

C# WinformApp

Windows Presentation Foundation



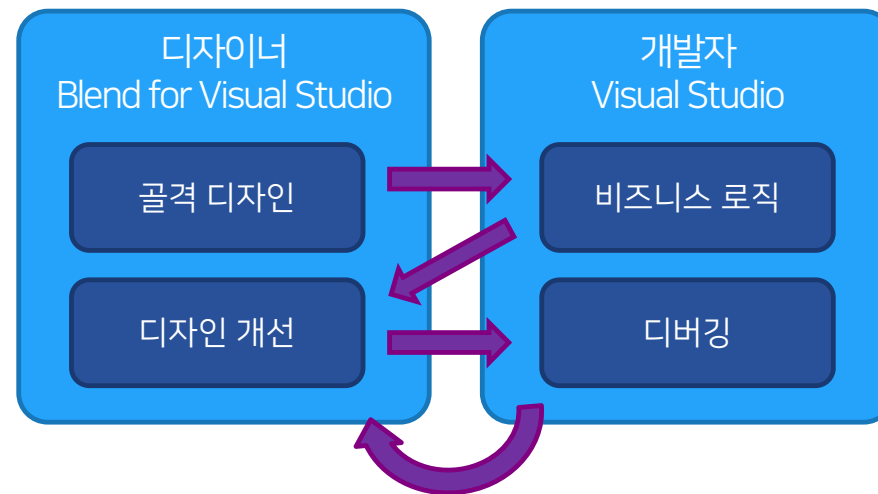
애플리케이션 생성 목차

- 개발자-디자이너 워크플로우
- 컨트롤 추가 방법
- 단순 컨트롤
- 검색
- 프로젝트 시작
- XAML 이론
- 반복
- MVVM 패턴

개발자-디자이너 워크플로우

1. WPF 작업 순서

- A. 개발자와 디자이너는 자신의 개발툴을 가지고 서로간에 작업을 공유할 수 있음
- B. 디자이너의 작업이 개발자의 비즈니스 코드에 영향을 미치지 않을 수 있음

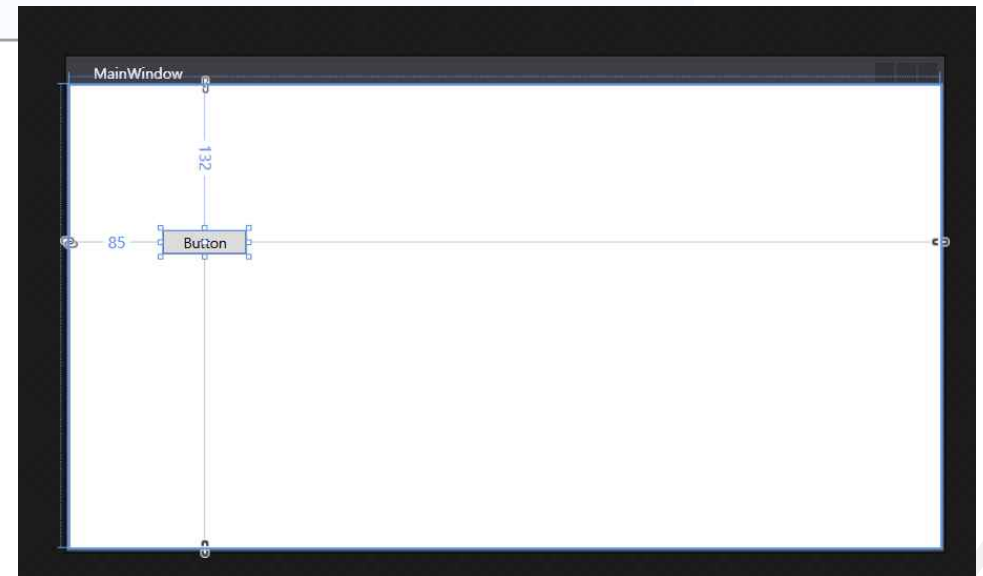


컨트롤 추가 방법

1. XAML 화면에 컨트롤을 추가하는 방법

- A. 도구상자에서 컨트롤 드래그
- B. XML 요소를 코딩하여 추가

```
...  
    <Grid>  
        <Button x:Name="BtnConnect" Content="Connect" Width="100" Height="30" />  
    </Grid>
```



단순 컨트롤

1. 기본 컨트롤

A. 텍스트 및 화면 디스플레이 관련

```
<TextBlock Text="TextBlock" Width="100" />  
<TextBox Text="TextBox" Width="100" />  
<ProgressBar Value="50" Width="100" Height="20" />  
<Slider Value="5" Width="100" />  
<PasswordBox PasswordChar="*" Password="Secret" Width="100" />
```

TextBlock

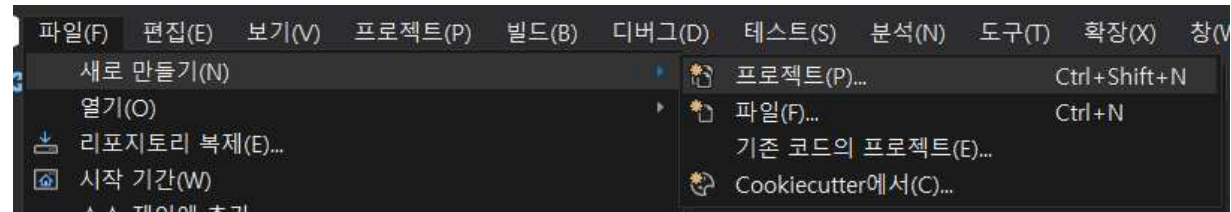
TextBox



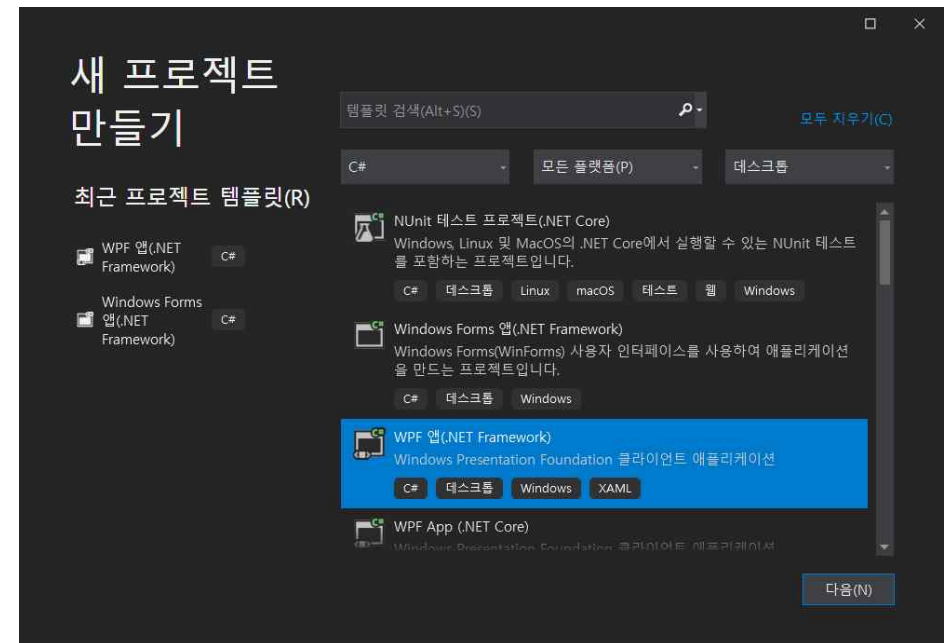
단순 컨트롤

1. 기본 컨트롤 사용

A. 프로젝트 생성



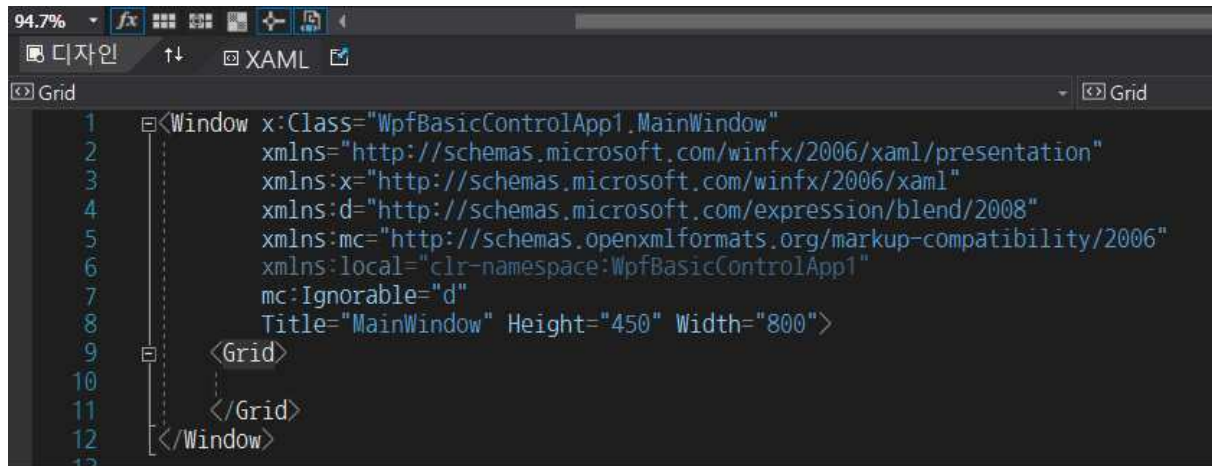
B. 새 프로젝트 만들기에서 WPF 앱(.NET Framework) 선택



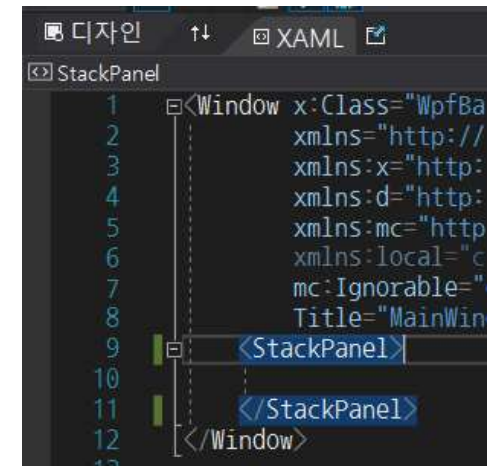
단순 컨트롤

1. 기본 컨트롤 사용

- A. 프로젝트 명 WpfBasicControlApp1 입력 후 확인
- B. 프로젝트 생성 완료 후 MainWindow.xaml 클릭
- C. Grid 태그를 StackPanel로 변경



```
1 <Window x:Class="WpfBasicControlApp1.MainWindow"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5       xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6       xmlns:local="clr-namespace:WpfBasicControlApp1"
7       mc:Ignorable="d"
8       Title="MainWindow" Height="450" Width="800">
9     <Grid>
10    </Grid>
11 </Window>
```

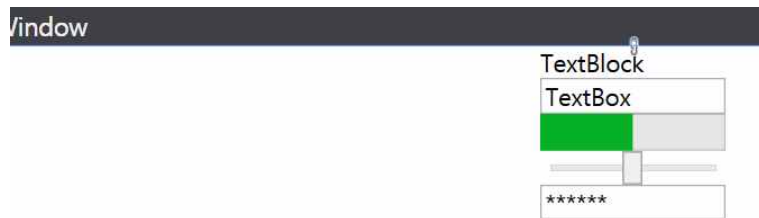


```
1 <Window x:Class="WpfBa
2       xmlns="http://
3       xmlns:x="http:
4       xmlns:d="http:
5       xmlns:mc="http
6       xmlns:local="c
7       mc:Ignorable="
8       Title="MainWin
9     <StackPanel>
10    </StackPanel>
11 </Window>
```

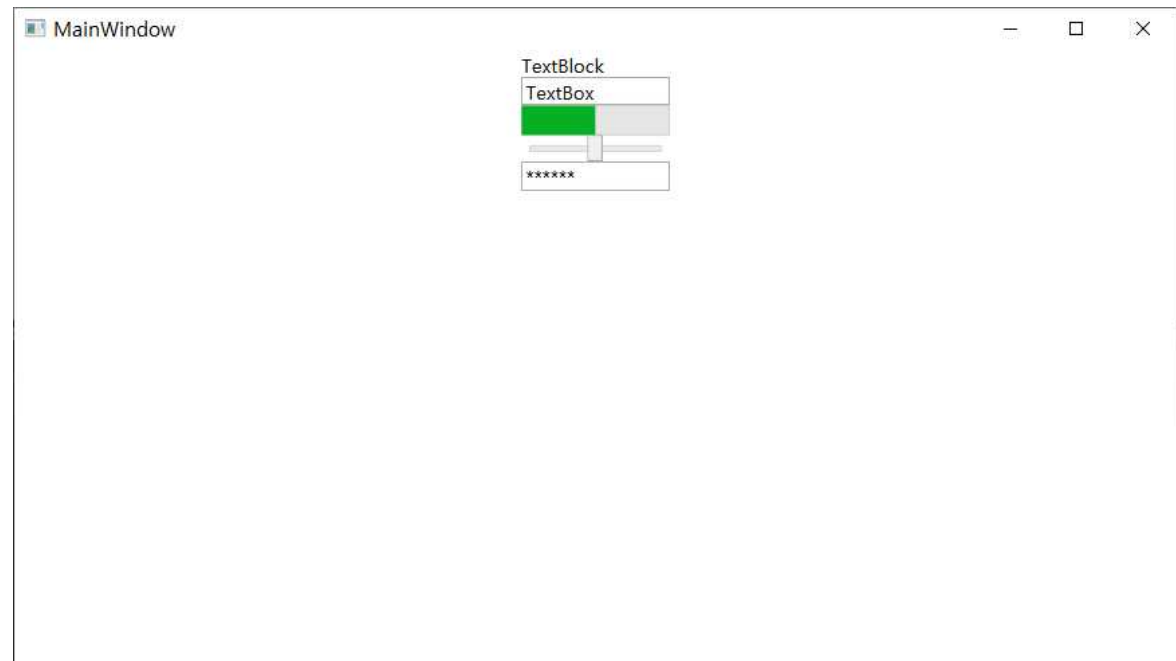
단순 컨트롤

1. 기본 컨트롤 사용

- A. 5페이지 컨트롤 입력 후
- B. 솔루션 빌드
- C. 실행 Ctrl + F5



```
XAML
<Title="MainWindow" Height="450" Width="800">
<StackPanel>
  <TextBlock Text="TextBlock" Width="100" />
  <TextBox Text="TextBox" Width="100" />
  <ProgressBar Value="50" Width="100" Height="20" />
  <Slider Value="5" Width="100" />
  <PasswordBox PasswordChar="*" Password="Secret" Width="100" />
</StackPanel>
```



단순 컨트롤

1. 멀티미디어 컨트롤

A. 이미지 및 동영상 관련

- a. 공통 크기 조절 - 컨트롤에 할당된 크기에 맞춰 조절(Stretch)
- b. Uniform(default) - 측면에 투명 여백을 남겨두고 이미지의 크기 비례해서 조절
- c. Fill - 이미지 비례적으로 크기 조절, Image 컨트롤에 할당된 전체 공간 채움

```
<Image Source="cat.jpg" Height="100" />  
<MediaElement Source="cat.wmv" Height="150" LoadedBehavior="Play" />
```



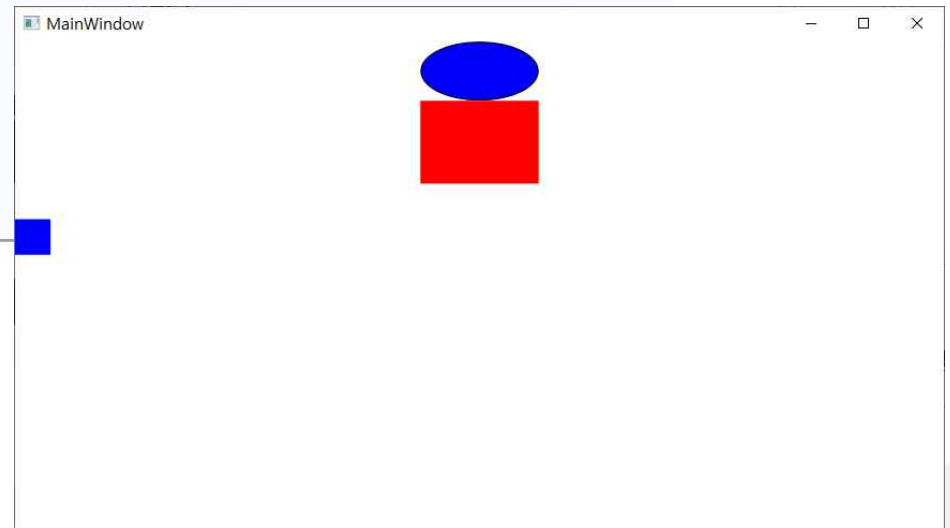
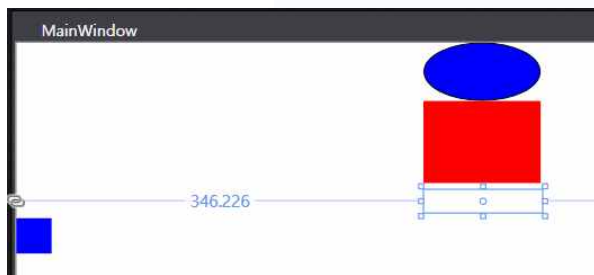
단순 컨트롤

