단위를 사용한다.

엘리먼트에는 X나 Y, Left나 Top과 같은 프로퍼티가 없다는 사실을 이미 확인했었다. Canvas 패널을 사용할 때는 Canvas.SetLeft와 Canvas.SetTop 정적 메소드를 사용해 자식 엘리먼트의 위치를 지정해야 한다. DockPanel에 정의된 SetDock 메소드처럼(그리고 Grid에 정의된 SetRow, SetColumn, SetRowSpan, SetColumnSpan 메소드처럼) SetLeft와 SetTop은 Canvas 클래스에 정의되어 있는 첨부 프로퍼티와 연결되어 있다. Canvas의 우측 하단을 기준으로 해서 자식 엘리먼트의 오른쪽 위치나 하단 위치를 지정할 필요가 있다면 Canvas.SetRight나 Canvas.SetBottom을 사용한다.

몇 가지 Shapes 클래스(구체적으로 Line, Path, Polygon, Polyline)에는 이미 좌표 데이터가 저장되어 있다. 이런 엘리먼트를 Canvas 패널의 Children 컬렉션에 추가하고 좌표를 지정하지 않으면 엘리먼트에 있는 좌표 데이터를 이용해 위치를 잡을 것이다. SetLeft나 SetTop을 사용해 명시적으로 좌표를 지정하면 그 위치는 엘리먼트의 좌표 데이터에 추가된다.

컨트롤과 같은 여러 엘리먼트들은 Canvas 위에서 적절하게 크기가 조절될 것이다. 그러나 일부 엘리먼트(예를 들면 Rectangle이나 Ellipse 클래스)는 이렇게 되지 않으므로, 명시적으로 Width와 Height에 값을 대입해야 한다. 또한 Canvas 패널 자체의 Width와 Height 프로퍼티에도 값을 지정하는 것이 일반적이다.

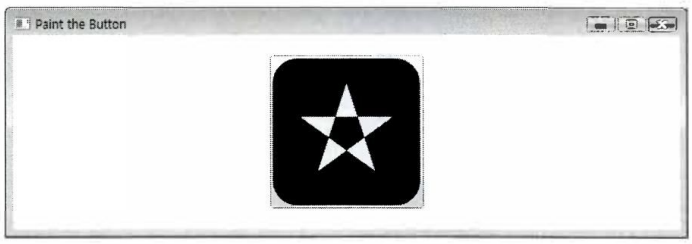
Canvas 패널에 엘리먼트를 겹치는 것도 가능하며, 때로는 그편이 바람직하다. 앞서 보았던 것처럼 여러 엘리먼트를 Grid의 한 셀에 넣을 수 있었지만 그 결과를 통제하기가 어려웠다. Cavas에서는 통제와 예측의 측면에서 볼 때 엘리먼트의 배치가 상당히 용이하다. Children 컬렉션에 먼저 추가한 엘리먼트는 나중에 추가한 엘리먼트에 가려진다.

다음 프로그램은 1.5인치의 둥근 모서리를 가진 사각형 배경에 직경 1인치의 노란색 별이 있는 버튼을 출력하는 예제다.

