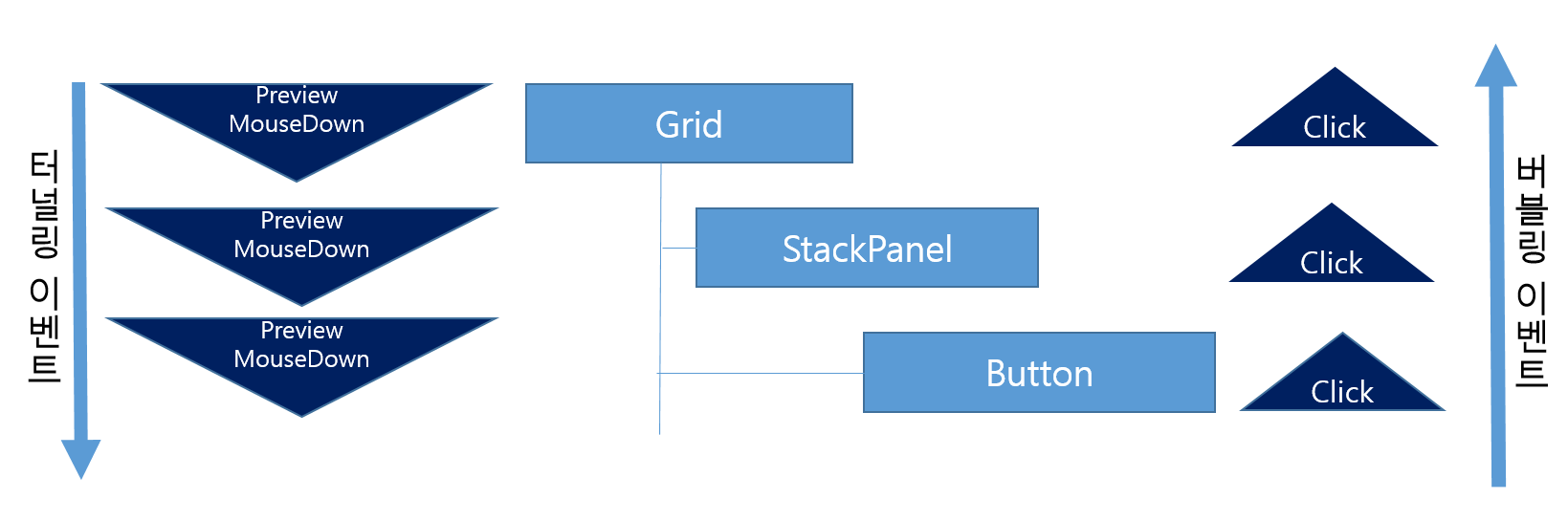
**(C#, WPF강좌)WPF이벤트, 이벤트의 라우팅(RoutedEvent), 버블링(Bubbling) 이벤트, 터널링(Tunneling) 이벤트**

* 이벤트 라우팅은 **“어떤 이벤트가 컨트롤의 하위 또는 상위로 전달되는 것”**을 이야기 하며 WPF에서 광범위하게 이용되는 방법입니다.
* 이벤트 라우팅은 세가지로 분류할 수 있는데 이벤트가 발생했을 때 현재 컨트롤에서 상위로 올라가면서 전달되는 경우를 “버블링 이벤트(Bubbleing Event)”라고 하고 반대로 자식 컨트롤로 전달되는 경우를 “터널링(Tunneling Event) 이벤트”라고 합니다. 하나의 엘리먼트에서만 발생하게 된다면 다이렉트 이벤트(Direct Event)라고 합니다.
* **터널링 이벤트의 경우 접두사로 preview를 붙이는데,** PreviewMouseDown, PreviewDragDown 등으로 쓰여지는 것들을 Tunneling 이벤트라고 이해하면 됩니다.
* **터널링 이벤트는** 최상위 부모 컨트롤로부터 자기자신까지 이벤트가 진행되는 것인데, 이벤트가 자식 요소에게 전달되기 전에 부모의 이벤트가 먼저 호출되므로 부모 이벤트가 먼저 호출할 수 있는 기회를 제공하며, 자식의 이벤트 발생을 막거나 자식이 벤트 처리 전에 부모 요소가 무엇인가 수행할 필요가 있는 경우에 유용 합니다.