1. 그리기 컨트롤

A. 타원, 사각형 등 벡터 이미지 컨트롤의 공통 속성

- a. Fill - 컨트롤 내부 칠하기 브러시
- b. Stroke - 컨트롤 윤곽 그리기 브러시
- c. Stretch - 크기 조절시 컨트롤의 모양크기 조절 방식

```
<Ellipse Width="100" Height="50" Fill="Blue" Stroke="Black" />  
<Rectangle Width="100" Height="70" Fill="Red" />  
<Path Width="100" Height="20" Margin="5, 5, 5, 5" />  
<Path Fill="Blue">  
  <Path.Data>  
    <RectangleGeometry Rect="0, 0, 30, 30"/>  
  </Path.Data>  
</Path>
```



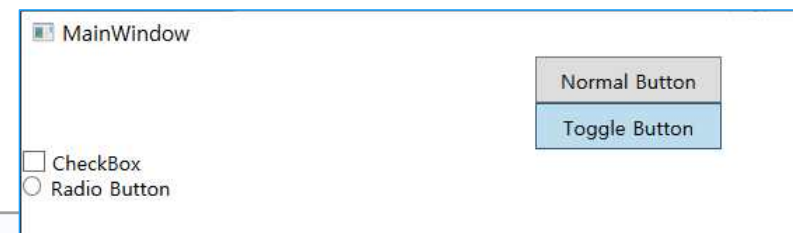
단순 컨트롤

1. 콘텐츠 컨트롤

A. Content 속성을 가진 모든 컨트롤

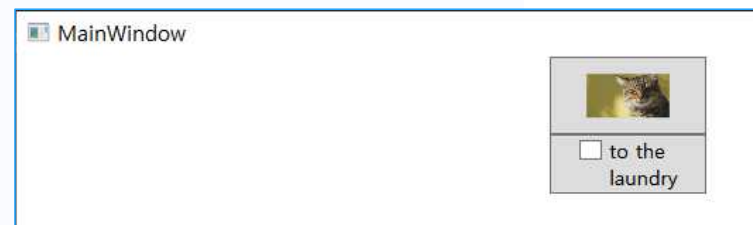
a. Button, Border, ScrollViewer, ViewBox

```
<Button Content="Normal Button" Width="120" Height="30" />
<ToggleButton Content="Toggle Button" Width="120" Height="30" />
<CheckBox Content="CheckBox" />
<RadioButton Content="Radio Button" />
```



B. 복잡한 콘텐츠를 사용하고자 하면 하위요소를 사용할 수 있음

```
<Button Padding="10" Height="50" Width="100">
  <MediaElement Source="cat.wmv" Height="30" />
</Button>
<Button Width="100">
  <CheckBox>
    <TextBlock Text="to the laundry"
               TextWrapping="Wrap" />
  </CheckBox>
</Button>
```



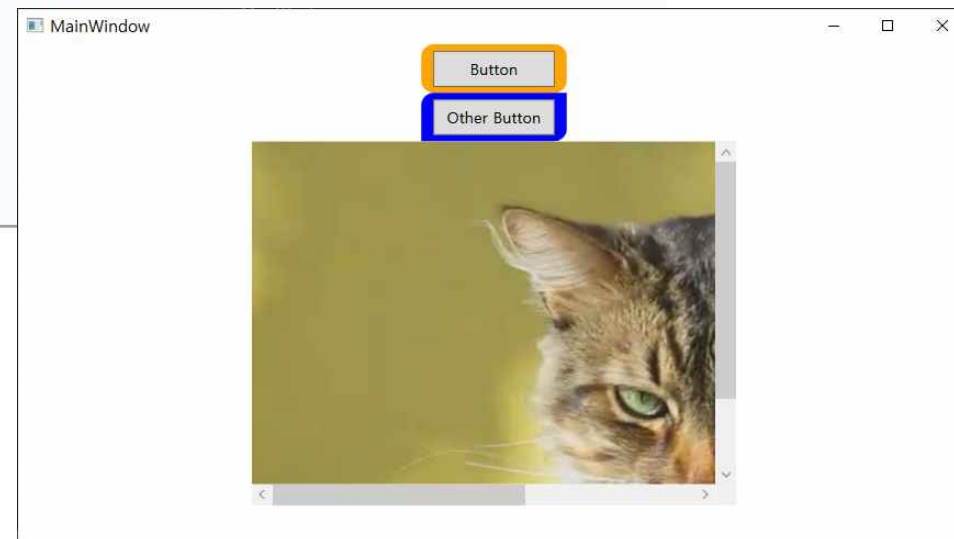
단순 컨트롤

1. 콘텐츠 컨트롤

A. 원품과는 다르게 컨트롤 외관을 쉽게 변형 가능(원품이 불가능한 것은 아님)

```
<Border Background="Orange" CornerRadius="10" Padding="5" Width="120" >
    <Button Content="Button" Width="100" Height="30" />
</Border>
<Border Background="Blue" CornerRadius="10,0,10,0" Padding="5" Width="120" >
    <Button Content="Other Button" Width="100" Height="30" />
</Border>

<ScrollViewer Height="300" Width="400"
    HorizontalScrollBarVisibility="Auto">
    <MediaElement Source="cat.wmv" Stretch="None" />
</ScrollViewer>
```



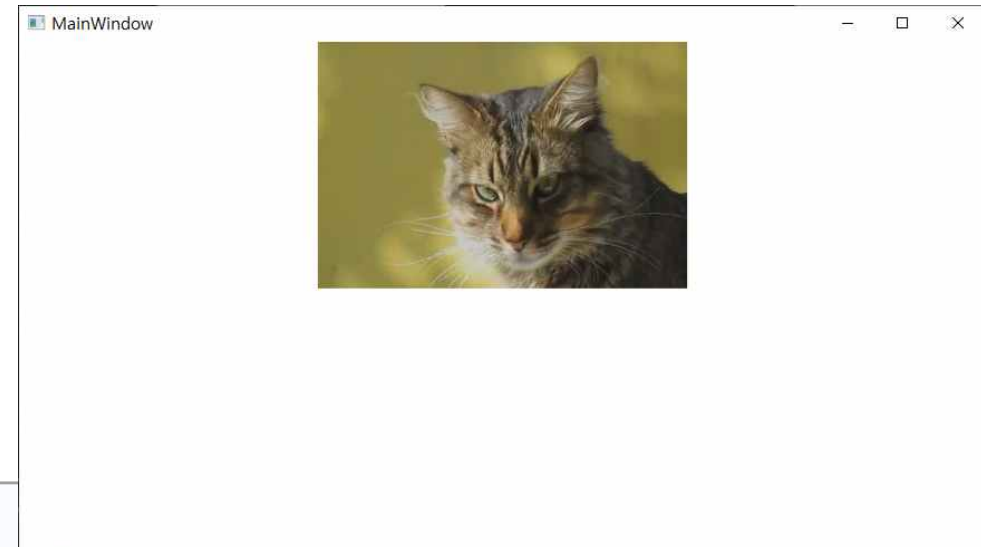
단순 컨트롤

1. 콘텐츠 컨트롤

A. ViewBox

- a. 모든 콘텐츠의 크기 조정가능
- b. 사용 가능한 너비와 높이에 맞게 화면을 조정

```
<Viewbox Stretch="Fill" Width="300" Height="200"  
  ScrollViewer.VerticalScrollBarVisibility="Visible">  
  <MediaElement Source="cat.wmv" Stretch="None" />  
</Viewbox>
```



단순 컨트롤

1. 콘텐츠 컨트롤

A. ViewBox

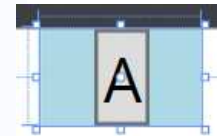
a. 뷰박스 없는 그리드

```
<Grid Height="60" Width="100" Background="LightBlue">  
  <Button Content="A" />  
</Grid>
```



b. 뷰박스 그리드

```
<Grid Height="60" Width="100" Background="LightBlue">  
  <Viewbox>  
    <Button Content="A" />  
  </Viewbox>  
</Grid>
```



c. 변형

```
<Grid Height="60" Width="100" Background="LightBlue">  
  <Viewbox Stretch="Fill">  
    <Button Content="A" />  
  </Viewbox>  
</Grid>
```

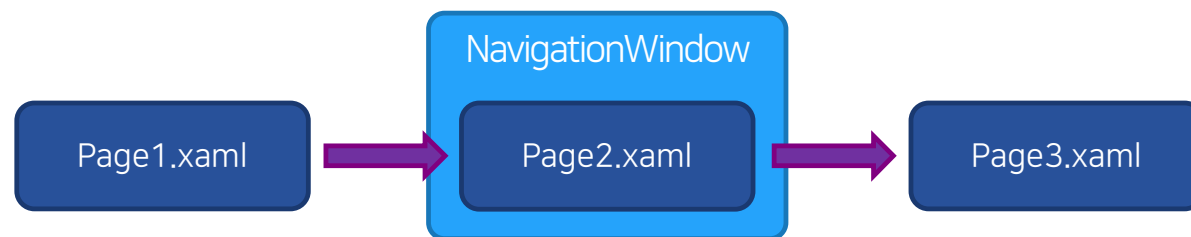


탐색

1. 개요

A. Navigation 사용

- a. 이전 화면 되돌아가기 미치 기록에서 되돌아가기
- b. WPF 내 NavigationWindow 기능을 이용



- c. 코드비하인드 및 xaml 상에서 모두 구현 가능

프로젝트 시작

1. 개요

- A. 작은 전자상거래 애플리케이션 구현 - BikeShop
 - a. 사용자가 제품을 검색, 장바구니 추가
 - b. 사이트 관리자가 제품을 나열, 생성, 수정 및 삭제 가능

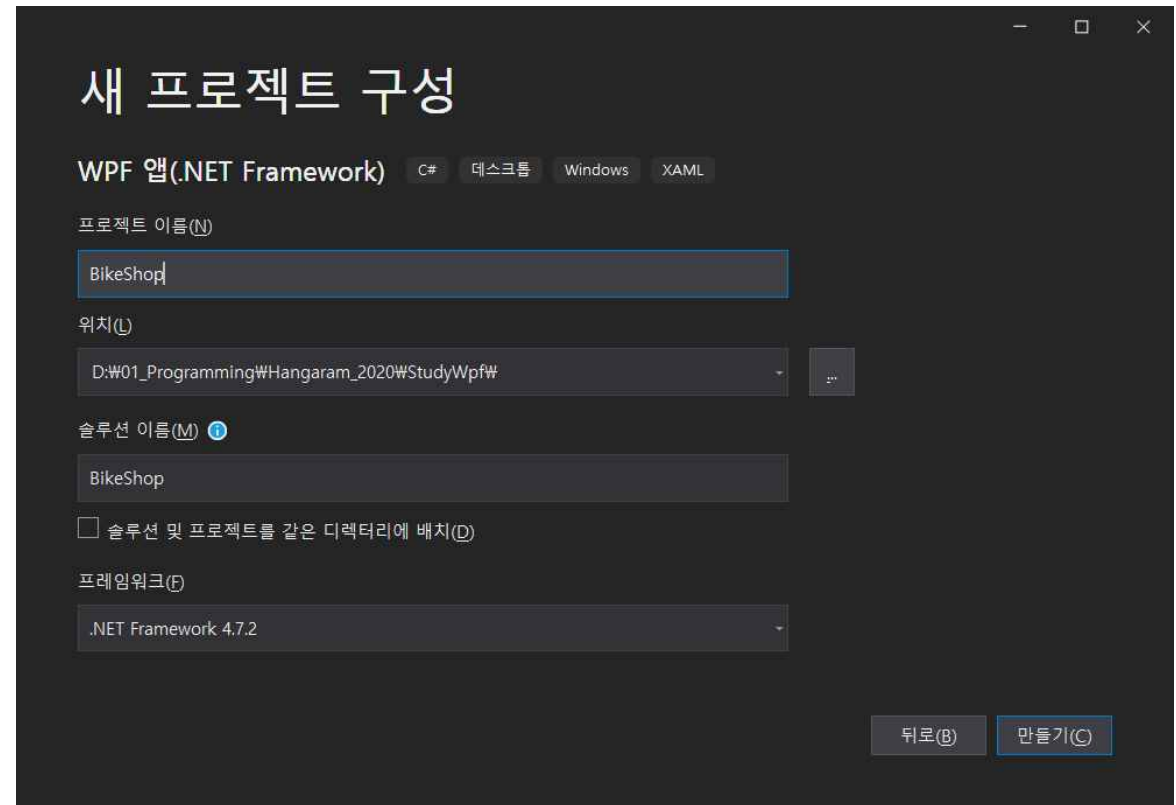
프로젝트 시작

1. Tutorial 1

A. 프로젝트 생성

a. 새 프로젝트 구성

- 1) 프로젝트 이름 - BikeShop
- 2) 위치 - 자신의 필요한 곳
- 3) 솔루션 이름 - BikeShop
- 4) 프레임워크 - .NET Framework 4.7.x

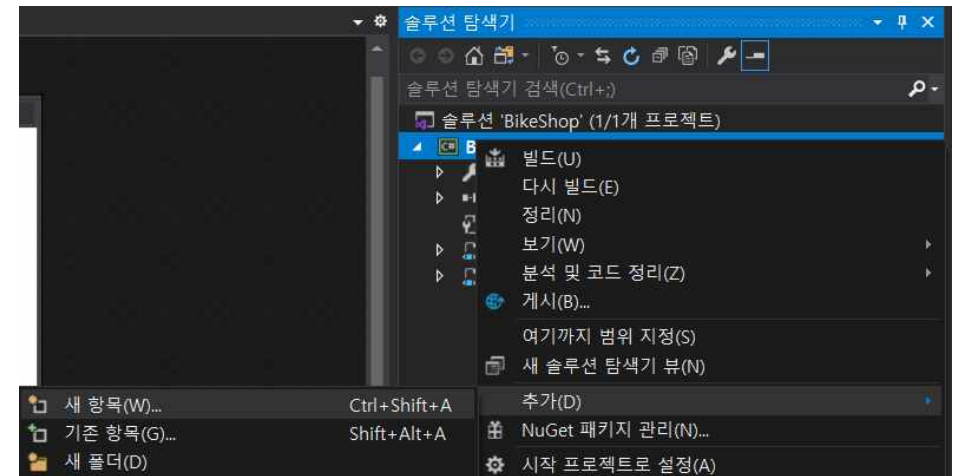


프로젝트 시작

1. Tutorial 1

A. 솔루션 탐색기

- a. 화면에 없으면 보기 → 솔루션 탐색기 (Ctrl + W, S)
- b. 컨텍스트 메뉴(마우스 오른쪽) → 추가 → 새 항목
 - 1) Ctrl + Shift + A 사용 추천
 - 2) 왼쪽 메뉴에서 WPF 선택
 - 3) 리스트에서 페이지(WPF) 선택
 - 4) 이름 Contact.xaml 입력 후 추가 버튼 클릭