PaintTheButton.cs

```
//-----------------------------------------------
// PaintTheButton.cs (c) 2006 by Charles Petzold
//-----------------------------------------------
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;

namespace Petzold.PaintTheButton
{
    public class PaintTheButton : Window
    {
        [STAThread]
        public static void Main()
        {
            Application app = new Application();
            app.Run(new PaintTheButton());
        }
        public PaintTheButton()
        {
            Title = "Paint the Button";

            // 버튼을 생성하고 창의 컨텐트로 지정
            Button btn = new Button();
            btn.HorizontalAlignment = HorizontalAlignment.Center;
            btn.VerticalAlignment = VerticalAlignment.Center;
            Content = btn;

            // Cavas를 생성하고 버튼의 컨텐트로 지정
            Canvas canv = new Canvas();
            canv.Width = 144;
            canv.Height = 144;
            btn.Content = canv;

            // Rectangle을 생성해 Cavas의 자식으로 지정
            Rectangle rect = new Rectangle();
            rect.Width = canv.Width;
            rect.Height = canv.Height;
            rect.RadiusX = 24;
```

```
            rect.RadiusY = 24;
            rect.Fill = Brushes.Blue;
            canv.Children.Add(rect);
            Canvas.SetLeft(rect, 0);
            Canvas.SetTop(rect, 0);

            // Polygon을 생성해 Cavas의 자식으로 지정
            Polygon poly = new Polygon();
            poly.Fill = Brushes.Yellow;
            poly.Points = new PointCollection();

            for (int i = 0; i < 5; i++)
            {
                double angle = i * 4 * Math.PI / 5;
                Point pt = new Point(48 * Math.Sin(angle),
                                     -48 * Math.Cos(angle));
                poly.Points.Add(pt);
            }
            canv.Children.Add(poly);
            Canvas.SetLeft(poly, canv.Width / 2);
            Canvas.SetTop(poly, canv.Height / 2);
        }
    }
}
```



▶ 실행 결과

이 프로그램에서는 Button을 생성하고 Window의 컨텐트로 지정한다. 그리고 1.5인치의 정사각형 Canvas를 생성하고 버튼의 컨텐트로 설정한다. 버튼의 HorizontalAlignment와 VerticalAlignment 프로퍼티가 Center로 설정되었기 때문에 버튼의 크기는 Canvas 패널의 크기에 맞게 조정될 것이다.

그런 후 Rectangle을 생성하고, Width와 Height 프로퍼티를 Canvas와 같게 지정한다. 이 Rectangle을 다른 패널에서 했던 방법대로 Canvas의 Children 컬렉션에 추가한다.

```
canv.Children.Add(rect);
```

다음 구문은 Rectangle이 Canvas의 좌측 상단 모서리에 위치될 수 있게 한다.

```
Canvas.SetLeft(rect, 0);
Canvas.SetTop(rect, 0);
```

엄밀하게 말하면 기본 설정이 0이기 때문에 이 두 구문이 반드시 필요한 것은 아니다.

다음 단계는 Polygon과 관련된다. Polygon 클래스에는 Points란 프로퍼티가 정의되어 있는데, PointCollection 타입이며 다각형을 구성하는 점들이 저장된다. 단, 새로 생성된 Polygon 객체의 Points 프로퍼티는 null이다. PointCollection 타입의 객체를 명시적으로 생성해서 이를 Points 프로퍼티에 대입해야 한다. PaintTheButton 프로그램에서는 인자가 없는 PointCollection 생성자를 사용해 이를 수행하는 방법을 보여준다. 그 후에 Add 메소드를 이용해 각 점을 컬렉션에 추가한다.

PointCollection에는 IEnumerable<Point> 타입의 인자를 받는 생성자도 정의되어 있다. 여기에는 List<Point> 컬렉션뿐만 아니라 Point 객체의 배열을 받을 수도 있다.

for 루프에 있는 코드는 별을 이루는 점들의 좌표를 계산한다. 이 점은 점 (0, 0)을 타원의 중심으로 하며, -48부터 48까지의 범위를 갖는 X와 Y 좌표를 가질 것이다. 타원을 Canvas의 중앙에 위치시키기 위해 다음 구문을 사용한다. 이렇게 하면 폴리곤 내의 모든 점들의 위치는 Canvas의 폭과 높이의 절반만큼 더해진다.

```
Canvas.SetLeft(poly, canv.Width / 2);
Canvas.SetTop(poly, canv.Height / 2);
```

별의 오각형 내부는 채워지지 않음을 알 수 있다. Polygon에 정의된 FillRule 프로퍼티의 기본 설정 때문이다. FillRule.EvenOdd 열거형 값을 사용하면 외부 공간으로부터 폐곡선을 구분 짓는 다각형의 선들에 근거한 알고리즘을 기반으로 한다. 외부 공간에서 폐곡선 내로 선을 그었을 때 가로지르는 선의 수가 홀수인 경우에만 영역을 채운다. FillRule을 FillRule.Nonzero로 설정하면 별의 중앙도 채워진다.