* 이벤트 핸들러 메소드는 RoutedEventArgs 매개변수를 가지는데 Source 속성을 통해 실제 이벤트를 발생시킨 요소에 대한 참조를 제공하며, 이 속성은 여러 요소에서 발생한 이벤트를 동일한 방법으로 처리하고자 할 때 특히 유용합니다.

<Border Height="50" Width="300" BoderBrush="Gray" BorderThickness="1">

<StackPanel Background="LightGray" Orientation=

"Horizontal" ***Button.Click="CommonClickHandler">***

<Button Name="YesButton" Width="Auto" >Yes</Button>

<Button Name="NoButton" Width="Auto" >No</Button>

<Button Name="CancelButton" Width="Auto" >Cancel</Button>

</StackPanel>

</Border>

**//sender는 이벤트를 등록한 객체**

**//e.Source는 이벤트 발생 객체**

private void CommonClickHandler(object sender, **RoutedEventArgs e**)

{

FrameworkElement feSource = **e.Source** as FrameworkElement;

switch (**feSource.Name**) **//버튼의 Name 속성의 값**

{

case "YesButton":

// do something here ...

break;

case "NoButton":

// do something ...

break;

case "CancelButton":

// do something ...

break;

}

}

* WPF의 라우팅된 이벤트에서 object sender는 이벤트를 등록한 객체, RoutedEventArgs의 Source 속성은 실제 이벤트를 발생시킨 객체를 의미합니다.
* **WPF의 이벤트 라우팅 모델은 자동으로 이벤트를 상위 객체로 라우팅 시켜줍니다.(버블링 이벤트)**
* 비주얼 스튜디오 -> WPF 응용프로그램 , **프로젝트명 : EventRoutingTest**
* **MainWindow.xaml**

<Window x:Class = "WPFRoutedEvents.MainWindow"

xmlns = "http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml"

Title = "MainWindow" Height = "450" Width = "604"

ButtonBase.Click = "**Window\_Click**" >

<Grid>

<StackPanel Margin = "20" ButtonBase.Click = "**StackPanel\_Click**">

<StackPanel Margin = "10">

<TextBlock Name = "txt1" FontSize = "18" Margin = "5" Text = "This is a TextBlock 1" />

<TextBlock Name = "txt2" FontSize = "18" Margin = "5" Text = "This is a TextBlock 2" />

<TextBlock Name = "txt3" FontSize = "18" Margin = "5" Text = "This is a TextBlock 3" />

</StackPanel>

<Button Margin = "10" Content = "Click me" Click = "**Button\_Click**" Width = "80"/>

</StackPanel>

</Grid>

</Window>

* **MainWindow.xaml.cs**

using System.Windows;

namespace WPFRoutedEvents

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

txt1.Text = "Button is Clicked";

}

private void StackPanel\_Click(object sender, RoutedEventArgs e)

{

txt2.Text = "Click event is bubbled to Stack Panel";

}

private void Window\_Click(object sender, RoutedEventArgs e)

{

txt3.Text = "Click event is bubbled to Window";