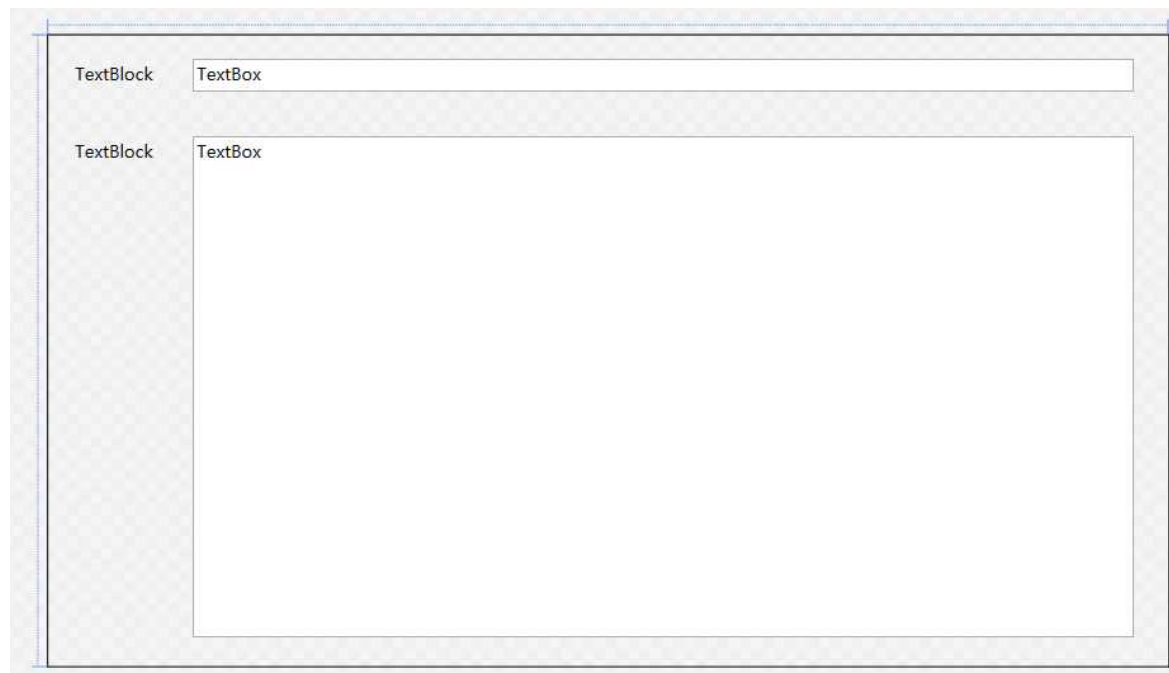
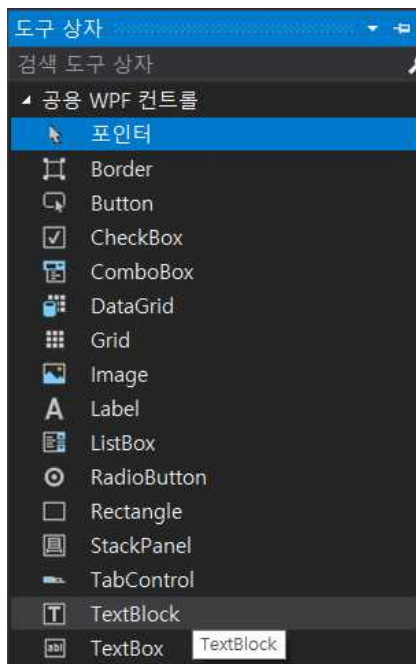


프로젝트 시작

1. Tutorial 1

A. 화면 구성

- TextBlock 2개, TextBox 2개로 화면 구성
- 도구상자 오픈, 공용 WPF 컨트롤에서 선택 드래그

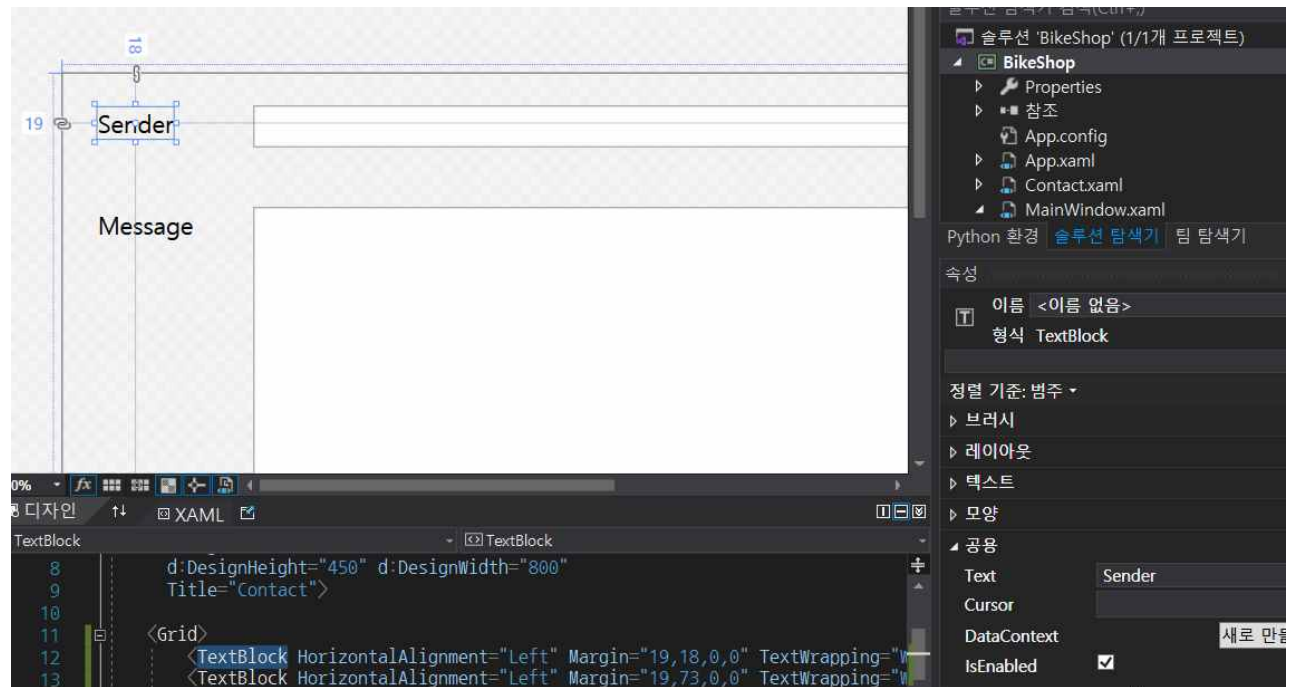


프로젝트 시작

1. Tutorial 1

A. 화면 구성

- TextBlock 클릭 후
- 속성, Text → Sender로 변경 엔터
- 두번째 TextBlock → Message로 변경
- TextBox 클릭 후
- 속성, Text 삭제



프로젝트 시작

1. Tutorial 1

A. 화면 구성

- a. 솔루션 탐색기 MainWindow.xml 파일 더블 클릭
- b. Xaml 코드에 추가
 - 1) Source 입력 후 속성창에서 선택 추천
 - 2) NavigationUIVisibility = Hidden 선택 (옵션)

```
<Grid>  
    <Frame x:Name="MainFrame" Source="/BikeShop;component/Contact.xaml" NavigationUIVisibility="Hidden" />  
</Grid>
```

c. 컴파일 후 실행

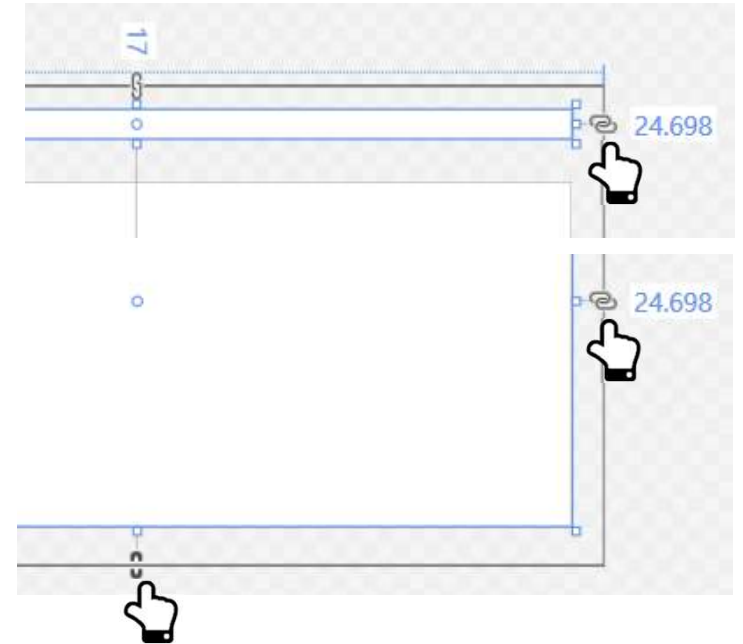


프로젝트 시작

1. Tutorial 1

A. 화면 구성

- a. 종료 후 Contact.xaml 열기
 - 1) 첫번째 TextBox 컨트롤 우측 자동 크기 조절 아이콘 클릭
 - 2) 두번째 TextBox 컨트롤 우측 및 하단 자동 크기 조절 아이콘 클릭
- b. 재실행 후 확인



XAML 이론

1. XAML 네임스페이스 이해

A. MainWindow.xaml 소스 코드 해석

a. 소스

```
<Window x:Class="BikeShop.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:BikeShop"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Frame x:Name="MainFrame" Source="/BikeShop;component/Contact.xaml" NavigationUIVisibility="Hidden" />
    </Grid>
```

b. xmlns - C# 코드의 using 키워드와 같이 XAML 네임스페이스 사용 선언 (URL)

- 1) 특성이 추가되면 이 요소와 해당 하위 요소에 기본적으로 URL이 접두어로 추가된다는 것을 의미

```
http://schemas.../presentation:Frame.
```

XAML 이론

1. XAML 네임스페이스 이해

A. MainWindow.xaml 소스 코드 해석

a. x:Name의 의미

```
Http://schemas....xaml:Name.
```

2. 객체 생성

A. 객체 생성 방법

a. C# 에서의 객체 생성

```
new Car();
```

b. XAML에서 객체 생성

```
<Car />
```

1) 이는 에러 발생 할 수 밖에 없음. WPF xml 네임스페이스 내에 Car에 대한 정의가 없기 때문

XAML 이론

1. 객체 생성

A. 계속

a. Car 클래스 선언

```
using System.Windows.Media;

namespace BusinessLogic
{
    public class Car
    {
        public double Speed { get; set; }
        public Color Color { get; set; }
    }
}
```

b. C# 객체 생성

```
using BusinessLogic;
new Car();
```

c. XAML에서 객체 생성

```
<Label xmlns:bl="clr-namespace:BusinessLogic"> <bl:Car /> </Label>
```

XAML 이론

1. 속성 정의

A. 생성한 객체의 속성 값 할당

a. C#

```
var c = new Car();  
c.Speed = 100;  
c.Color = Colors.Red;
```

b. XAML

```
<Label xmlns:bl="clr-namespace:BusinessLogic">  
  <bl:Car Speed="100" Color="Red" />  
</Label>
```

XAML 이론

1. 속성 정의

A. 확장

a. C#

```
public class Human
{
    public string FirstName { get; set; }
    public bool HasDrivingLicense { get; set; }
}

public class Car
{
    public double Speed { get; set; }
    public Color Color { get; set; }
    public Human Driver { get; set; }
}
```

XAML 이론

1. 속성 정의

A. 확장

a. 두 코드는 동일한 의미

```
var h = new Human();  
h.FirstName = "Nick";  
h.HasDrivingLicense = true;
```

```
var c = new Car();  
c.Speed = 100;  
c.Color = Colors.Red;  
c.Driver = h;
```

b. XAML

```
<Label xmlns:bl="clr-namespace:BusinessLogic">  
  <bl:Car Speed="100" Color="Red">  
    <bl:Car.Driver>  
      <bl:Human FirstName="Nick" HasDrivingLicense="True" />  
    </bl:Car.Driver>  
  </bl:Car>  
</Label>
```

XAML 이론

1. 명명 규칙

A. XAML 요소를 참조할 때

a. x:Name 특성 추가 in XAML

```
xmlns:bl="clr-namespace:BusinessLogic "  
mc:Ignorable="d"  
d:DesignHeight="450" d:DesignWidth="800"  
Title="Test">  
<Grid>  
  <Label>  
    <bl:Car x:Name="myCar" Speed="100" Color="Red" />  
  </Label>  
</Grid>
```

b. C#에서 사용

```
private void Inits()  
{  
    myCar.Color = Colors.Blue;  
}
```

XAML 이론

1. 이벤트

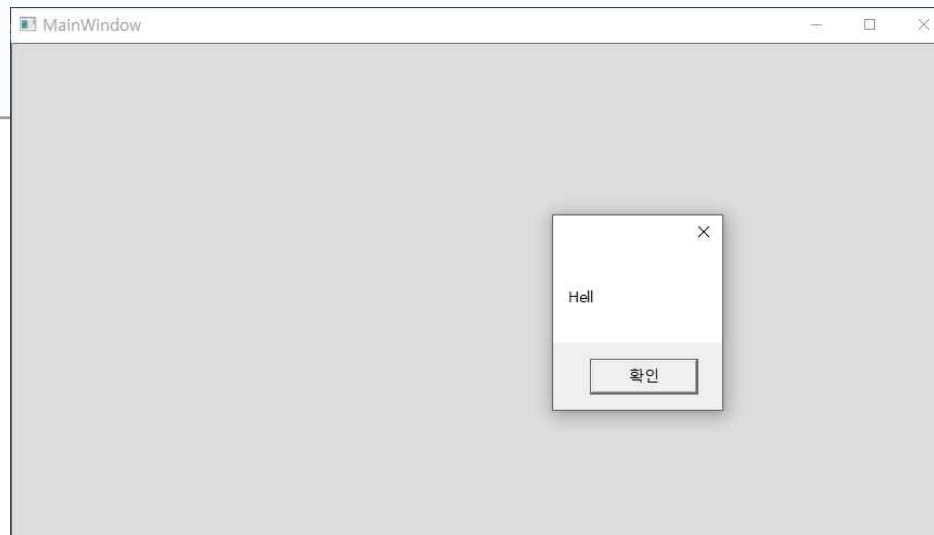
A. 코드 비하인드 메서드와 연계

a. xaml

```
<Button Click="Greet" />
```

b. CS

```
private void Greet(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hell");
}
```

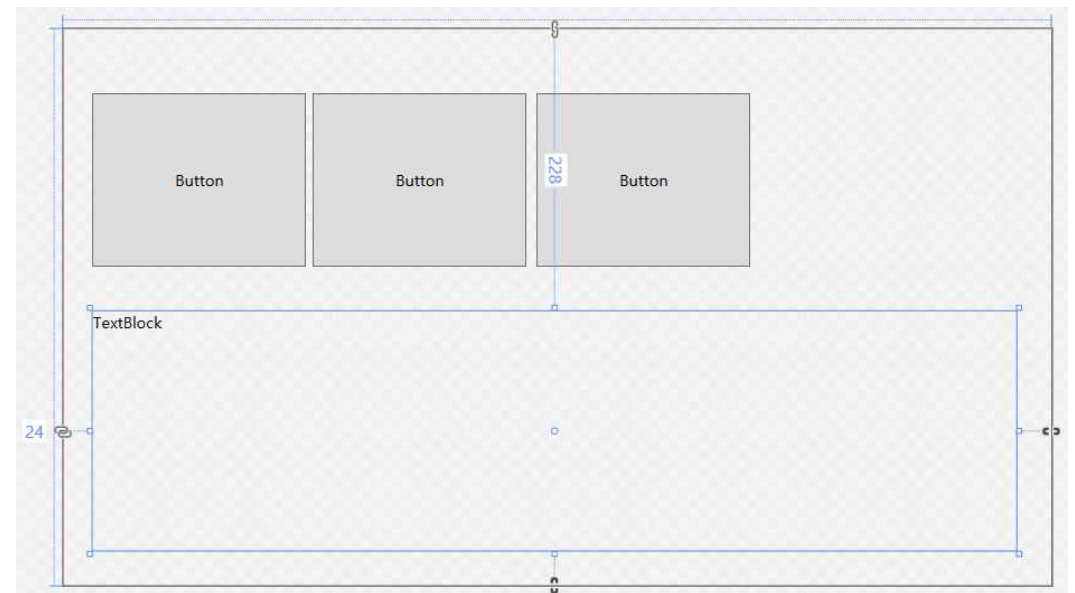


프로젝트 계속

1. Tutorial 2

A. 메뉴 페이지 생성

- Menu.xaml 새 페이지 추가, MainWindow.xaml 화면에 기본으로 표시
- 솔루션 탐색기에서 페이지 추가
- Button 컨트롤 세 개와 TextBlock 컨트롤 하나 구성



