**08. 의존프로퍼티**

**- 의존 프로퍼티란 뭘까?**

'의존' 이란 간단히 말해, 어떤 객체의 모습 또는 행동이 의존 속성에 의해 영향을 받는다 라는 뜻.

-의존이라는 말은 의지하다 기대다 라는 말로 해석할수 있다.

<일반적인 엘리먼트 트리 구조>

|  |
| --- |
| Window  ↓  Grid  ↓  Button  ↓  TextBlock |

🡺 프로그램 내에서 Button의 FontSize를 바꿀때 TextBlock의 FontSize가 명시되어 있지 않다면, Button의 FontSize의 값이 변하였을 때 TextBlock의

FontSize도 변한다는 것이다.

🡺이 말은 즉 TextBlock의 크기가 변하면 Button의 크기자체가 변했었고,

이는 Grid나 Window의 Height, Width프로퍼티에도 영향을 준다는 것이다.

이러한 반응들을 포함하기위한 일관된 방식을 위해 태어난 개념이 바로

**‘의존프로퍼티(dependency properties)’**

의존프로퍼티라 부르는 이유는 이 프로퍼티가 외부의 영향에 의존적인 프로퍼티이기 때문이다.

이런식으로 의존프로퍼티불리는 프로퍼티는 FontSize 말고도 몇가지 더있는데, UIElement에

엘리먼트 트리를 통해 계승되는 네가지 프로퍼티 프로퍼티(AllowDrop,IsEnabled,IsVisible,SnapToDevicePixels)가 있고, 199페이지에 보면 몇가지가 더있다.

예제1 **SetFontSizeProperty**

**기존 .Net 프로그래밍 에서는**

|  |
| --- |
| **Public double FontSize**  **{**  **Get**  **{**  **Return fontsize;**  **}**  **Set**  **{**  **Fontsize = value;**  **……………………..**  **}**  **}** |

**Set 접근자에 있는 생략표시는 여기에 실행되야 할 코드가 더 많이 있다는**

**뜻이며 설정된 값을 컨트롤에서 필요로 할수 있으며, 새로운 크기로 다시 그려야**

**할 것이다. Fontsizechanged 이벤트를 발생시켜야 할수도 있고 엘리먼트 트리의 자손으로 fontsize를 계승 해야 할수도 있다.**

**WPF에서 의존 프로퍼티를 사용하면 공통된 시나리오를 통해 많은 통지 작업을 자동으로 수행 할수 있다,**

**- WPF 의존프로퍼티**

|  |
| --- |
| **public class Control : FrameworkElement**  **{**  **...**  **public static readonly DependencyProperty FontSizeProperty;**  **...**  **}** |

**🡺** 기본public이며 static임으로 클래스이름으로 참조된다(**Control.FontSizeProperty**).

🡺 static readonly 필드는 필드 자체의 정의부분이나 정적 생성자에서 설정할 수 있음

🡺 일반적으로 클래스에서는DependencyProperty.Register라는 메소드를 호출함으로써 DependencyProperty 타입의 객체를 생성

**- DependencyProperty.Register 메소드?**

|  |
| --- |
| FontSizeProperty = DependencyProperty.Register("FontSize", typeof(double), typeof(Control)); |

**🡺** Register()의 파라미터는 의존프로퍼티와 관련된 프로퍼티의 텍스트 Name과 타입,

이 프로퍼티를 등록하는 클래스 타입등의 정보가 들어감

🡺 But! DependencyProperty 객체에는 프로퍼티의 중요한 특성을 나타내는 MetaData를 포함

**- 실제 Control 클래스 구조**

|  |
| --- |
| **public class Control : FrameworkElement**  **{**  **...**   **public static readonly** **DependencyProperty** **FontSizeProperty;**  **...**   **static Control()**  **{**   **FrameworkPropertyMetadata** **metadata =** **new** **FrameworkPropertyMetadata();**  **// 메타테이터의 기본값이 11임을 알수있다.**  **metadata.DefaultValue = 11;**  **// 엘리먼트의 크기에 영향을 주므로 true...**   **metadata.AffectsMeasure = true;**  **// 엘리먼트 트리를 통해 계승될 수 있다.**  **metadata.Inherit = true;**  **// 데이터 바인딩이 허용되며**  **metadata.IsDataBindingAllowed =true;**  **metadata.DefaultUpdateSourceTrigger =** **UpdateSourceTrigger.PropertyChanged;**  **// 메타데이터가 포함된다.**  **FontSizeProperty =**  **DependencyProperty.Register("FontSize",typeof(double),**  **typeof(Control),metadata,** **ValidateFontSize);**  **}**  **// 폰트사이즈를 검증하는 메소드**  **static bool** **ValidateFontSize(object obj)**  **{**  **double dFontSize = (double) obj**  **return dFontSize > 0 && dFontSize <= 35791;**  **}**  **....**  **}** |

🡺 실제적으로 Control 클래스는 SetValue와 GetValue를 통해서 FontSize프로퍼티를 정의한다.

|  |
| --- |
| **public class Control : FrameworkElement**  **{**  **...**  **public double FontSize**  **{**  **get  {  return (double) GetValue(FontSizeProperty); }**  **set  {  SetValue(FontSizeProperty, value); }**  **}**  **}** |

🡪 SetValue에서는 DependencyProperty에 기반하여 Value 값을 평가한다.

