**29장 – 그래픽 변환**

**◎ PT에 앞서서 핵심포인트**

- Transform의 파생 클래스들의 기능 파악

- RenderTransform과 LayOutTransform 차이점을 이해하고 언제 사용하는지 파악

**Transform(변형시키다) 파생 클레스 계층도**

**Object**

**DispatcherObject (abstract)**

**DependencyObject**

**Freezable (abstract)**

**Animatable (abstract)**

**GeneralTransform (abstract)**

**GeneralTransformGroup**

**Transform (abstract)**

**TranslateTransform**

**ScaleTransform**

**SkewTransform**

**RotateTransform**

**MatrixTransform**

**TransformGroup**

**1) 변환이 무엇인가?**

- 때로는 그림을 크게 또는 작게 만들고 여러 각도로 회전 시켜야 할 경우가있다.

- 변환은 그래픽 객체의 모든 좌표를 일괄적으로 수정할 수 있는 좀 더 일반화되고 시스템적인 접근 방법이다.

- **RenderTransform**, **LayOutTransform**의 타입은 **Transform** 이다.

- **UIElement**은 **RenderTransform**을 가지고 있다.

- **FrameworkElement**은 **LayoutTransform**을 가지고 있다.

**예제 1) 간략정리 (1072 P)**

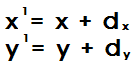
|  |  |  |
| --- | --- | --- |
| **TranslateTransform** |  | 2차원 x-y 좌표계에서 개체를 변환 이동 |
| **ScaleTransform** |  | 2차원 x-y 좌표계에서 개체의 배율을 조정 |
| **SkewTransform** |  | 개체에 적용되는 2차원 기울이기 |
| **RotateTransform** |  | 2차원 x-y 좌표계에서 지정한 점을 기준으로 개체를 시계 방향으로 회전 |

※ WPF 이전 그래픽 환경 프로그램, 드로잉 표면의 원점을 기준으로 한다.

※ WPF의 회전은 Button 자신의 원점을 기준으로 한다. (버튼 좌측 상단)

**예제 2) 간략정리 (1075 P)**

1076p를 보면 **TranslateTransform**에 대한 공식을 볼 수 있다.

는 원래 위치 (초기 값)

 TranslateTransform의 x, y 값

 버튼이 그려질 위치

**※ ScaleTransform과 SkewTransforme**도 각 해당 공식이 존재 하니 조금 심도 있게

공부 하고 싶은 사람들은 알아서 봐둘 것

**※ LayoutTransform**은 위치 이동에 관한 요소를 무시한다.

**※참고자료 Transform 파생 클래스의 프로퍼티들**

**TranslateTransform (x, y)**

**ScaleTransform (scaleX, scaleY, CenterX, CenterY) default (1 , 1, 0 , 0)**

**SkewTransform (AngleX, AngleY, CenterX, CenterY) default (0 , 0, 0 , 0)**

**RotateTransform (Angle, CneterX, CenterY) default (0, 0, 0)**

**2) RenderTransform과 LayoutTransform**

- RenderTransform의 경우 시스템 엘리먼트의 OnRender 메소드가 그린 이미지를 확보한다음 변환을 적용하고 그 결과를 화면에 뿌림.

- LayoutTransform의 경우 조금이라도 바뀌면 레이아웃을 새로 만들기 위한 절차가 시작된다. (MeasureOverride와 ArrangeOverrids가 호출됨)

※ MeasureOverride는 Size결정

※ ArrangeOverridssms Size 좌표를 지정 **-> 출력**

**예제 3) 간략정리 (1086 P)**

|  |  |
| --- | --- |
|  |  |
| RenderTransform | LayoutTransform |

-RenderTransform의 경우 자신보다 먼저 생성된 엘리먼트 보다는 위에, 늦게 생성

된 엘리먼트 보다는 아래에 표시

-변환을 이용해서 애니메이션등을 구현할때는 바인딩 처리 효율을 높이기 위해

RenderTransform을 사용하는 것이 좋다.

**3) RenderTransformOrigin란?**

- 앞서 본 XAML파일들의 버튼은 좌측 상단 (0,0)을 기준으로 회전을 했다.

중앙을 기준으로 회전시키기 위해서는 엘리먼트의 크기를 알아야 하는데

이는 레이아웃에 의해 결정되고 런타임 때 결정되기 때문에 예측 불가능

- 그래서 RenderTransformOrigin을 사용하여 중앙 정렬가능 (1089 P)

|  |
| --- |
| <Button Content=”Button” **RenderTransformOrigin**=”0.5 0.5”>  <Button.RenderTransform>  <RotateTransform Angle=”45”/>  </Button.RenderTransform>  </Button> |

**4) MatrixTransform과** **TransformGroup**

**※ MatrixTransform 을 사용하여 RotateTransform, ScaleTransform, SkewTransform 및 TranslateTransform 클래스에서 제공하지 않는 사용자**

**지정 변환을 만들 때 쓴다.**

**※ 행렬 유사 변환을 곱하여 변환이 따르는 회전 및 기울이기 변환과 같은 선형 변환을 만들 수 있습니다.**

**Matrix 구조체에 대한 설명**

|  |
| --- |
| - 변환을 나타내는 행렬은 Matrix라는 이름의 구조체로 캡슐화 되어 있다.  - Matrix 구조체는 읽고 쓰기 가능한 6개의 프로퍼티를 갖는다.  - 3번째 행의 값들은 고정되어 있어 바꿀 수 없다.    m11 변환 Matrix의 (1, 1) 위치에 있는 값입니다.  m12 변환 Matrix의 (1, 2) 위치에 있는 값입니다.  m21 변환 Matrix의 (2, 1) 위치에 있는 값입니다.  m22 변환 Matrix의 (2, 2) 위치에 있는 값입니다.  offset 변환 Matrix의 (3, 1) 위치에 있는 값입니다.  offset 변환 Matrix의 (3, 2) 위치에 있는 값입니다. |

**예제 4) 간략정리 (1096 P 보면서 간단하게 개념정리)**

**※ TransformGroup는** **여러 Transform 개체를 단일 속성에 적용하려는 경우**

**사용한다.**

**※ 특별한 경우 개체의 순서가 바뀌어도 상관 없지만 대부분 순서에 따라 결과가**

**달라지게 된다.**

**예제 5) 순서에 따른 결과값 (MSDN 참조)**

|  |
| --- |
| <Canvas  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  Width="200" Height="200">  <TextBlock FontSize="28"  Canvas.Left="100" Canvas.Top="100"  Width="50" Height="50"  Text="Hello">  <TextBlock.RenderTransform>  <TransformGroup>  <RotateTransform Angle="45" />  <SkewTransform CenterX="0" CenterY="0" AngleX="60"/>  </TransformGroup>  </TextBlock.RenderTransform>  </TextBlock>  </Canvas> |