프로젝트 계속

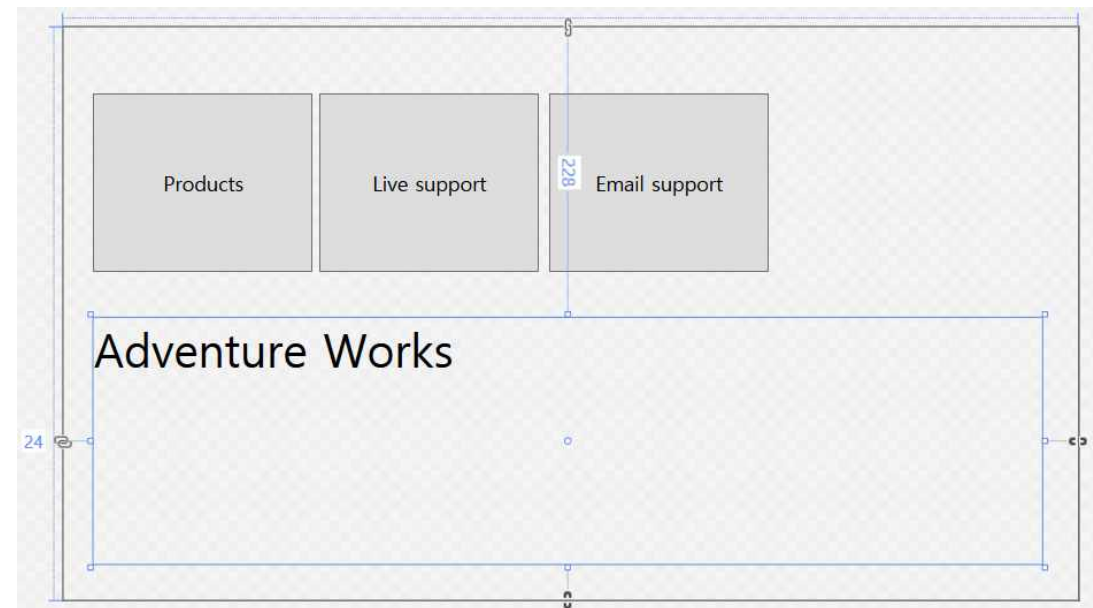
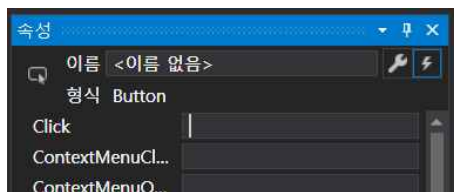
1. Tutorial 2

A. 메뉴 페이지 생성

- 첫번째 버튼 컨트롤 클릭, Content → Products 변경, 두번째 버튼 Live support, 세번째 버튼 Email support
- 텍스트 속성의 텍스트 사이즈, 16 px로 변경
- TextBlock 컨트롤을 클릭하고 Text 속성값 Adventure Works 로 변경, 텍스트 크기 36 px로 변경

B. 버튼 이벤트 추가

- Email support 버튼 클릭 특성 추가, 속성 사용
 - 속성, 선택한 요소의 이벤트 처리기 클릭
 - Click 속성의 텍스트 박스 더블클릭(추천)



프로젝트 계속

1. Tutorial 2

A. 버튼 이벤트 추가(계속)

a. Email support 버튼 이벤트 작성

1) 생성된 이벤트 처리기 코드에 NavigationService 메서드 작성

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    NavigationService.Navigate(
        new Uri("/Contact.xaml", UriKind.RelativeOrAbsolute)
    );
}
```

B. MainWindow.xaml 수정

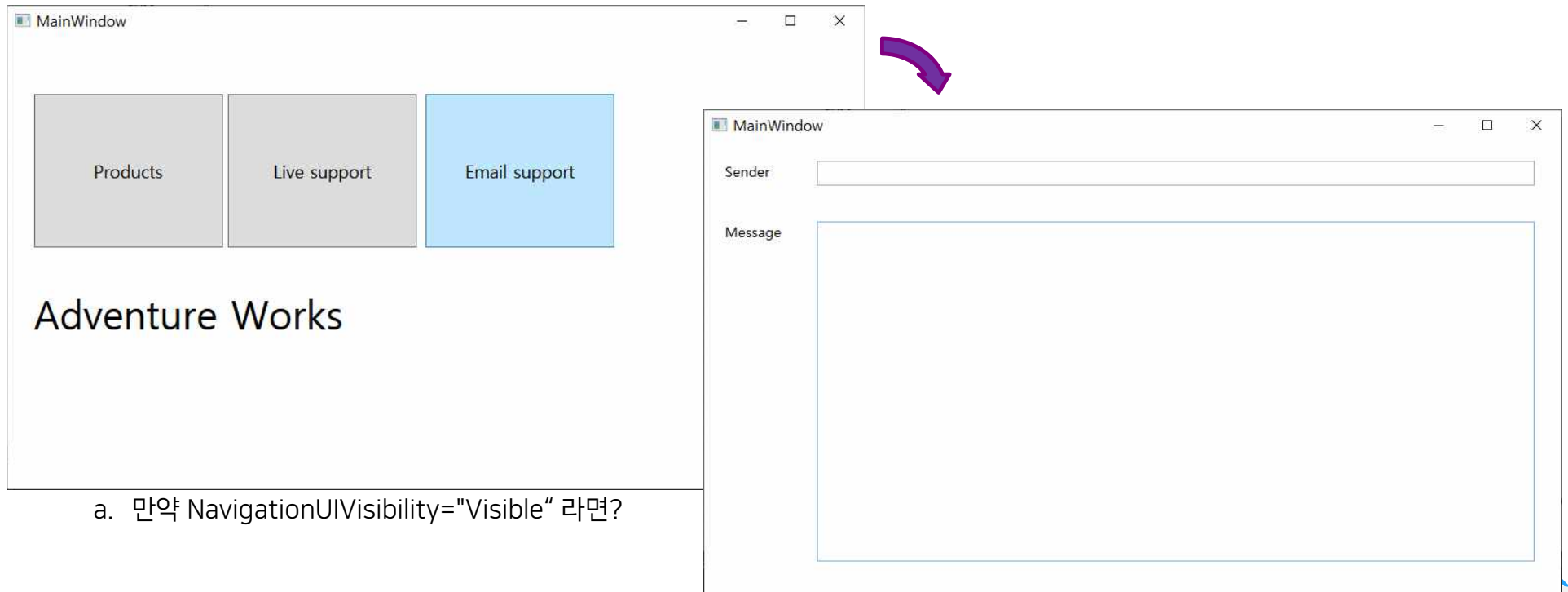
a. 기본 페이지 변경

```
<Grid>
    <Frame x:Name="MainFrame" Source="/Menu.xaml" NavigationUIVisibility="Hidden" />
</Grid>
```

프로젝트 계속

1. Tutorial 2

A. 실행 결과



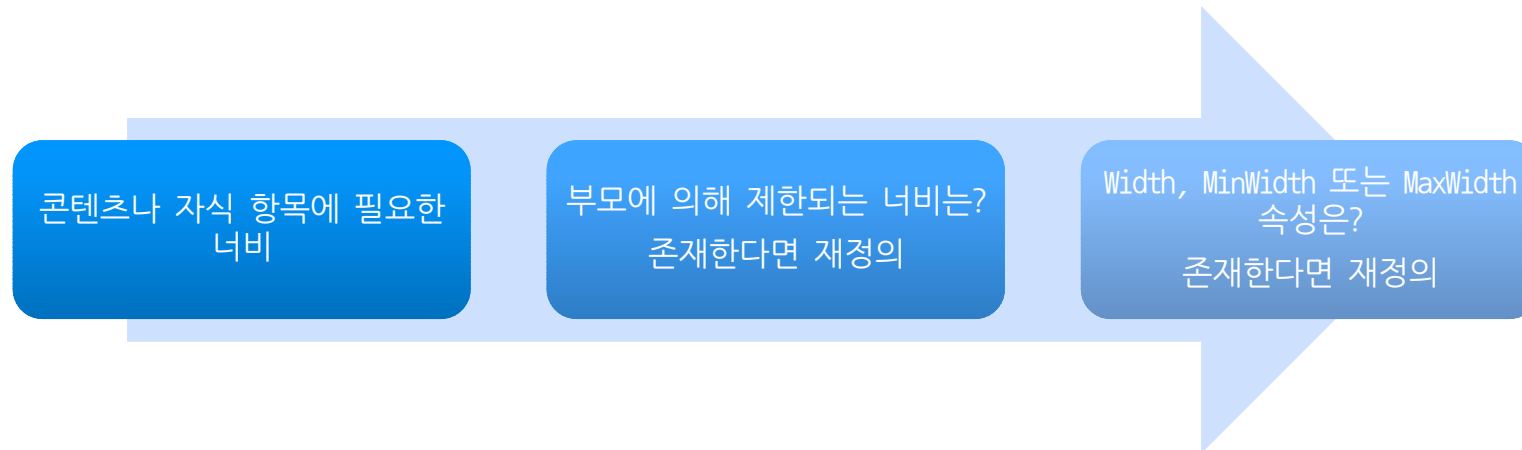
XAML 이론

1. 레이아웃

- A. 화면 크기 조절이 안되는 이유 (메뉴 페이지)
 - a. 애플리케이션 창 크기를 조절했을 때 컨트롤의 크기가 같이 변하지 않음
 - b. 창이 컨트롤들 구성된 크기보다 작아지면 컨트롤들의 일부가 가려짐
 - c. 컨트롤이 메뉴 페이지 중심에 있었으면...
- B. Panel을 사용하자!

2. 크기 할당

- A. 컨트롤의 최종 너비를 계산하기 위해 아래의 과정 진행



XAML 이론

1. 크기 할당

A. Canvas를 쓸 경우

```
<Canvas Width="50" Height="50" Background="Orange">  
    <Button Content="Hell World" Margin="5" />  
</Canvas>
```



B. Grid를 쓸 경우

```
<Grid Width="50" Height="50" Background="Orange">  
    <Button Content="Hell World" Margin="5" />  
</Grid>
```



C. 결론

- a. Canvas 컨트롤의 경우, 자식 컨트롤 크기를 제한하지 않음
- b. Grid 컨트롤은 자식의 크기를 제한함

XAML 이론

1. 크기 할당

A. Panel 컨트롤

- a. 하나의 컨트롤만 허용하는 여러 개의 컨트롤을 표시
- b. 사용 가능한 크기에 따라 컨트롤을 배치
- c. 애플리케이션 화면을 사용 가능한 화면 영역에 맞춰 조정할 수 있으므로 광범위한 의미로 반응형 디자인(Response Design)이라고 할 수 있음

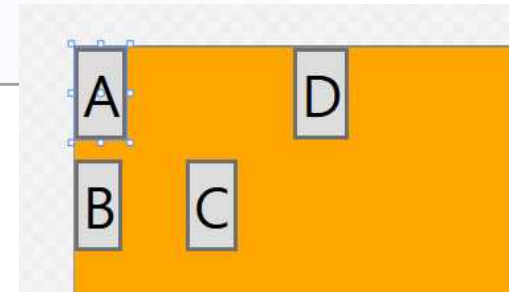
XAML 이론

1. 주요 컨트롤

A. Canvas 패널 - 자체 좌표를 제공하는 컨트롤 배치

```
<Canvas Background="Orange">  
  <Button Canvas.Top="0" Canvas.Left="0" Content="A"/>  
  <Button Canvas.Top="25" Canvas.Left="0" Content="B"/>  
  <Button Canvas.Top="25" Canvas.Left="25" Content="C"/>  
  <Button Canvas.Top="0" Canvas.Left="50" Content="D"/>  
</Canvas>
```

- a. Canvas.Left - 왼쪽 상단 기준에서부터 x 값
- b. Canvas.Top - 왼쪽 상단 기준에서부터 y 값
- c. Width, Height 도 사용 가능



XAML 이론

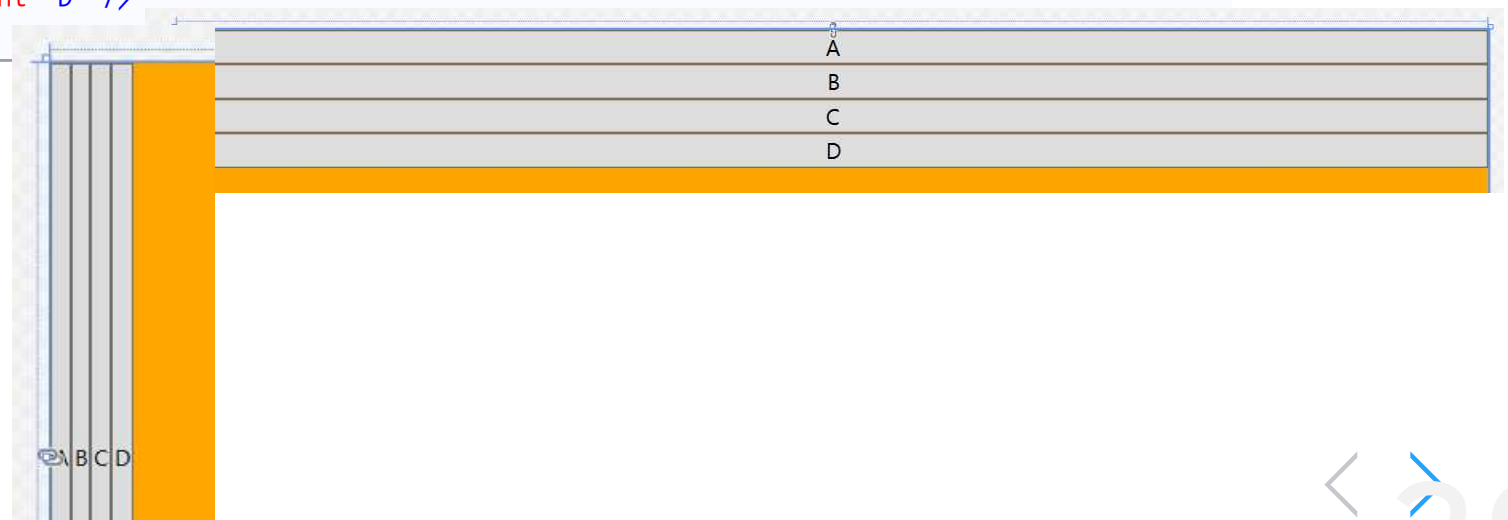
1. 주요 컨트롤

A. StackPanel - 컨트롤을 하단에서 상단으로 쌓아서 각 컨트롤에 전체 너비를 할당

a. Orientation 제어 사용 방향변경 가능

```
<StackPanel Orientation="Vertical" Background="Orange">  
    <Button Content="A" />  
    <Button Content="B" />  
    <Button Content="C" />  
    <Button Content="D" />  
</StackPanel>
```

b. Horizontal 로 변경시

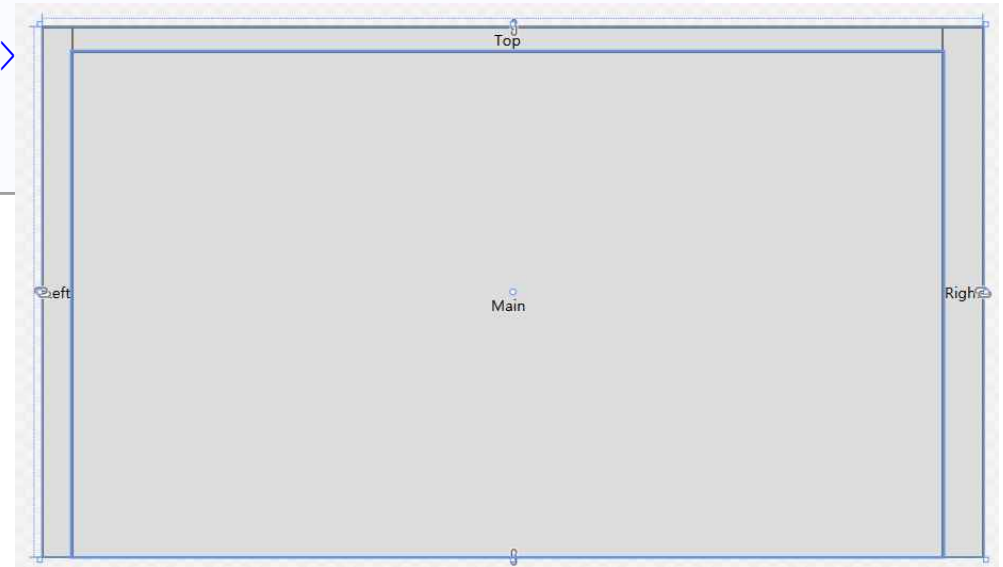


XAML 이론

1. 주요 컨트롤

- A. DockPanel - 대부분의 데스크탑 앱과 같은 화면 레이아웃을 구성
 - a. 연결 속성 추가

```
<DockPanel Background="Orange">  
    <Button Content="Left" DockPanel.Dock="Left" />  
    <Button Content="Right" DockPanel.Dock="Right" />  
    <Button Content="Top" DockPanel.Dock="Top" />  
    <Button Content="Main" />  
</DockPanel>
```

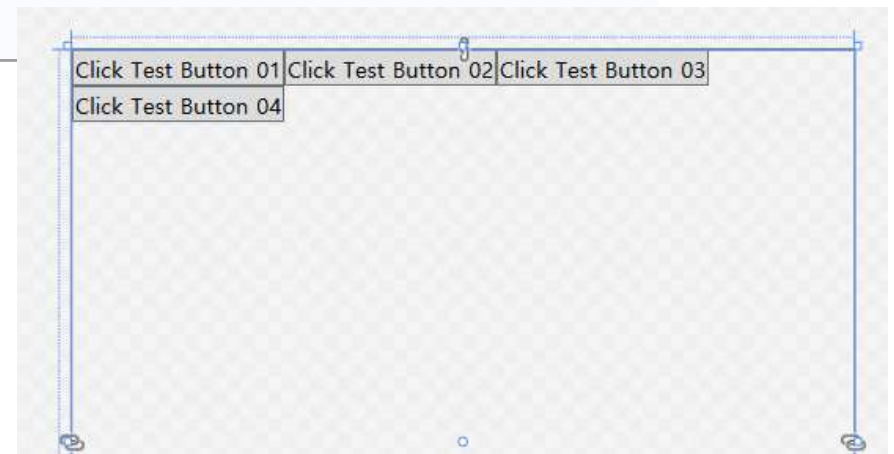


XAML 이론

1. 주요 컨트롤

- A. WrapPanel - xaml을 수동으로 편집할 때 사용 용이
 - a. 화면이 꽉차면 Wrap되는 패널