**- SetValue와 GetValue는 어디서 나왔나?**

🡺 Set, Get메소드들은 DependencyObject 클래스에 정의되어 있으며, Window나 기타 여러 것들이 이 클래스에서 파생되었다.

|  |
| --- |
| Object    ㄴDispatcherObject (abstract)       ㄴ**DependencyObject**           ㄴVisual (abstract)              ㄴUIElement                 ㄴFrameworkElement                    ㄴControl |

🡺 DependencyObject와 DependencyProperty는 둘 다 클래스

🡺 WPF의 많은 클래스들은 DependencyObject 클래스의 자손

그래서 SetValue와 GetValue 메소드가 존재

🡺 이 메소드들은 정적 DependencyProperty 객체로 정의된 필드를 다룬다

SetValue와 GetValue 메소드에 넘기는 DependencyProperty 객체는 static이지만,

SetValue와 GetValue는 인스턴스 메소드이다.

그리고 이 메소드들은 특정 인스턴스와 관련된 값을 얻거나 설정한다.

DependencyObject는 현재의 값을 그대로 유지하며, 모든 루틴들도 처리한다.

예를 들어 특정 Control 인스턴스에 대해서 아직 SetValue가 호출되지 않았다면?

🡪 GetValue는 FontSizeProperty의 메타데이터에 있는 DefaultVlue 프로퍼티를 반환한다.

이로써 맨 위에서의 **프로퍼티 계승에 대한 우선순위가 동작**하게 되는 것이다.

**여기서 잠시, 위에서 나온 ‘의존속성’ 의 구성 요소를 자세히 살펴보면?**

|  |
| --- |
| 1. System.Windows.**DependencyObject**  2. System.Windows.**DependencyProperty**  3. System.Windows.**PropertyMetadata** |

**1. DependencyObject ;** '의존 속성'을 가지는 객체

🡺 WPF의 대부분의 UI 요소들은 이 DependencyObject를 상속받아 구현되어 있다.

이 DependencyObject는 GetValue 와 SetValue 메소드가 있는데,

모든 '의존 속성'은 이 GetValue 와 SetValue를 통해 구현되어 있다.

**2. DependencyProperty** ; '의존 속성' 자체를 정의한 것

🡺 예를 들어 Button의 Background Brush에 대한 DependencyProperty가 있고,

또한 Brush의 Color에 대한 DependencyProperty도 있다.

**3. PropertyMetadata** ; DependencyProperty의 부가 정보

일단 DependencyProperty는 DependecyObject의 인스턴스 단위가 아닌 클래스 단위로 정의된다. 모든 Button에 Background Brush가 있는 것이지, 어떤 Button에는 Brush가 있고, 어떤 Button에는 Brush가 없는 것이 아니기 때문이다. 그냥 무조건 세트라고 보면 된다.

따라서 DependencyProperty는 DependencyObject의 **static(정적)멤버**로 정의된다.

**그래서** **SetFontSizeProperty 예제에서 폰트 사이즈가 변하게 되면**

🡺 그 값이 적절한지검증을 하고. (위에서는 ValidateFontSize 메소드가 이를 처리)

🡺 컨트롤에서 크기를 다시 계산하고 그린다.

🡺 다시 그릴 때 새로운 FontSize를 사용하며

🡺 Inherits 플래그가 true이기 때문에 새 값을 트리를 통해 아래로 전달해야한다.

(물론 자식객체가 FontSize를 명시적으로 선언했다면 적용이 되지 않을 것이다.)

예제2 **dependency properties 참조**

이전에 살펴본 FontSize 프로퍼티는 control에 정의되어 있으며 Button과

Window 모두가 이를 상속 받는다. 예제2 에서는 spaceButton에 추가한 새로운 space 프로퍼티는 엘리먼트 트리를 통해 계승되지만 다른 어디에도 정의 된 곳이 없다.

프로퍼티 계승에 대해 설명하기 위해 같은 의존 프로퍼티를 구현한 SpaceWindow.cs 도 그자체는 프로퍼티를 가지고 있지만 아무런 일도 하지않는다.

**- 첨부 프로퍼티?**

임의의 객체에 추가하기 위해 사용되는 특별한 의존 프로퍼티

🡺 원래는 가지지 않는 속성을 추가 해 주는 것 (간결하고 확장 가능한 프로그래밍 가능)

하나의 스택 패널 안에 5개 텍스트박스가 있고, 그 5개 텍스트 박스의 폰트를 모두 맑은고딕으로 설정하기 위해 스택패널의 폰트 속성을 설정하려 하니 스택패널에는 폰트 속성이 존재하지 않아서 에러가 발생. 이럴때 첨부프로퍼티를 사용하여 폰트 속성을 추가해 준다.

(예제참고는 5장에서 나왔던 버튼)

