

헤드 퍼스트 C#의 일부 프로젝트들은 윈도우 8이 필요한 느토어 앱으로 되어 있습니다. 부록 ii에서 WPF를 이용하여 느토어 앱으로 만든 프로젝트를 데스크톱 앱으로 만들어 봅시다.

PDF Windows Presentation Foundation

헤드 퍼스트 C#, WPF 사용자 가이드

좋은 소식이 있어요! 당신의 요청이 승인되어서 컴퓨터를 Windows 2003으로 업그레이드 해줄대요.



윈도우 8이 없다고요? 괜찮습니다.

헤드 퍼스트 C#의 3판에서 일부 프로젝트를 하기 위해서는 윈도우 8과 Visual Studio 2013이 필요합니다. 하지만, 아직까지 회사나 집에서 최신 버전의 윈도우를 사용하는 사람들이 많지 않습니다. 그래서 WPF(Windows Presentation Foundation)가 있는 거죠. WPF는 조금 오래된 윈도우에서 실행되는 Visual Studio 2008과 2010에서도 동작합니다. WPF에는 **C#의 핵심 기술**이 포함되어 있습니다. 만약 여러분이 윈도우 8을 사용하고 있더라도, WPF를 설치하고 경험해 보는 것은 큰 도움이 될 것입니다. 이 부록에서 WPF를 이용하여 책에 있는 **윈도우 스토어 앱** 프로젝트를 **데스크톱 앱**으로 만들어 봅시다.

WPF, 왜 배워야 되죠?

WPF(Windows Presentation Foundation)는 .NET으로 쓰인 프로그램의 사용자 인터페이스를 만들기 위한 기술입니다. WPF 프로그램은 일반적으로 윈도우 데스크톱의 사용자 인터페이스를 보여 줍니다. WPF는 윈도우 소프트웨어 개발 중에서 하나의 인기 있는 기술이기도 하죠. 그리고 WPF의 친숙한 환경과 C#과 .NET의 전문 개발자들을 위해서 마이크로소프트 개발팀에서 많은 노력을 기울였습니다.

WPF 프로그램은 XAML(Extensible Application Markup Language)을 이용해서 사용자 인터페이스를 만듭니다. 이 책에서 윈도우 스토어 앱에 관한 내용을 접한 독자들에게는 정말 좋은 소식이지요. 이 책의 대부분의 윈도우 스토어 앱 프로젝트들은 약간의 XAML 코드 수정 혹은 수정 없이 WPF로 만들 수 있습니다.

앱 바와 페이지 네비게이션 같은 것들은 윈도우 스토어 앱과 다릅니다. 부록에서는 WPF에서 대체 가능한 것들을 보여 줄 거예요.

윈도우 8과 Visual Studio 2013을 쓰고 있는데 WPF를 해야 하나요?



C# 개발자들은 WPF를 다룰 수 있어야 합니다

거의 모든 프로그래밍 언어는 수많은 환경과 운영체제에서 사용할 수 있어야 합니다. C#도 예외는 아니죠. 여러분이 만약 C# 전문 개발자가 되고 싶다면, 다양한 기술을 접하고 현장에서 적용할 수 있어야 합니다. 그리고 WPF는 특히 C# 개발자들에게 중요한 기술이죠. 회사에서 WPF를 이용하여 많은 프로그램을 만들고 있기 때문입니다. 앞으로도 계속해서 그럴 거예요. 만약 여러분의 목표가 C# 전문가라면, WPF는 이력서의 리스트에 작성되어야 할 겁니다.


WPF를 배우는 것은 또한 윈도우 8을 사용하고 헤드 퍼스트 C#에 있는 모든 코드를 작성한 독자들에게 좋은 취미거리가 될 것입니다. 개발 언어를 학습하는 데 가장 효과적인 방법은 같은 문제를 다양한 방법으로 접근하는 거죠. 이 부록에서는 WPF를 이용하여 책에 있는 프로젝트를 만들기 위한 방향을 제시해 줍니다. WPF와 윈도우 8로 만든 프로젝트를 비교해서 살펴보는 것은 여러분에게 가치 있는 관점을 제공해 줍니다. 그리고 이것은 좋은 개발자가 되기 위한 하나의 과정이기도 합니다.