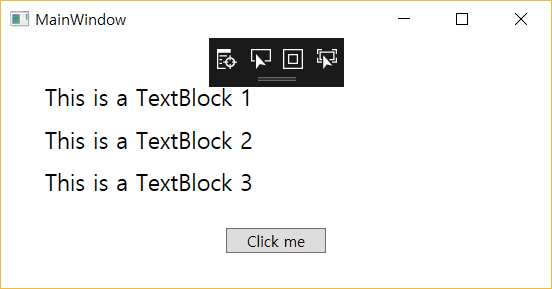
}

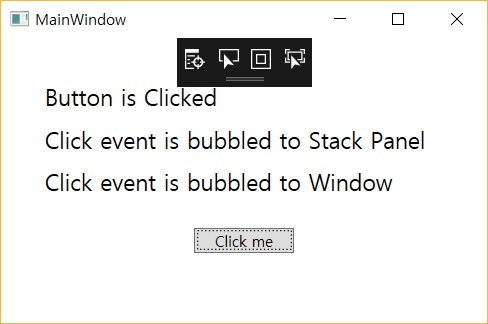
}

}

예제에서 버튼을 클릭시 클릭 이벤트가 버블링 되어 상위 객체로 라우팅 됩니다. 즉 버튼 클릭 이벤트 실행 -> 스택 패널 클릭 이벤트 실행 -> 윈도우 클릭 이벤트 순서로 됩니다.

* 실행화면





* 상위 객체로 라우팅 되는 것을 막으려면 **RoutedEventArgs의 Handled 속성을 true**로 설정하면 된다. 버튼 클릭 이벤트에 “e.Handled = true” 라고 하면 버튼까지만, 스택 패널 클릭 이벤트에 “e.Handled = true” 라고 한다면 스택 패널 까지만 이벤트가 전달됩니다.
* Preview 이벤트에 Handled 속성을 지정하면 Preview 이벤트의 터널링이 중단될 뿐만아니라 버블링 이벤트의 발생도 중단 됩니다.

private void StackPanel\_Click(object sender, RoutedEventArgs e) {

txt2.Text = "Click event is bubbled to Stack Panel";

**e.Handled = true;**

}



* 이번에는 Tunneling Event에 대해 실습을 해보자.
* **MainWindow.xaml**

<Window x:Class="EventRoutingTest.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Events Demo" Height="215" Width="343"

**PreviewMouseDown="Window\_PreviewMouseDown">**

<StackPanel Height="120" HorizontalAlignment="Center"

VerticalAlignment="Center" Width="200"

Background="Transparent"

PreviewMouseDown="StackPanel\_PreviewMouseDown">

<Border Margin="20" Padding="5" Background="LightYellow"

BorderBrush="SteelBlue" BorderThickness="3,5,3,5" CornerRadius="3"

VerticalAlignment="Top">

<Button Margin ="20" Height="50" Width="50"

PreviewMouseDown="Button\_PreviewMouseDown"/>

</Border>

</StackPanel></Window>

* **MainWindow.xaml.cs**

using System.Windows;

namespace EventRoutingTest

{

public partial class MainWindow : Window

{

string mouseActivity = string.Empty;

public MainWindow()

{

InitializeComponent();

}

private void Button\_PreviewMouseDown(object sender, RoutedEventArgs e)

{

mouseActivity = "PreviewMouseDown Button \n";

MessageBox.Show(mouseActivity);

}

private void StackPanel\_PreviewMouseDown(object sender, RoutedEventArgs e)

{

mouseActivity = "PreviewMouseDown StackPanel \n";

MessageBox.Show(mouseActivity);

}

private void Window\_PreviewMouseDown(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

mouseActivity = "PreviewMouseDown Window \n";

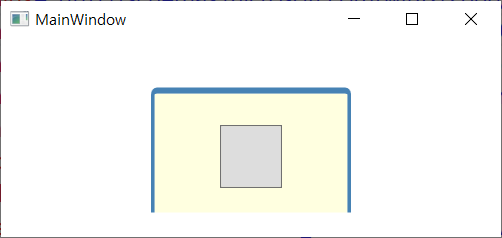
MessageBox.Show(mouseActivity);

}

}

}

* 실행화면



* Window -> StackPanel -> Button의 구조를 가지며 터널링 이벤트는 이벤트명 앞에 Peview가 붙습니다.
* 만약 버튼에서 클릭하면 Window -> StackPanel -> Button의 PreviewMouseDown 이벤트가 호출되고 StackPanel에서 클릭하면 Window -> StackPanel의 PreviewMouseDown 이벤트가 호출됩니다. 즉 최상위 객체에서 클릭한 객체까지 이벤트가 내려가면서 라우팅 되는 것입니다.