컨트롤을 창에 배치하는 작업을 하기 위해 Canvas를 사용할 수 있긴 하지만 이 작업에 도움이 되기보다는 방해가 된다고 느낄 것이다. Canvas는 그래픽을 출력하거나(2부에서 살펴볼 것이다) 마우스로 그리는 작업을 할 때는 매우 좋지만(9장에서 살펴볼 것이다) 일반적인 레이아웃 작업을 할 때는 피하는 것이 좋다.

다음 클래스는 Canvas를 상속받아 게임에서 사용되는 타일을 구현한 것이다. 클래스에서 타일의 크기를 의미하는 2/3인치를 SIZE란 상수로 정의했고, 경계선의 두께를 의미하는 1/16인치를 BORD란 상수로 정의했다. 이 경계선은 입체 효과를 주기 위해 그림자 효과를 주게 된다. 관습상으로 컴퓨터 화면상의 객체는 광원에 반사된 모습을 흉내 내기 위해 좌측 상단 모서리

를 밝게 한다. 경계선은 두 개의 Polygon 객체로 구성되며, 그 색은 상단과 왼쪽에 쓰일 SystemColors.ControlLightLightBrush와 하단과 오른쪽에 쓰일 SystemColors.ControlDarkBrush다. 이렇게 해서 타일의 내부는 돌출되어 보이게 된다.

Tile.cs

```
//-------------------------------------
// Tile.cs (c) 2006 by Charles Petzold
//-------------------------------------
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;

namespace Petzold.PlayJeuDeTacquin
{
    public class Tile : Canvas
    {
        const int SIZE = 64;    // 2/3인치
        const int BORD = 6;     // 1/16인치
        TextBlock txtblk;

        public Tile()
        {
            Width = SIZE;
            Height = SIZE;

            // 좌측과 상단의 밝게 반사된 경계
            Polygon poly = new Polygon();
            poly.Points = new PointCollection(new Point[]
            {
                new Point(0, 0), new Point(SIZE, 0),
                new Point(SIZE-BORD, BORD),
                new Point(BORD, BORD),
                new Point(BORD, SIZE-BORD), new Point(0, SIZE)
            });
            poly.Fill = SystemColors.ControlLightLightBrush;
            Children.Add(poly);

            // 우측과 하단의 그림자진 경계
            poly = new Polygon();
```

```
            poly.Points = new PointCollection(new Point[]
            {
                new Point(SIZE, SIZE), new Point(SIZE, 0),
                new Point(SIZE-BORD, BORD),
                new Point(SIZE-BORD, SIZE-BORD),
                new Point(BORD, SIZE-BORD), new Point(0, SIZE)
            });
            poly.Fill = SystemColors.ControlDarkBrush;
            Children.Add(poly);

            // 중앙의 텍스트
            Border bord = new Border();
            bord.Width = SIZE - 2 * BORD;
            bord.Height = SIZE - 2 * BORD;
            bord.Background = SystemColors.ControlBrush;
            Children.Add(bord);
            SetLeft(bord, BORD);
            SetTop(bord, BORD);

            // 텍스트를 출력
            txtblk = new TextBlock();
            txtblk.FontSize = 32;
            txtblk.Foreground = SystemColors.ControlTextBrush;
            txtblk.HorizontalAlignment = HorizontalAlignment.Center;
            txtblk.VerticalAlignment = VerticalAlignment.Center;
            bord.Child = txtblk;
        }

        // 텍스트로 설정하는 public 프로퍼티
        public string Text
        {
            set { txtblk.Text = value; }
            get { return txtblk.Text; }
        }
    }
}
```

이 프로그램에서 `Polygon` 객체의 `Points` 컬렉션을 설정하는 방법에 주목하자. `PointCollection`의 생성자에 전체 `Point` 배열을 정의하고 있다!