```
<WrapPanel Orientation="Horizontal">  
    <Button Content="Click Test Button 01" />  
    <Button Content="Click Test Button 02" />  
    <Button Content="Click Test Button 03" />  
    <Button Content="Click Test Button 04" />  
</WrapPanel>
```



XAML 이론

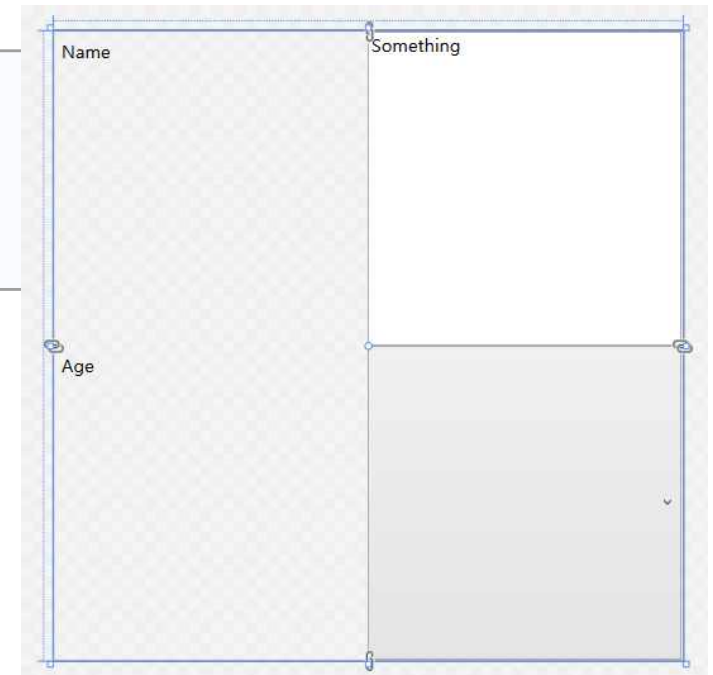
1. 주요 컨트롤

A. UniformGrid - xaml을 수동으로 편집할 때 시간을 들이지 않고 입력 UI를 배치하는 컨트롤

a. 컨트롤에 필요한 행과 열을 자동으로 계산

```
<UniformGrid>
  <Label Content="Name" />
  <TextBox Text="Something" />
  <Label Content="Age" />
  <ComboBox />
</UniformGrid>
```

b. 단점 - 모든 행과 열을 동일한 너비와 높이로 만드는 점



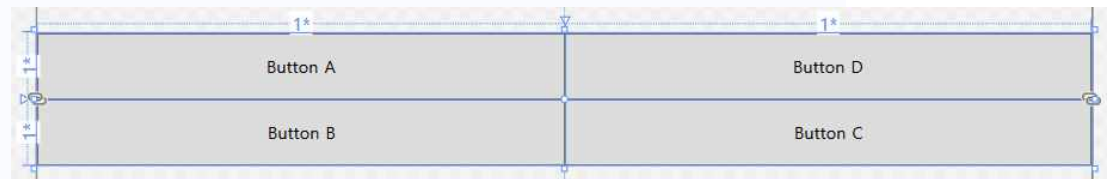
XAML 이론

1. 주요 컨트롤

- A. Grid - 레이아웃 크기를 조정할 수 있는 최상의 다기능 컨트롤
 - a. 복잡한 대신 정확하게 원하는 레이아웃을 만들 수 있음

```
<Grid Height="100">  
  <Grid.ColumnDefinitions>  
    <ColumnDefinition />  
    <ColumnDefinition />  
  </Grid.ColumnDefinitions>  
  <Grid.RowDefinitions>  
    <RowDefinition />  
    <RowDefinition />  
  </Grid.RowDefinitions>
```

```
  <Button Grid.Row="0" Grid.Column="0" Content="Button A" />  
  <Button Grid.Row="1" Grid.Column="0" Content="Button B" />  
  <Button Grid.Row="1" Grid.Column="1" Content="Button C" />  
  <Button Grid.Row="0" Grid.Column="1" Content="Button D" />  
</Grid>
```



XAML 이론

1. 주요 컨트롤

A. Grid - 레이아웃 크기를 조정할 수 있는 최상의 다기능 컨트롤

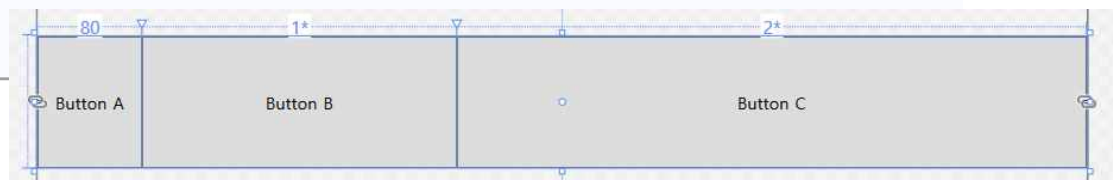
a. 레이아웃 규칙

- 1) 컨트롤은 자신이 속한 전체 칸을 채움
- 2) 각 칸에 속한 컨트롤이 각 칸의 상단에 표시
- 3) 컨트롤을 Grid.RowSpan 및 Grid.ColumnSpan 연결 속성을 사용 몇 개의 열이나 행을 채울 수 있음

b. Width / Height 규칙

- 1) 고정 숫자 : 픽셀의 수 할당
- 2) Auto : 자체 콘텐츠에 대한 크기로 적용
- 3) 별 또는 별이 붙은 숫자 : 남은 너비/높이에 비례한 크기 지정

```
<Grid.ColumnDefinitions >  
    <ColumnDefinition Width="80" />  
    <ColumnDefinition Width="*" />  
    <ColumnDefinition Width="2*" />  
</Grid.ColumnDefinitions>
```



XAML 이론

1. 패널 컨트롤 요약

A. 여섯개 컨트롤 정리

컨트롤	크기 강제	사용 편의
Canvas	No	디자인 뷰
DockPanel	Yes	XAML
Grid	Yes	디자인 뷰
StackPanel	Yes	XAML
UniformGrid	Yes	XAML
WrapPanel	Yes	XAML

XAML 이론

1. 목록 컨트롤

A. 데이터 요소 컬렉션에 바인딩 되는 기본 컨트롤

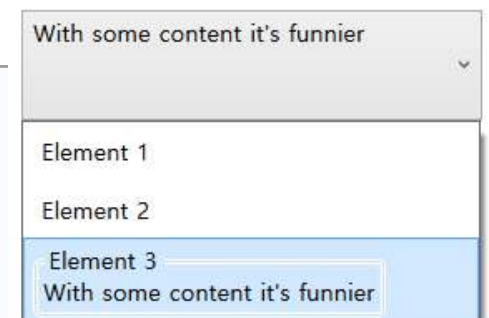
a. 선택 컨트롤 - ListBox

```
<ListBox Height="100">  
  <Label Content="Element 1" />  
  <Label Content="Element 2" />  
  <GroupBox Header="Element 3">  
    With some content it's funnier  
  </GroupBox>  
</ListBox>
```



b. 선택 컨트롤 - ComboBox

```
<ComboBox Height="60">  
  <Label Content="Element 1" />  
  <Label Content="Element 2" />  
  <GroupBox Header="Element 3">  
    With some content it's funnier  
  </GroupBox>  
</ComboBox>
```



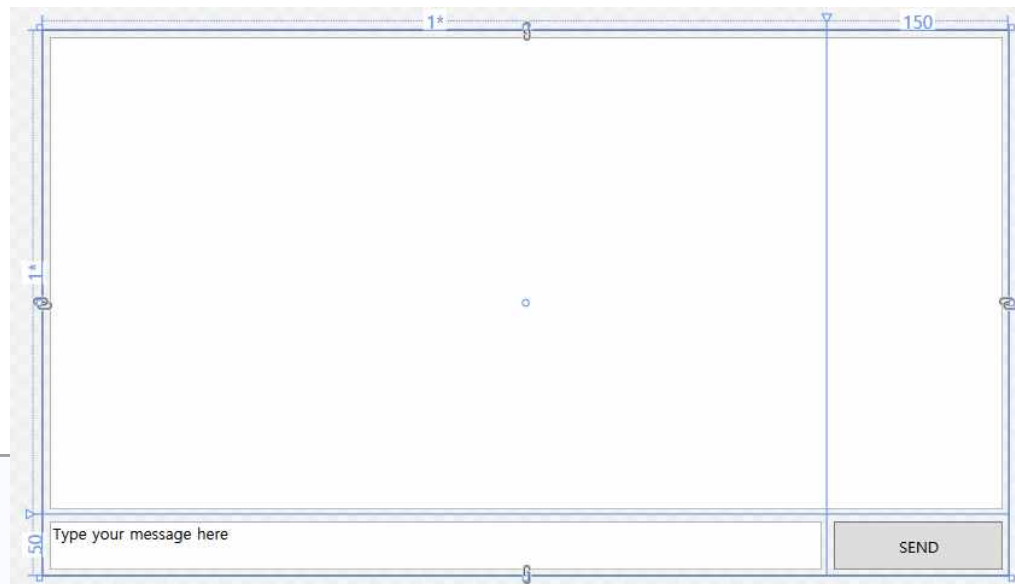
프로젝트 계속

1. Tutorial 3

A. 토론 페이지 생성

- Discussion.xaml 페이지 추가
- 파일 오픈 후 코딩 추가

```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="150" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="50" />
</Grid.RowDefinitions>
<ListBox Grid.ColumnSpan="2" Margin="5" />
<TextBox Grid.Row="1" Grid.Column="0" Margin="5" Text="Type your message here" />
<Button Grid.Row="1" Grid.Column="1" Margin="5" Content="SEND" />
```

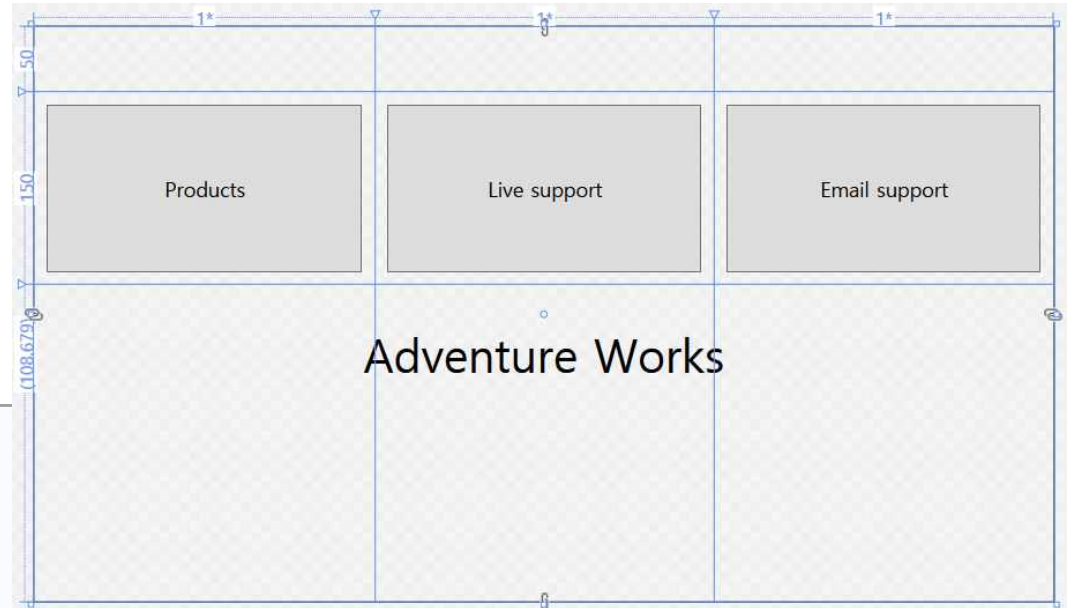


프로젝트 계속

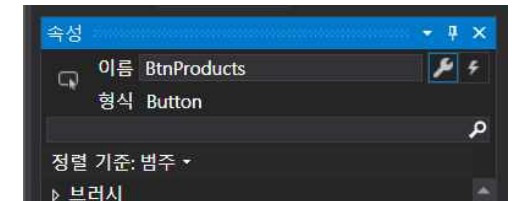
1. Tutorial 3

- A. 토론 페이지 생성
 - a. Menu.xaml 오픈
 - b. xaml 코드 수정

```
<Grid.RowDefinitions>
    <RowDefinition Height="50" />
    <RowDefinition Height="150" />
    <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Button Content="Products" Grid.Row="1" Grid.Column="0" FontSize="16" Margin="10"/>
<Button Content="Live support" Grid.Row="1" Grid.Column="1" FontSize="16" Margin="10"/>
<Button Content="Email support" Grid.Row="1" Grid.Column="2" FontSize="16" Margin="10"
Click="Button_Click"/>
<TextBlock Grid.Row="2" Grid.ColumnSpan="3" Margin="30" TextAlignment="Center" TextWrapping="Wrap"
Text="Adventure Works" FontSize="36"/>
```



프로젝트 계속



1. Tutorial 3

A. 토론 페이지 생성

a. Menu.xaml 수정

b. 각 버튼 x:Name 지정

1) Product → BtnProduct, Live support → BtnLiveSupport, Email support → BtnEmailSupport

c. 이전 Email support 버튼 클릭 이벤트 수정

1) 기존 Button_Click 삭제 후 더블클릭

2) Button_Click 내 코드, BtnEmailSupport_Click으로 잘라내기, 붙여넣기 후

3) Button_Click 메서드 삭제

① Shift + Delete 추천

```
참조 1개
private void Button_Click(object sender, RoutedEventArgs e)
{
    NavigationService.Navigate(
        new Uri("/Contact.xaml", UriKind.RelativeOrAbsolute)
    );
}

참조 0개
private void BtnEmailSupport_Click(object sender, RoutedEventArgs e)
{
}
```

Ctrl+X

Ctrl+V

프로젝트 계속

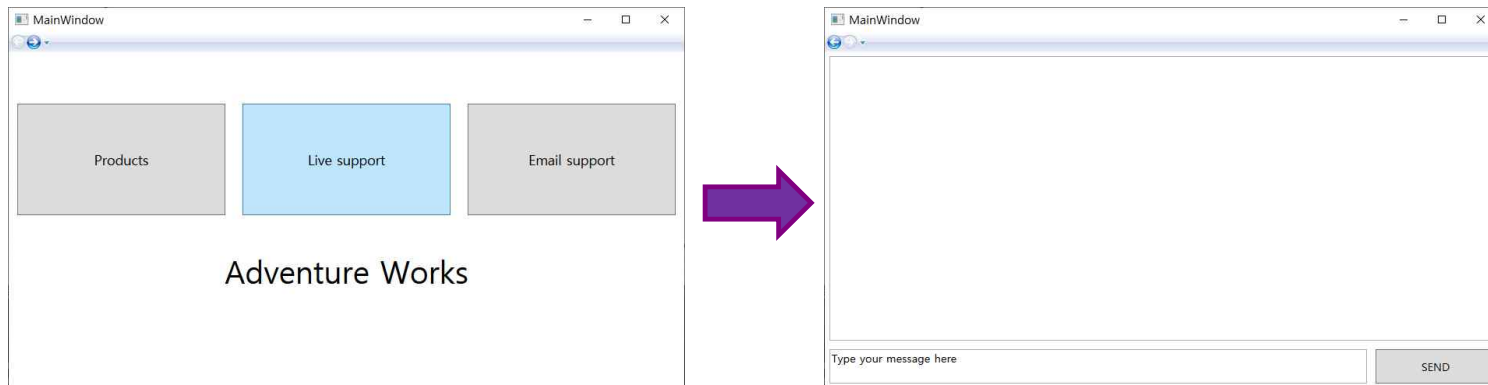
1. Tutorial 3

A. 토론 페이지 생성

- a. Menu.xaml 수정
- b. Live support 버튼 선택 후 이벤트 Click 확인 및 텍스트박스 더블클릭