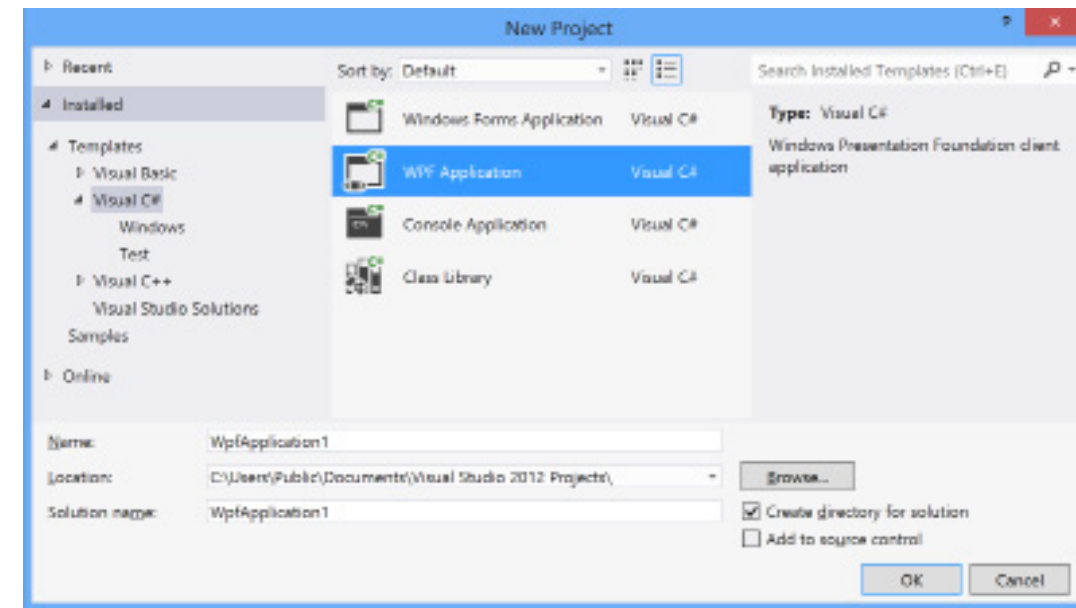
아래의 사이트에서 프로젝트 코드를 내려받을 수 있습니다.

<http://www.hanbit.co.kr/exam/2165>

비주얼 스튜디오에서 WPF 프로젝트 만들기

비주얼 스튜디오에서 새 WPF 응용 프로그램을 생성하는 것은 데스크톱 응용 프로그램을 생성하는 것과 비슷합니다. Visual Studio Express 2013일 경우, Visual Studio 2013 Express for Desktop 에디션을 사용하세요(for Windows 에디션은 WPF 프로젝트를 생성하지 않습니다). 여러분은 또한 Visual Studio 2013 Professional, Premium 혹은 Ultimate 에디션을 사용할 수 있습니다. 새 프로젝트를 생성할 때, 파일(File) 메뉴 > 새 프로젝트(New Project) > Visual C#을 선택하고,

 WPF Application (WPF 응용 프로그램)을 선택해 주세요.



또한 Visual Studio 2010, Visual C# 2010 Express, Visual Studio 2008에서 C# WPF 응용 프로그램을 생성할 수 있습니다. Visual Studio 2008 혹은 2010 Express 에디션을 사용한다면, 프로젝트 파일은 처음에 임시 폴더에 생성됩니다. 그리고 저장(Save) 혹은 모두 저장(Save All)을 누를 때까지는 새 프로젝트 창에서 작성한 위치에 저장되지 않습니다.

WPF는 인터넷 익스플로러나 다른 브라우저안에서 실행하는 XAML 브라우저 응용 프로그램을 만들 수 있습니다. 이 부록에서는 다루지 않지만, 자세한 사항은 아래의 사이트를 참고하세요.
<http://msdn.microsoft.com/ko-kr/library/aa970060.aspx>
마이크로소프트는 XAML을 사용한 또 다른 기술이 있습니다. 이를 실버라이트(Silverlight)라고 하는데, 자세한 사항은 아래의 사이트를 참고하세요.
<http://www.microsoft.com/korea/silverlight/>

부록에 대한 설명

이 부록은 WPF 프로젝트를 위해서 헤드 퍼스트 C# 3판을 대체하는 내용을 담고 있습니다. 부록을 각 장별로 나눠서 무엇을 해야 하는지에 대한 전반적인 개요와 어떤 페이지가 바뀌었고, 책의 어떤 부분을 읽어야 하는지에 대한 가이드를 제시합니다.

비주얼 스튜디오가 최신 버전이 아니라도, 프로젝트를 만들 수 있습니다. 하지만 조금 힘들 수 있습니다.