`Tile` 클래스에서 타일 중앙에 텍스트를 출력해야 한다. `TextBlock` 엘리먼트의 실제 크기를 알아내어 `Canvas`의 중앙에 위치시킬 수 있지만 여기에서는 더 쉬운 방법을 생각해보자. Tile 클래스에서 두 개의 폴리곤으로 경계선을 만든 후 `Border` 타입의 실체 객체를 생성한다.

Border는 Decorator를 거쳐서 FrameworkElement를 상속받는다. 이 Border 엘리먼트는 타일의 중앙에 위치한다. Decorator에는 Child란 프로퍼티가 정의되어 있어서 UIElement의 인스턴스 하나를 저장할 수 있는데, 여기에서는 TextBlock이 된다. TextBlock의 HorizontalAlignment와 VerticalAlignment 프로퍼티는 Center로 설정해서 Border 객체의 중앙에 위치시킨다. Border 객체는 단색의 경계선을 출력할 수 있지만 Tile에서는 이럴 필요가 없다.

Tile 클래스는 쥬 드 타킨(Jeu de Tacquin)이라는 퍼즐을 위한 클래스다. 쥬 드 타킨은 14-15 퍼즐이라고도 한다. 이 퍼즐은 1870년대에 미국인 퍼즐 제작자인 샘 로이드(Sam Loyd, 1841-1911)가 고안한 것으로 추정하고 있다. 이 게임은 4×4의 눈금 속에 15개의 숫자가 써진 정사각형으로 만들어진다. 한 칸은 빈칸이어서 주위에 있는 정사각형을 움직일 수 있다. 다음은 빈칸을 위한 클래스다.

Empty.cs

```
//---------------------------------------
// Empty.cs (c) 2006 by Charles Petzold
//---------------------------------------

namespace Petzold.PlayJeuDeTacquin
{
    class Empty : System.Windows.FrameworkElement
    {
    }
}
```

최초의 형태는 숫자가 써진 정사각형이 순서대로 배치되는데, 14와 15만 바뀌어 있다. 샘 로이드는 정사각형을 움직여 정확한 순서대로 맞추는 방법을 발견한 사람에게 1000달러의 상금을 걸었다. 이 문제를 해결하는 것은 불가능하다고 1879년의 미국 수학 저널(American Journal of Mathematics)에서 밝혀졌다. 『수학의 세계 4권』(The World Of Mathematics Vol. 4, Simon and Schuster, 1956)에는 이에 대한 분석이 실려 있다.

컴퓨터로 만들어진 쥬 드 타킨은 애플 매킨토시에서 개발된 최초의 게임 프로그램 중의 하나였으며, PUZZLE이라고 불렀다. 마이크로소프트 윈도우 소프트웨어 개발 키트(Microsoft Windows Software Development Kit)의 초기 버전에서도 이를 볼 수 있었는데(이름은 MUZZLE로 바뀌었다), C가 아닌 마이크로소프트 파스칼(Pacal)로 개발된 예제 프로그램이었다.

매킨토시와 윈도우 버전 모두 셀을 정확한 순서로 표시하고, 메뉴에서 선택해 셀을 뒤섞을 수 있었다. 이 책의 예제에서는 셀을 뒤섞기 위한 Button을 만들며, DispatcherTimer를 생성해서 뒤섞는 동작을 직접 확인할 수 있게 했다.

레이아웃은 두 개의 자식이 있는 StackPanel로 시작한다. StackPanel의 자식은 뒤섞기 버튼과 Border 객체다. 이 Border는 미적인 목적으로 Button과 Border의 내용을 분리하기 위한

얇은 선을 그린다. Border의 Child 프로퍼티는 UniformGrid 패널을 지정했으며, 여기에 15개의 Tile 객체와 하나의 Empty 객체가 들어간다.