```
private void BtnLiveSupport_Click(object sender, RoutedEventArgs e)
{
    NavigationService.Navigate(
        new Uri("/Discussion.xaml", UriKind.RelativeOrAbsolute)
    );
}
```

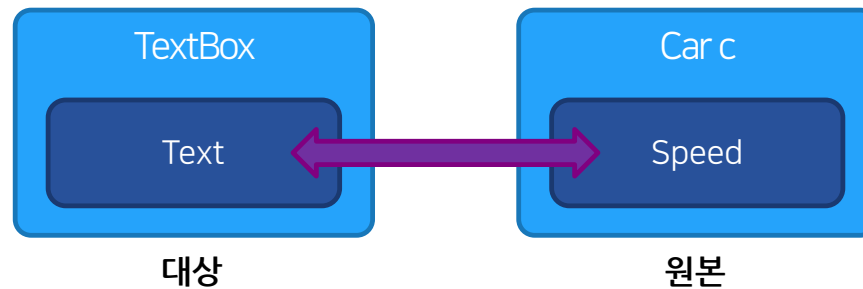
c. 결과 확인



XAML 이론

1. 데이터 바인딩

- A. 현재의 거의 모든 앱은 데이터 중심. 데이터 저장소(데이터베이스, 파일시스템, 클라우드 등)에서 데이터를 가져와 사용자에게 표시, 사용자가 변경, 업데이트 된 내용을 저장소에 다시 저장
- B. 바인딩의 두가지 디자인 패턴
 - a. Early(static) Binding - (전통적) 함수호출에서 실행될 함수나 프로시저를 컴파일 시점에 결정하는 것
 - b. Lazy(dynamic) Binding - (객체지향적) 메시지에 응답하기 위해 실행될 메서드를 실행시점에 결정하는 것(게으른)
 - 1) 적절하게 사용될 경우 프로그램의 운영 차원의 효율성
- C. 데이터를 표시하는 컨트롤 → 컨트롤 속성에 변수 할당 → 사용자는 이 컨트롤로 데이터 확인 → 변경 후 이벤트처리로 원 변수 값을 변경 → 수정된 데이터를 저장소로 이동



```
<TextBox Text="{Binding Speed, ElementName=c}" />
```

XAML 이론

1. 데이터 바인딩

A. 사용법

a. Binding 키워드

```
<TextBox Text="{Binding Speed}" />
```

b. 명시적 구문과 동일

```
<TextBox Text="{Binding Path=Speed}" />
```

c. Binding 구문내 속성

- 1) ElementName - 상태 표시
- 2) Source - 원본 데이터 객체를 리소스로 정의할 때

```
<TextBox Text="{Binding Source={StaticResource someList},Path=Height}" />
```

- 3) someList 컨트롤의 높이가 표시, TextBox 컨트롤에 새 값을 입력하면 someList의 높이가 변경됨

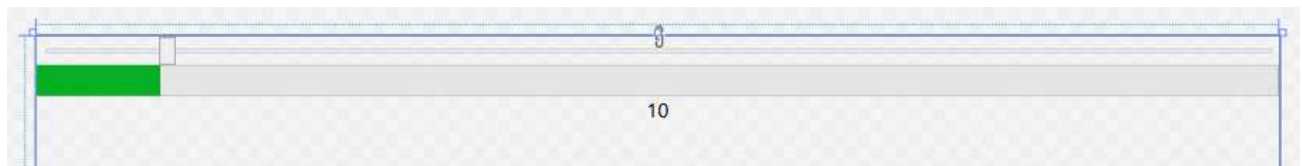
XAML 이론

1. 데이터 바인딩

A. 예제 분석

a. xaml 소스

```
<StackPanel>
  <Slider x:Name="slider" Maximum="100"
    Value="10" />
  <ProgressBar Value="{Binding Value, ElementName=slider}" Height="20" />
  <TextBlock Text="{Binding Value, ElementName=slider}" TextAlignment="Center" />
</StackPanel>
```



XAML 이론

1. 데이터 바인딩

A. 바인딩 모드

a. 4가지 모드

업데이트 시		
모드	대상 변경	값 변경
TwoWay	Yes	Yes
OneWay	No	Yes
OneWayToSource	Yes	No
OneTime	No	No

b. 모드 변경 시

```
{Binding Path=Speed, Mode=TwoWay}
```

XAML 이론

1. 데이터 바인딩

A. 데이터 컨텍스트

- a. 그룹화된 데이터들은 동일한 데이터 객체의 데이터 사용
- b. DRY(Don't Repeat Yourself)
- c. 바인딩에 소스 데이터 객체를 지정하지 않으면 소스가 현재 데이터 컨텍스트로 간주

```
<StackPanel DataContext="...">  
    <TextBox Text="{Binding Name}" />  
    <Label Content="{Binding SSN}" />  
</StackPanel>
```

- d. 코드 비하인드에서 할당 가능

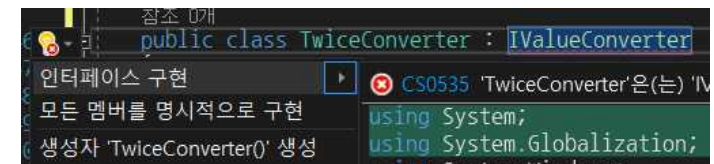
```
this.DataContext = ...;
```

XAML 이론

1. 데이터 바인딩

A. 변환기

- 데이터 바인딩 시 유형을 객체 유형을 변환
- IValueConverter 인터페이스를 상속해 작성하는 단순 클래스
 - Convert - 표준 메서드
 - ConvertBack - 양방향 데이터 바인딩에만 사용



```
namespace Maths
{
    public class TwiceConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return ((int)value) * 2;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            //throw new NotImplementedException();
            return null;
        }
    }
}
```


XAML 이론

1. 데이터 바인딩

A. 변환기

- a. xaml에서 사용시 아래와 같이 사용

```
<Page x:Class="BikeShop.Test"
      xmlns:c="clr-namespace:Maths">
  <Page.Resources>
    <c:TwiceConverter x:Key="twiceConv" />
  </Page.Resources>
  <Grid>
    <TextBlock value="{Binding Speed, Converter={StaticResource twiceConv}}" />
  </Grid>
</Page>
```

XAML 이론

1. 데이터 바인딩

A. 데이터 컬렉션

- a. 목록 컨트롤을 사용하는 컬렉션 표시 - IEnumerable(Array, Stack, List ...) 를 사용하는 ItemsSource

```
private void Inits()
{
    //myCar.Color = Colors.Blue;
    var cars = new List<Car>();
    for (int i = 0; i < 10; i++)
    {
        cars.Add(new Car()
        {
            Speed = i * 10
        });
    }
    this.DataContext = cars;
}
```

BusinessLogic.Car
BusinessLogic.Car
BusinessLogic.Car
BusinessLogic.Car
BusinessLogic.Car
BusinessLogic.Car
BusinessLogic.Car
BusinessLogic.Car
BusinessLogic.Car
BusinessLogic.Car

- b. xaml

```
<ListBox ItemsSource="{Binding}" />
```

XAML 이론

1. 데이터 바인딩

A. 목록 컨트롤 사용자 정의

- BusinessLogic.Car 만 나오는 이유 - WPF가 각 Car 클래스의 인스턴스를 표시하는 방법을 모르기 때문에 각 인스턴스의 ToString 메서드를 호출
- 방법
 - ItemsPanel은 요소를 배치하는 방법을 설명
 - ItemTemplate은 각 요소에 대해 반복이 필요한 템플릿을 제공 - 각 목록 항목에 대해 반복되는 DataTemplate 이어야 함
 - ItemContainerStyle은 항목을 선택하거나 마우스를 올릴 때 동작 방법을 설명
 - Template은 컨트롤 자체를 렌더링 하는 방법을 설명

c. 해결

```
<ListBox ItemsSource="{Binding}">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding Speed}" />
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

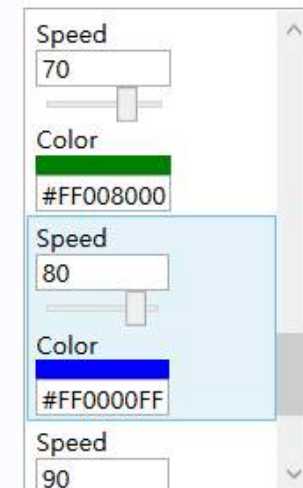
0
10
20
30
40
50
60
70
80
90

XAML 이론

1. 데이터 바인딩

- A. 목록 컨트롤 사용자 정의
 - a. 텍스트 외에도 표현 가능

```
<ListBox ItemsSource="{Binding}">
  <ListBox.ItemTemplate>
    <ItemContainerTemplate>
      <StackPanel>
        <TextBlock Text="Speed" />
        <TextBox Text="{Binding Speed}" />
        <Slider Value="{Binding Speed}" Maximum="100" />
        <TextBlock Text="Color" />
        <Border Height="10">
          <Border.Background>
            <SolidColorBrush Color="{Binding Color}" />
          </Border.Background>
        </Border>
        <TextBox Text="{Binding Color}" />
      </StackPanel>
    </ItemContainerTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```



XAML 이론

1. 데이터 바인딩

A. 요약

목록 주위(테두리, 배경,
스크롤바, ...)

• Template

항목 레이아웃

• ItemsPanel

각 항목의 모양

• ItemTemplate

요소 효과 (MouseOver,
Selected, ...)

• ItemContainerStyle

프로젝트 계속

1. Tutorial 4

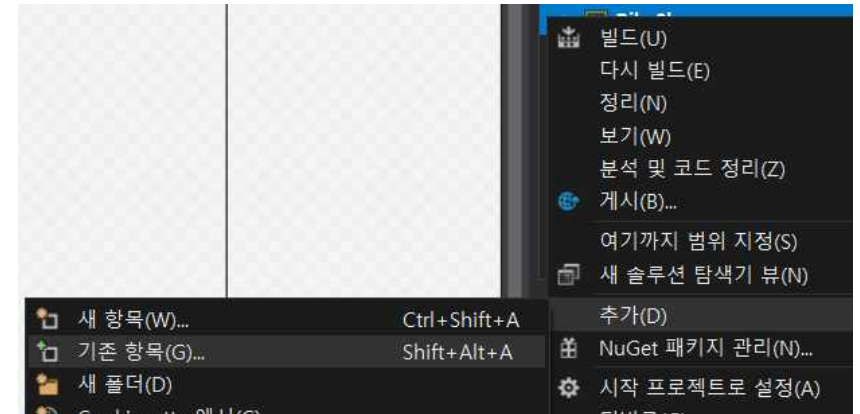
A. 데이터 객체 메시지 표시

- a. 프로젝트 컨텍스트 메뉴 > 추가 > 기존 항목
 - 1) chat.png, Talk.cs 파일 선택
- b. Discussion.xaml 파일 오픈
 - 1) Page 요소에 추가

```
xmlns:data="clr-namespace:BikeShop"
```

- 2) Grid 요소에 포함된 ListBox 선언 찾아서 수정

```
<ListBox Grid.ColumnSpan="2" Margin="5">  
    <ListBox.ItemsSource>  
        <data:Talk />  
    </ListBox.ItemsSource>  
</ListBox>
```



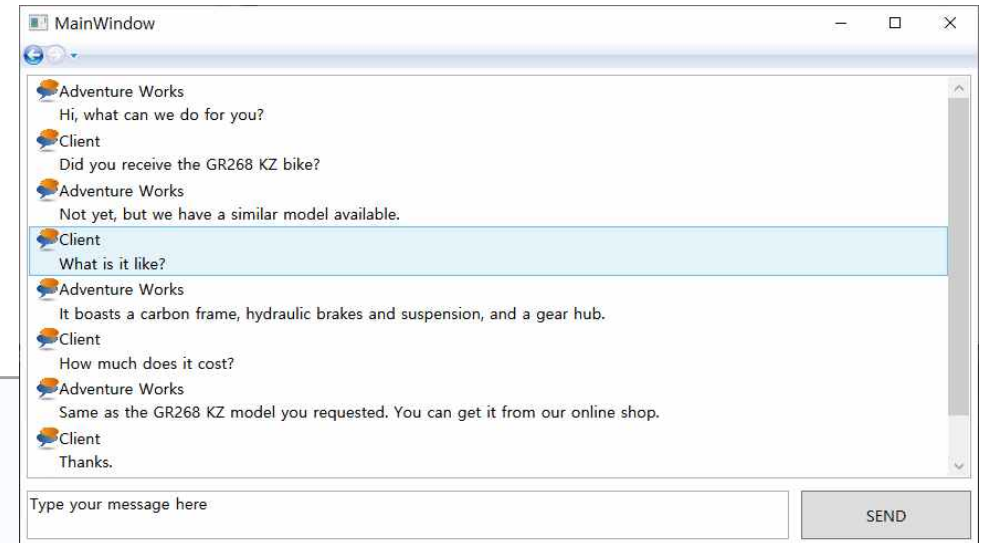
프로젝트 계속

1. Tutorial 4

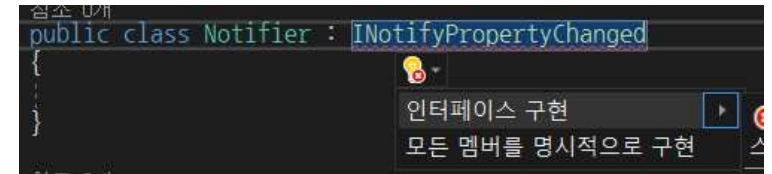
A. 데이터 객체 메시지 표시

a. ListBox 선언을 ItemTemplate 속성으로 대체

```
<ListBox Grid.ColumnSpan="2" Margin="5">
    <ListBox.ItemsSource>
        <data:Talk />
    </ListBox.ItemsSource>
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel>
                <StackPanel Orientation="Horizontal">
                    <Image Source="chat.png" Width="20" />
                    <TextBlock Text="{Binding Sender}" />
                </StackPanel>
                <TextBlock Text="{Binding Content}" Margin="20,0,0,0" TextWrapping="Wrap" />
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```



XAML 이론



1. 데이터 바인딩

- A. INotifyPropertyChanged - 사용자가 속성을 업데이트하면 바인딩된 컨트롤도 업데이트 되어야할 때 이벤트 발생
 - a. 데이터 객체에 속성 변경 이벤트가 필요

```
public class Notifier : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```


XAML 이론

1. 데이터 바인딩

A. INotifyPropertyChanged - 사용자가 속성을 업데이트하면 바인딩된 컨트롤도 업데이트 되어야할 때 이벤트 발생

a. 사용

```
public class Car : Notifier
{
    private double speed;
    public double Speed
    {
        get { return speed; }
        set
        {
            speed = value;
            OnPropertyChanged("Speed");
        }
    }
}
```

XAML 이론

1. 데이터 바인딩

- A. INotifyCollectionChanged - 컬렉션 내용이 변경되면 알림을 생성하는 이벤트
 - a. INotifyPropertyChanged 는 사용자가 메시지 보내거 받을 때 모든 목록 내용 제거하고 다시 추가해야 함
 - b. INotifyCollectionChanged 는 컬렉션의 추가, 제거 및 변경을 알릴 수 있음
 - 1) 직접 인터페이스를 구현할 필요도 없음

프로젝트 계속

1. Tutorial 5

A. 제품 및 세부 정보 표시

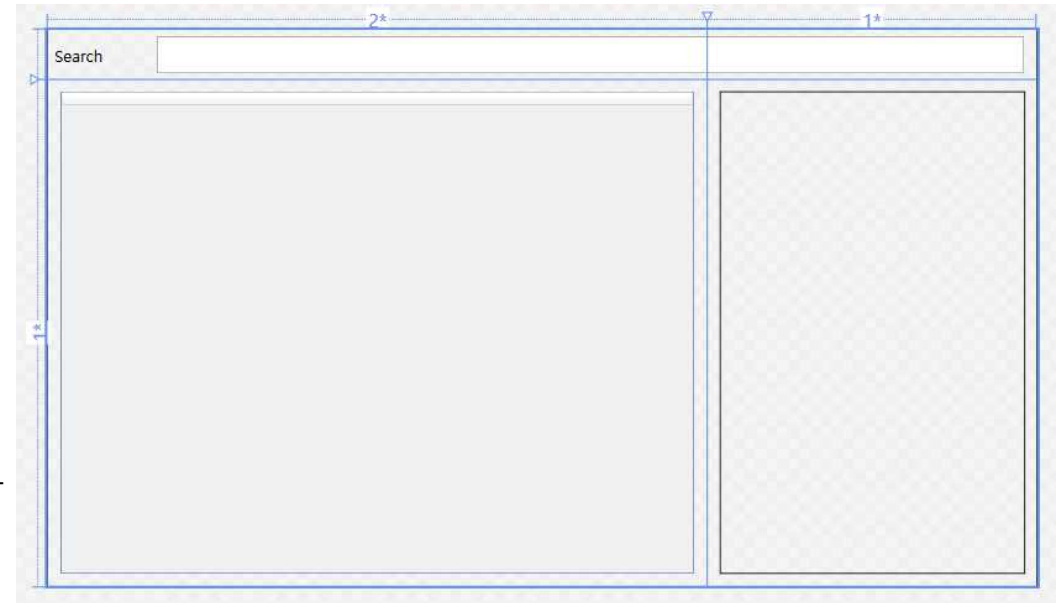
- a. ProductManagement.xaml 페이지 생성
- b. Notifier.cs 및 ProductsFactory.cs 파일 선택 추가 및 내용 확인
 - 1) Notifier를 상속받은 Product (Title, Price, Color, Reference)
 - 2) 내 속성이 변경되면 OnPropertyChanged 이벤트 호출
 - 3) In-memory data로 표시된 접혀있는 코드 부분은 products 제품 배열을 생성 100개의 제품을 랜덤으로 생성하는 로직

프로젝트 계속

1. Tutorial 5

A. 제품 및 세부 정보 표시

- a. ProductManagement.xaml 페이지 수정
- b. TextBox 이벤트 TextChanged 추가
- c. ProductManagement 클래스 상단과 TextChanged 이벤트에 코드 추가



```
ProductsFactory factory = new ProductsFactory();  
  
참조 0개  
private void TxtSearch_TextChanged(object sender, TextChangedEventArgs e)  
{  
    DgrProducts.ItemsSource = factory.FindProducts(TxtSearch.Text);  
}
```

프로젝트 계속

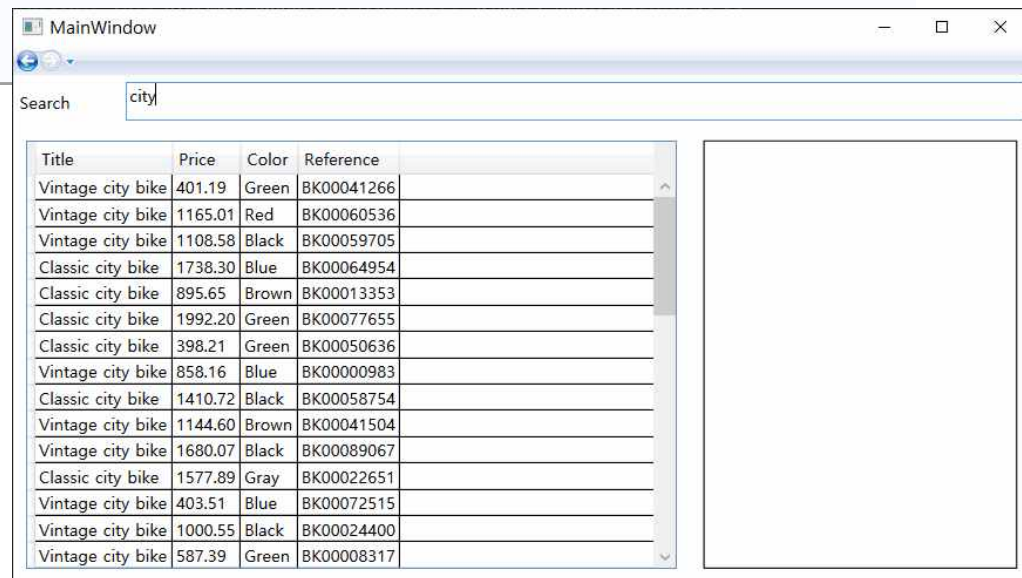
1. Tutorial 5

A. 화면 연결

- a. Menu.xaml 파일 열어서 Products 표시 버튼 더블클릭

```
private void BtnProducts_Click(object sender, RoutedEventArgs e)
{
    NavigationService.Navigate(
        new Uri("/ProductsManagement.xaml", UriKind.RelativeOrAbsolute)
    );
}
```

- b. 실행



프로젝트 계속

1. Tutorial 5

A. ProductManagement.xaml 파일 수정

a. Border 요소 상세 정보 태그 입력 - 실행 후 상세 확인 및 값 변경 확인

```
<Border BorderBrush="Black" BorderThickness="1" Grid.Column="1"
Margin="9.566,10.151,9.698,10.113" Grid.Row="1"
DataContext="{Binding SelectedItem,ElementName=DgrProducts}">
  <StackPanel Margin="10">
    <TextBox Text="Product details" FontWeight="Bold" FontSize="16"
      HorizontalAlignment="Center" Margin="10" />
    <TextBlock Text="Title" />
    <TextBox Text="{Binding Title, Mode=TwoWay}" />
    <TextBlock Text="Price" />
    <TextBox Text="{Binding Price, Mode=TwoWay}" />
    <TextBlock Text="Color" />
    <TextBox Text="{Binding Color, Mode=TwoWay}" />
    <Border Background="{Binding Color}" Height="10" />
    <TextBlock Text="Reference" />
    <TextBox Text="{Binding Reference, Mode=TwoWay}" />
  </StackPanel>
</Border>
```

Product details

Title

Classic city bike

Price

1620.70

Color

Red

Reference

BK00058567

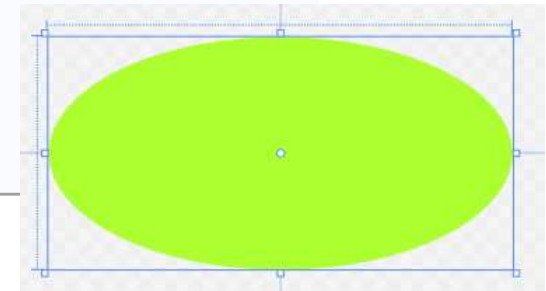
XAML 이론

1. 컨트롤 디자인

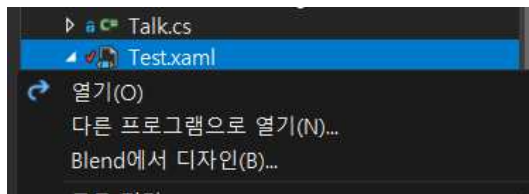
A. 템플릿

- a. 원품과 비교, 컨트롤의 디자인을 손쉽게 바꿀 수 있음
- b. 거의 모든 WPF 컨트롤에 Template 속성 존재, 컨트롤에 새로운 모양 적용하려면 ControlTemplate 인스턴스 할당

```
<Button Content="Press me">  
  <Button.Template>  
    <ControlTemplate TargetType="{x:Type Button}">  
      <Ellipse Fill="GreenYellow" />  
    </ControlTemplate>  
  </Button.Template>  
</Button>
```



c. Blend for Visual Studio



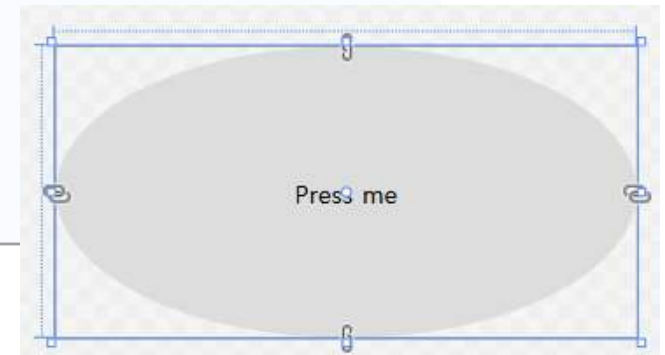
XAML 이론

1. 컨트롤 디자인

A. 템플릿바인딩

a. 이전 작업 계속

```
<Button Content="Press me">
  <Button.Template>
    <ControlTemplate TargetType="{x:Type Button}">
      <Grid>
        <Ellipse Fill="{TemplateBinding Background}" />
        <Label Content="{TemplateBinding Content}"
              HorizontalAlignment="Center"
              VerticalAlignment="Center" />
      </Grid>
    </ControlTemplate>
  </Button.Template>
</Button>
```



XAML 이론

1. 컨트롤 디자인

A. 템플릿바인딩

a. Blend for Visual Studio에서 수정

```
<Rectangle Fill="{TemplateBinding Background}" RadiusX="12" RadiusY="12" StrokeThickness="3" />  
<Label Content="{TemplateBinding Content}"  
    HorizontalAlignment="Center"  
    VerticalAlignment="Center" />
```



XAML 이론

1. 컨트롤 디자인

A. ItemsPresenter

- a. ListBox 및 ComboBox 같은 Items 속성이 있는 목록 컨트롤도 템플릿으로 만들 수 있음

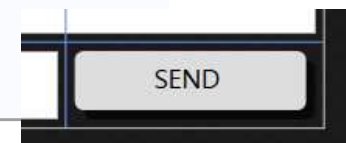
프로젝트 계속

1. Tutorial 6

A. 기본 버튼 템플릿 생성

- Discussion.xaml 파일 오픈
- Xaml 코드 버튼 디자인 수정

```
<Button Grid.Row="1" Grid.Column="1" Margin="5" Content="SEND">
  <Button.Template>
    <ControlTemplate TargetType="Button">
      <Grid>
        <Rectangle Fill="#AA000000" Margin="5,5,0,0" RadiusX="5.0" RadiusY="5.0" />
        <Rectangle Fill="{TemplateBinding Background}"
          Margin="0,0,5,5" RadiusX="5.0" RadiusY="5.0"/>
        <Viewbox Margin="5,5,15,15">
          <ContentPresenter />
        </Viewbox>
      </Grid>
    </ControlTemplate>
  </Button.Template>
</Button>
```



XAML 이론

1. 리소스

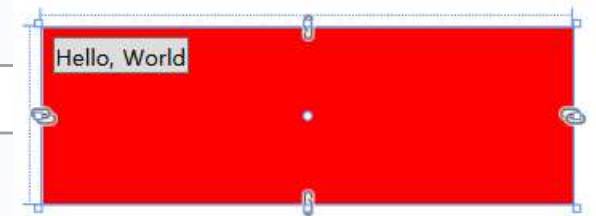
A. 컨트롤의 공유화

- a. 컨트롤에 대한 템플릿이 작성되면 애플리케이션 전체에서 사용할 수 있음
- b. 템플릿 코드를 복제하는 것은 유지보수가 불가능 → 리소스 사용
- c. 리소스 저장 장소
 - 1) 화면 : 페이지, 사용자 정의 컨트롤 또는 창과 같이 단일 화면으로 범위가 지정된 리소스
 - 2) 애플리케이션 : App.xaml에 선언된 Application 요소와 같이 애플리케이션 전반에 걸쳐 사용되는 리소스
- d. App.xaml 파일에 선언

```
<Application.Resources>  
    <Button x:Key="button">Hello, World</Button>  
    <SolidColorBrush x:Key="accentBrush" Color="Red" />  
</Application.Resources>
```

e. 실제 xaml에서 사용

```
<Label Content="{StaticResource button}"  
        Background="{StaticResource accentBrush}"/>
```



XAML 이론

1. 리소스

A. ResourceDictionaries

- a. App.xaml에 파일 유지보수가 어려울 수 있기때문에 리소스 사전이 만들어짐
- b. 프로젝트에 몇 개의 리소스 사전 파일을 추가 후 리소스 파일을 해당 파일의 Resources 속성에 참조

```
<Window.Resources>  
  <ResourceDictionary Source="Brushes.xaml" />  
</Window.Resources>
```

프로젝트 계속

1. Tutorial 7

A. 배경 설정

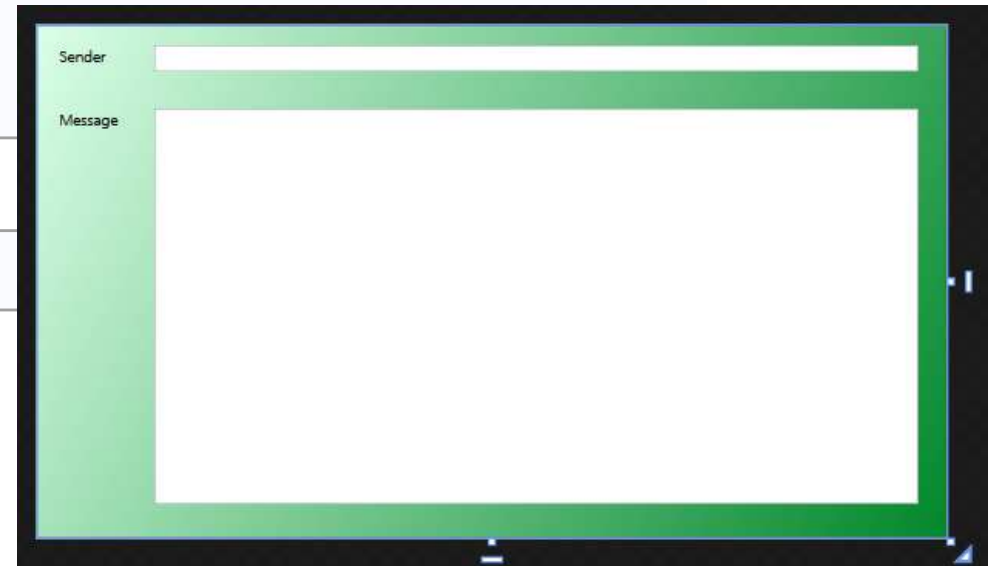
a. App.xaml 파일 더블 클릭

1) Application.Resources 요소 아래에 XAML 코드를 추가

```
<LinearGradientBrush x:Key="background">  
    <GradientStop Color="#FFDBFFE7" Offset="0" />  
    <GradientStop Color="#FF03882D" Offset="1" />  
</LinearGradientBrush>
```

b. 각 화면의 Page 요소에 Background 특성을 추가

```
Background="{StaticResource background}"
```



XAML 이론

1. 스타일

A. 컨트롤 모양 스타일링 → 다중 속성 설정자(multi property setters)

a. 여러 컨트롤의 배경을 설정할 경우 리소스 없이 만들 수도 있으나 리소스로 작업하는 것이 간편

```
<Style x:Key="niceButton" TargetType="Button">
  <Setter Property="Width" Value="50" />
  <Setter Property="Height" Value="50" />
  <Setter Property="Background">
    <Setter.Value>
      <LinearGradientBrush>
        <GradientStop Color="Red" />
        <GradientStop Color="Yellow" Offset="1" />
      </LinearGradientBrush>
    </Setter.Value>
  </Setter>
</Style>
```

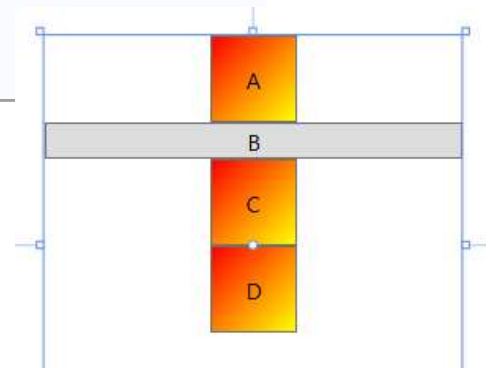
XAML 이론

1. 스타일

A. 컨트롤 모양 스타일링 사용

a. XAML

```
<StackPanel Margin="30">  
    <Button Style="{StaticResource niceButton}" Content="A" />  
    <Button Content="B" />  
    <Button Style="{StaticResource niceButton}" Content="C" />  
    <Button Style="{StaticResource niceButton}" Content="D" />  
</StackPanel>
```

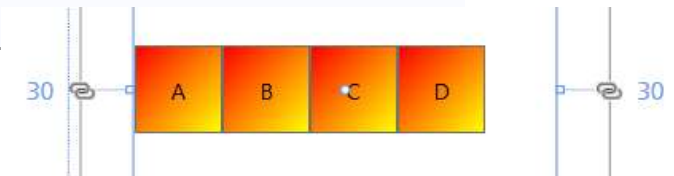


XAML 이론

1. 스타일

- A. 컨트롤 모양 스타일링, 두번째 방법
 - a. 각 페이지의 리소스에 저장

```
<Page.Resources>  
    <Style TargetType="Button">  
        ...  
    </Style>  
</Page.Resources>  
  
<StackPanel Margin="30" Orientation="Horizontal">  
    <Button Content="A" />  
    <Button Content="B" />  
    <Button Content="C" />  
    <Button Content="D" />  
</StackPanel>
```



프로젝트 계속

1. Tutorial 8

A. 메뉴 페이지 개선

a. Menu.xaml 파일 더블 클릭