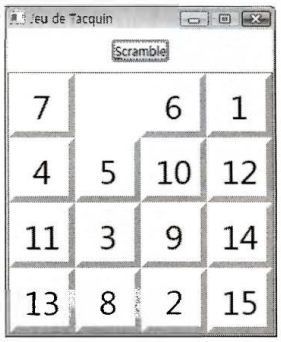
PlayJeuDeTacquin.cs

```
//---------------------------------------------
// PlayJeuDeTacquin.cs (c) 2006 by Charles Petzold
//---------------------------------------------
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Threading;

namespace Petzold.PlayJeuDeTacquin
{
    public class PlayJeuDeTacquin : Window
    {
        const int NumberRows = 4;
        const int NumberCols = 4;

        UniformGrid unigrid;
        int xEmpty, yEmpty, iCounter;
        Key[] keys = { Key.Left, Key.Right, Key.Up, Key.Down };
        Random rand;
        UIElement elEmptySpare = new Empty();

        [STAThread]
        public static void Main()
        {
            Application app = new Application();
            app.Run(new PlayJeuDeTacquin());
        }

        public PlayJeuDeTacquin()
        {
            Title = "Jeu de Tacquin";
            SizeToContent = SizeToContent.WidthAndHeight;
            ResizeMode = ResizeMode.CanMinimize;
            Background = SystemColors.ControlBrush;
```

```
        // StackPanel을 생성하고 창의 컨텐트로 지정
        StackPanel stack = new StackPanel();
        Content = stack;

        // 상단의 버튼을 생성
        Button btn = new Button();
        btn.Content = "_Scramble";
        btn.Margin = new Thickness(10);
        btn.HorizontalAlignment = HorizontalAlignment.Center;
        btn.Click += ScrambleOnClick;
        stack.Children.Add(btn);

        // 미적 목적으로 경계선을 생성
        Border bord = new Border();
        bord.BorderBrush = SystemColors.ControlDarkDarkBrush;
        bord.BorderThickness = new Thickness(1);
        stack.Children.Add(bord);

        // UniformGrid를 생성하고 Border의 자식으로 지정
        unigrid = new UniformGrid();
        unigrid.Rows = NumberRows;
        unigrid.Columns = NumberCols;
        bord.Child = unigrid;

        // Tile 객체를 생성해서 하나를 제외한 모든 셀에 추가
        for (int i = 0; i < NumberRows * NumberCols - 1; i++)
        {
            Tile tile = new Tile();
            tile.Text = (i + 1).ToString();
            tile.MouseLeftButtonDown += TileOnMouseLeftButtonDown;
            unigrid.Children.Add(tile);
        }

        // Empty 객체를 생성해서 마지막 셀에 추가
        unigrid.Children.Add(new Empty());
        xEmpty = NumberCols - 1;
        yEmpty = NumberRows - 1;
    }

    void TileOnMouseLeftButtonDown(object sender,
                                   MouseButtonEventArgs args)
    {
        Tile tile = sender as Tile;
```

```
        int iMove = unigrid.Children.IndexOf(tile);
        int xMove = iMove % NumberCols;
        int yMove = iMove / NumberCols;

        if (xMove == xEmpty)
            while (yMove != yEmpty)
                MoveTile(xMove, yEmpty + (yMove - yEmpty) /
                    Math.Abs(yMove - yEmpty));
        if (yMove == yEmpty)
            while (xMove != xEmpty)
                MoveTile(xEmpty + (xMove - xEmpty) /
                    Math.Abs(xMove - xEmpty), yMove);
    }

    protected override void OnKeyDown(KeyEventArgs args)
    {
        base.OnKeyDown(args);

        switch (args.Key)
        {
            case Key.Right: MoveTile(xEmpty - 1, yEmpty);  break;
            case Key.Left:  MoveTile(xEmpty + 1, yEmpty);  break;
            case Key.Down:  MoveTile(xEmpty, yEmpty - 1);  break;
            case Key.Up:    MoveTile(xEmpty, yEmpty + 1);  break;
        }
    }

    void ScrambleOnClick(object sender, RoutedEventArgs args)
    {
        rand = new Random();
        iCounter = 16 * NumberCols * NumberRows;

        DispatcherTimer tmr = new DispatcherTimer();
        tmr.Interval = TimeSpan.FromMilliseconds(10);
        tmr.Tick += TimerOnTick;
        tmr.Start();
    }

    void TimerOnTick(object sender, EventArgs args)
    {
        for (int i = 0; i < 5; i++)
        {
```

```
                MoveTile(xEmpty, yEmpty + rand.Next(3) - 1);
                MoveTile(xEmpty + rand.Next(3) - 1, yEmpty);
            }
            if (0 == iCounter--)
                (sender as DispatcherTimer).Stop();
        }
        void MoveTile(int xTile, int yTile)
        {
            if ((xTile == xEmpty && yTile == yEmpty) ||
                xTile < 0 || xTile >= NumberCols ||
                    yTile < 0 || yTile >= NumberRows)
                return;

            int iTile = NumberCols * yTile + xTile;
            int iEmpty = NumberCols * yEmpty + xEmpty;

            UIElement elTile = unigrid.Children[iTile];
            UIElement elEmpty = unigrid.Children[iEmpty];

            unigrid.Children.RemoveAt(iTile);
            unigrid.Children.Insert(iTile, elEmptySpare);
            unigrid.Children.RemoveAt(iEmpty);
            unigrid.Children.Insert(iEmpty, elTile);

            xEmpty = xTile;
            yEmpty = yTile;
            elEmptySpare = elEmpty;
        }
    }
}
```