```
<Page.Resources>
  <Style TargetType="Button">
    <Setter Property="Template">
      <Setter.Value>
        <ControlTemplate />
      </Setter.Value>
    </Setter>
  </Style>
</Page.Resources>
```

b. 이전 Discussion.xaml에서 선언한 ControlTemplate 요소 부분 코드를 복사

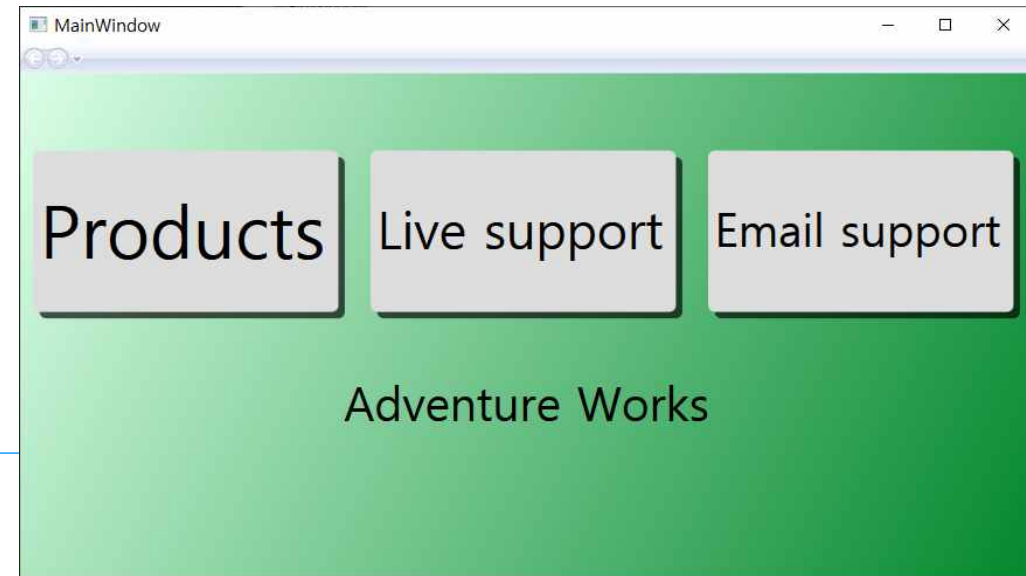
프로젝트 계속

1. Tutorial 8

A. 메뉴 페이지 개선

a. Menu.xaml 파일 에 붙여넣기

```
style TargetType="Button" /
<Setter Property="Template">
  <Setter.Value>
    <ControlTemplate TargetType="Button">
      <Grid>
        <Rectangle Fill="#AA000000" Margin="5,5,0,0" RadiusX="5.0" RadiusY="5.0" />
        <Rectangle Fill="{TemplateBinding Background}"
          Margin="0,0,5,5" RadiusX="5.0" RadiusY="5.0" />
        <Viewbox Margin="5,5,15,15">
          <ContentPresenter />
        </Viewbox>
      </Grid>
    </ControlTemplate>
  </Setter.Value>
</Setter>
```



XAML 이론

1. 테마

- A. 암시적 및 명시적 스타일이 포함된 ResourceDictionary 파일 제공 → 테마로 호칭
 - a. <https://archive.codeplex.com/?p=wpfthemes> or
 - b. <https://github.com/StanislawSwierc/WpfThemesCollection> 등 github를 활용

MVVM 패턴

1. 스파게티 코드

A. 유지보수가 불가능하고 재사용이 불가능한 코드

a. 문제점

- 1) 거대한 파일 만들고 유지보수가 갈수록 어렵게 됨. 한 화면에 대해 5,000줄 이상의 코드 비하인드 생성 가능
- 2) 테스트가 어려움. 테스트 실행 시 UI를 인스턴스 해야 함
- 3) 재사용 어려움. 컨트롤에 대한 참조는 다른 화면에서 사용 불가

```
void Loaded(object sender, RoutedEventArgs args) {  
    BankData = GetBankData();  
    BankData.DataChanged += new EventHandler(BankData_Changed);  
}  
  
void BankData_Changed(object Sender, BankDataEventArgs args) {  
    BankData b = args.Data;  
    this.BalanceDisplay = b.Balance;  
    if (this.IncludeInterests.Checked) {  
        this.BalanceDisplay = b.Balance * b.InterestRate;  
    }  
}  
  
void BalanceDisplay_TextChanged(object sender, RoutedEventArgs args) {  
    ...  
}
```

MVVM 패턴

1. MVC

A. Model View Controller - 스파게티 개선 모델

a. 세 가지 파트

- 1) 뷰 : 순수 XAML
- 2) 모델 : INotifyPropertyChanged 및 INotifyCollectionChanged를 구현하는 클래스
- 3) 컨트롤러 : 명령, 트리거, 관련 이벤트, NavigationService

2. MVVM

A. Model View ViewModel 패턴



a. 세 가지 파트

- 1) 뷰 : UI. 이상적으로 View 는 순수 XAML로 구성. View의 DataContext는 ViewModel 의미하며, 데이터바인딩은 둘 사이의 접착제 역할
- 2) 뷰모델 :
 - ① 하나의 뷰에 대한 메서드(명령)로 속성 및 액션을 사용해 데이터 노출
 - ② 뷰에 크게 의존하며, 다른 DataModel 혼합을 허용하거나 비동기 호출의 복잡성을 숨길 수 있음
 - ③ 단위 테스트를 쉽게
 - ④ INotifyPropertyChanged 를 구현
- 3) (데이터)모델 : 비즈니스 클래스. UI에 제공된 데이터를 가지고 있음

MVVM 패턴

1. 권장 구현 단계

A. MVVM 사용 화면 코딩 단계

- a. ViewModel을 생성
- b. ViewModel이 노출해야 하는 속성 검색
- c. 알림 속성을 코딩
- d. ViewModel을 View 의 DataContext로 사용
- e. View를 ViewModel에 데이터바인딩
- f. 기능적 논리를 코딩

B. ViewModel 생성

- a. 단순 클래스인 ViewModel은 각 화면당 하나씩 있어야 함
- b. ViewModel 클래스는 INotifyPropertyChanged를 구현해야 함 (Notifier 클래스 상속)
- c. 생성 시 권자 파일명
 - 1) View : \View\MyScreen.xaml
 - 2) ViewMode : \ViewModel\MyScreenViewModel.cs

MVVM 패턴

1. 권장 구현 단계

A. ViewModel이 노출해야 하는 속성 찾기

a. 모든 사용자 입/출력에 대하여 ViewModel에 속성 추가

B. 코드 알림 속성

a. ViewModel에 추가하는 속성에 알림 속성(Notifying properties)이어야 함

```
private double speed;

public double Speed {
    get { return speed; }
    set {
        speed = value;
        OnPropertyChanged("Speed");
    }
}
```

b. 어떤 시점에 해당 속성 중에 하나가 변경되면 ViewModel이 무언가를 수행하게 할 수 있음

MVVM 패턴

1. 권장 구현 단계

A. ViewModel 확장

```
private double speed;

public double Speed {
    get { return speed; }
    set {
        speed = value;
        OnPropertyChanged("Speed");
        OnSpeedChanged();
    }
}

void OnSpeedChanged() {
    // 기능적인 코드 추가
}
```

MVVM 패턴

1. 권장 구현 단계 - 사이드

A. View의 DataContext로 ViewModel 사용

a. 첫번째 방법 - XAML 사용 및 코드 비하인드 사용

```
<Window xmlns:vm="clr-namespace:ViewModels"
    ...>
    <Window.DataContext>
        <vm:MyScreenViewModel />
    </Window.DataContext>
</Window>
```

b. 단점 - 디자인 뷰를 표시할 때 ViewModel 클래스가 인스턴스화

c. 두번째 방법 - 코드 비하인드

```
public partial class MyScreen : Window {
    public MyScreen() {
        InitializeComponent();
        this.DataContext = new MyScreenViewModel();
    }
}
```

d. 단점 - 디자인뷰에서 바인딩 편집기의 도움을 못받음

MVVM 패턴

1. 권장 구현 단계

A. ViewModel 사용 : 요구 버전

- a. 런타임에만 ViewModel 인스턴스화 및 데이터바인딩 편집기를 얻기 위한 중간 해결책
- b. 클래스를 지정하는 XAML의 d:DataContext 특성 사용

```
<Window xmlns:vm="clr-namespace:ViewModels"
        mc:Ignorable="d"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        d:DataContext="{d:DesignInstance vm:MyScreenViewModel}"
        ...>
...
</Windows>
```

c. 코드 비하인드에서 DataContext 할당

```
public partial class MyScreen : Window {
    public MyScreen() {
        InitializeComponent();
        this.DataContext = new MyScreenViewModel();
    }
}
```

프로젝트 계속

1. Tutorial 9

A. MVVM

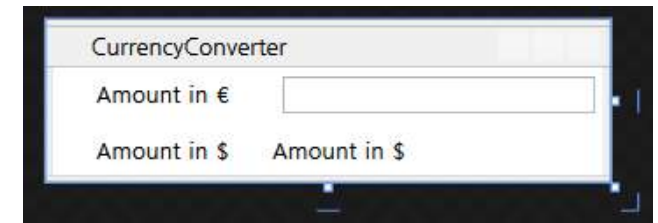
a. Action을 처리하는 방법 - 간단한 통화 변환기

1) ViewModel 생성

```
public class CurrencyConverterViewModel : Notifier
{
}
```

2) 필요한 속성 찾기

```
private decimal euros;
private decimal dollars;
public decimal Euros {
    get { return euros; }
    set { euros = value; OnPropertyChanged("Euros"); }
}
public decimal Dollars {
    get { return dollars; }
    set { dollars = value; OnPropertyChanged("Dallars"); }
}
```



프로젝트 계속

1. Tutorial 9

A. MVVM

a. Action을 처리하는 방법 - 간단한 통화 변환기

1) View의 DataContext로 ViewModel을 사용. XAML에 작성

```
xmlns:vm="clr-namespace:BikeShop.ViewModel "  
    Title="CurrencyConverter" Height="91" Width="316.088">  
    <Window.DataContext>  
        <vm:CurrencyConverterViewModel />  
    </Window.DataContext>  
</Grid>
```

2) 데이터바인딩

```
<TextBox Grid.Row="0" Grid.Column="1" Margin="5" Text="{Binding Euros}" />  
    <TextBlock Grid.Row="1" Grid.Column="0" Text="Amount in $" TextAlignment="Center"  
VerticalAlignment="Center" />  
    <TextBlock Grid.Row="1" Grid.Column="1" TextAlignment="Left" VerticalAlignment="Center"  
Text="{Binding Dollars}" />
```

프로젝트 계속

1. Tutorial 9

A. MVVM

a. Action을 처리하는 방법 - 간단한 통화 변환기

1) 기능적 논리 코딩

```
public decimal Euros {  
    get { return euros; }  
    set {  
        euros = value;  
        OnPropertyChanged("Euros");  
        OnEurosChanged();  
    }  
}  
  
private void OnEurosChanged() {  
    Dollars = Euros * 1.1M;  
}
```

프로젝트 계속

1. Tutorial 9

A. MVVM

- a. Action을 처리하는 방법 - 복잡한 예제
 - 1) 더 까다로운 화면

Currency	<input type="text"/>
Amount in €	<input type="text" value="0"/>
Amount in CURRENCY	<input type="text" value="0"/>

```
public class Currency
{
    public string Title { get; set; }
    public decimal Rate { get; set; }

    public Currency(string title, decimal rate)
    {
        Title = title ?? throw new ArgumentNullException(nameof(title));
        Rate = rate;
    }
}
```

```
public class CurrencyConverterViewModel2 : Notifier {
    // 무시할 수 있는 장황한 코드
    private decimal euros;
    private decimal converted;
    private Currency selectedCurrency;
    private IEnumerable<Currency> currencies;
    private string resultText;

    public decimal Euros {
        get { return euros; }
        set {
            euros = value;
            OnPropertyChanged("Euros");
            OnEurosChanged();
        }
    }

    public decimal Converted {
        get { return converted; }
        set {
            converted = value;
            OnPropertyChanged("Converted");
        }
    }

    public Currency SelectedCurrency {
        get { return selectedCurrency; }
        set {
            selectedCurrency = value;
            OnPropertyChanged("SelectedCurrency");
            OnSelectedCurrencyChanged();
        }
    }
}
```



```

public IEnumerable<Currency> Currencies {
    get { return currencies; }
    set { currencies = value;
        OnPropertyChanged("Currencies");
    }
}

public string ResultText {
    get { return resultText; }
    set { resultText = value;
        OnPropertyChanged("ResultText");
    }
}

// 생성자!!
public CurrencyConverterViewModel2() {
    Currencies = new Currency[] {
        new Currency("US Dollar", 1.1M), new Currency("British Pound", 0.9M),
    };
}

private void OnEurosChanged() { ComputeConverted(); }

private void OnSelectedCurrencyChanged() { ComputeConverted(); }

private void ComputeConverted() {
    if (SelectedCurrency == null) return;

    Converted = Euros * SelectedCurrency.Rate;
    ResultText = string.Format($"Amount in {SelectedCurrency.Title}");
}
}

```

프로젝트 계속

1. Tutorial 9

A. MVVM

a. Action을 처리하는 방법 - 복잡한 예제

1) View 에서 데이터바인딩

```
<ComboBox SelectedItem="{Binding SelectedCurrency}" ItemsSource="{Binding Currencies}">
  <ComboBox.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding Title}" />
    </DataTemplate>
  </ComboBox.ItemTemplate>
</ComboBox>
<TextBox Text="{Binding Euros}" />
<TextBlock Text="{Binding ResultText}" />
<TextBlock Text="{Binding Converted}" />
```

프로젝트 계속

1. Tutorial 9

A. MVVM 명령과 메서드

- a. Textbox Text 속성과 ComboBox SelectedItem 속성은 사용자 상호작용이 바로 됨
- b. 버튼에 대한 처리는 명령을 사용해야 함

B. 명령: 쉬운 방법

- a. ICommand 인터페이스 구현하여 해당 클래스를 인스턴스화 하면 Button 및 MenuItem 컨트롤의 Command 속성 사용, 해당 인스턴스를 참조 가능
- b. ICommand 구현하면 꽤 많은 코드가 필요, 대부분의 MVVM 프레임워크는 DelegateCommand 클래스 제공하여 해당 프로세스 정보를 훨씬 간결하게 만듦
- c. 인스턴스화된 명령을 ViewModel의 특성에 할당해야 함
- d. Click 이벤트만 명령을 트리거함. MouseOver 같은 다른 이벤트를 처리하려면 메서드를 사용하는 것과 비슷한 모양을 갖는 자세한 XAML을 사용해야 함

프로젝트 계속

1. Tutorial 9

A. MVVM

```
<Button>
  <i:Interaction.Triggers>
    <i:EventTrigger EventName="Click">
      <i:InvokeCommandAction
        Command="{Binding LoadMoreHotelsCommand}"/>
    </i:EventTrigger>
  </i:Interaction.Triggers>
</Button>
```

- a. 답, 사용하지 말것(?)

프로젝트 계속

1. Tutorial 9

A. MVVM - 메서드 : 쉬운 방법

- a. 메서드를 사용하면 C# ViewModel이 매우 쉬워지고 XAML은 더 복잡해짐
- b. 프로젝트에 System.Windows.Interactivity.dll 참조 추가 - NuGet System.Windows.Interactivity.WPF 다운로드
- c. XAML 화면 루트 요소에 특성 추가

```
xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"  
xmlns:ei="http://schemas.microsoft.com/expression/2010/interactions"
```

- d. XAML의 각 컨트롤을 위한 트리거 추가

```
??
```

프로젝트 계속

1. Tutorial 9

A. MVVM - 권장단계

- a. ViewModel 생성
- b. ViewModel이 공개해야 하는 속성(입력, 출력)과 메서드(액션) 검색
- c. 알림 속성을 선언하고 공용 메서드 추가
- d. ViewModel을 View의 DataContext로 사용
- e. View를 ViewModel 속성에 데이터 바인딩
- f. View에 ViewModel 메서드를 호출하는 트리거를 추가
- g. 기능적 논리코딩

프로젝트 계속

1. Tutorial 9

A. MVVM 요약

- a. INotifyPropertyChanged 구현
- b. 메서드 호출하는 트리거 XAML 작성
- c. ViewModel 인스턴스화, View에 할당과 같은 연결 작업
- d. ViewModel 이나 뷰 간 통신방법 포함하는 종속성 주입이 필요

B. MVVM 프레임워크는

- a. INotifyPropertyChanged 구현하는 ViewModel 생성하기 위한 ViewModelBase 클래스 상속
- b. 오류 제한, 작업자 스레드에 알림이 발생하기 않게 속성 변경 알림
- c. 명령을 사용하기로 결정해야 하지만 명령을 쉽게 작성하기 위한 DelegateCommand 클래스 사용

C. 이를 도와주는 프레임워크 종류

- a. Prism - Microsoft Patterns and Practices 팀에서 제작,
- b. MVVM Light - 가볍고 좋으나 현재 방치됨
- c. Caliburn.Micro - 코드가 간단함
- d. 자신에게 맞는 것을 결정, 중소형에는 Caliburn / 대형프로젝트에는 Prism 추천