▶ 실행 결과

Tile 객체의 이동은 UniformGrid의 Children 컬렉션을 조작함으로써 처리되며, 이 작업은 파일의 맨 아래에 있는 MoveTile 메소드에서 한다. MoveTile에 넘기는 두 개의 인자는 움직일 타일의 가로와 세로 좌표다. 이 타일이 어디로 움직일지는 결정할 수 있다. 타일은 빈칸 즉, Empty 객체를 향해서만 움직일 수 있기 때문이다. MoveTile의 코드 시작 부분에서 바꿔야 할 두 엘리먼트를 Children 컬렉션의 인덱스 형태로 얻어오고 있는 부분을 주목하자. 그 후에 이 둘에 대해 RemoveAt과 Insert를 호출해서 서로 위치를 바꾸고 있다. 이때 구문의 순서에 주의해야 한다. 자식 엘리먼트를 제거하거나 삽입한 후에는 모든 엘리먼트의 인덱스들도 더불어 바뀌기 때문이다.

이 프로그램은 키보드와 마우스 인터페이스를 모두 지원한다. 마우스 인터페이스에 대한 코드는 Tile 객체의 MouseLeftButtonDown 이벤트를 처리하는 이벤트 핸들러에 있다. 클릭된 특정 Tile 객체는 이벤트 핸들러의 첫 번째 인자를 형 변환함으로써 쉽게 얻을 수 있다. while 루프가 있는 이유는 한 번의 마우스 클릭으로 여러 개의 타일을 한꺼번에 움직이는 것을 처리하기 위해서다.

키보드 인터페이스는 OnKeyDown 메소드를 오버라이딩해서 처리한다. 커서 이동 키를 이용해 빈 셀에 인접한 타일을 빈 셀로 이동시키게 된다.

PlayJeuDeTacquin 프로그램에서 행과 열의 수는 두 개의 필드로 지정하고 있으며, 원하는 수로 이 값을 변경할 수 있다. 단, 자리수가 4자리 이상이 되면 타일에 숫자 출력하는 곳이 부족할 것이고 뒤섞는 과정도 더 오래 걸릴 것이다.