마이크로소프트팀은 XAML을 편집하는 기능에서 특히 Visual Studio 2013의 사용자 인터페이스를 향상시키는 데 많은 노력을 기울였습니다. 그리고 헤드 퍼스트 C#에서 한 가지 중요한 것은 개발 도구인 비주얼 스튜디오 IDE를 경험하고 학습하는 겁니다. 가능하다면, 비주얼 스튜디오의 최신 버전의 사용을 권장합니다.

하지만, 모든 독자들이 Visual Studio 2013을 설치할 수 없다는 걸 잘 알고 있습니다(예를 들어, 회사 컴퓨터를 사용하는 독자들이 새로운 소프트웨어를 설치하려면, 관리자 권한이 필요한 경우도 있습니다). 이전 버전의 비주얼 스튜디오로 프로젝트들을 수행하기 힘들더라도, 여러분이 이 책을 잘 활용했으면 좋겠습니다. 우리가 할 수 있는 한 많은 지침을 제공하기 위해서 최선을 다했습니다. 그러나 비주얼 스튜디오의 최신 버전을 사용하는 독자 대부분에게 해를 끼치지 않도록 학습의 균형을 유지하고 있습니다.

여러분이 Visual Studio 2010 혹은 그 이전 버전의 사용으로 인해 어려운 점이 있다면, 책에 나오는 메뉴 옵션들을 여러분의 화면에서 찾을 수 없는 경우입니다. 이 경우에는 XAML과 C# 코드를 직접 손으로 입력하는 것을 권장합니다. 비주얼 스튜디오에 코드를 복사 및 붙여넣기해도 좋습니다. XAML 코드가 정확하다면, IDE에서 제공하는 기능들을 쉽게 파악할 수 있습니다.

부록에서는 다운로드할 수 있는 책의 모든 소스 코드를 제공합니다. 메뉴 옵션이 없어서 코드를 복사 및 붙여넣기를 해야 할 경우에 사용하면 좋습니다. 소스 코드의 자세한 내용은 이 책의 웹사이트 (<http://www.hanbit.co.kr/exam/2165>)를 참고하세요.

여러분은 또한 코드를 직접 <http://hfcsharp.codeplex.com/>에서 다운로드할 수 있습니다. 부록에서는 반드시 WPF 폴더에서 코드를 내려 받아야 합니다. 만약 WPF 프로젝트에서 윈도우 스토어 코드를 사용하려고 하면 오류를 만나게 될 거예요.

이 부록에서는 이 책 혹은 PDF에서 대체된 부분의 페이지를 찾을 수 있습니다. 페이지 번호를 잘 참고해 주세요. 만약 여러분이 킨들(Kindle)이나 또 다른 eBook을 사용한다면, 페이지 번호가 맞지 않을 수도 있습니다. 대신에 해당 챕터의 소제목을 찾아보세요. 예를 들어 이 부록의 2장에서 종이 책의 72, 73 페이지를 대체할 "기초부터 만들기"라는 소제목이 있습니다. 그러면 eBook에서 2장 목차에서 해당하는 소제목을 찾을 수 있습니다(연습문제인 83쪽과 연습문제 정답인 85쪽은 목차에 나오지 않습니다. 연습문제는 각 장마다 있습니다). 이렇게 부록의 PDF를 쉽게 활용할 수 있습니다.

Chapter 1

WPF에서 Save The Humans 게임을 만들 수 있습니다. 책에서 대체된 페이지는 56페이지에서 91페이지입니다.



게임을 만들며, IDE를 온몸으로 느껴 봅시다

이 책의 첫 번째 프로젝트는 재미있는 게임을 만드는 과정을 안내합니다. 이 프로젝트의 목표는 독자들이 비주얼 스튜디오 IDE를 사용하면서 사용자 인터페이스의 생성과 C# 코드를 작성하는데 익숙해지는 것입니다.

이 책의 55페이지까지 읽고 난 후에, 이 부록의 1장을 읽으면 됩니다. 이 부록은 책의 56페이지에서 91페이지를 대체합니다. WPF 버전의 Save the Humans을 다 만들었다면, 2장으로 넘어가면 됩니다.

이번 장에서 나타나는 화면은 Visual Studio 2013 for Windows Desktop 입니다. (부록을 번역하는 시점에서) 비주얼 스튜디오의 최신 버전이죠. 만약, 여러분이 Visual Studio 2010을 사용한다면, IDE의 탭과 메뉴 옵션이 다를 거예요. 올바른 메뉴 옵션을 찾을 수 있도록 우리가 도와줄 겁니다.

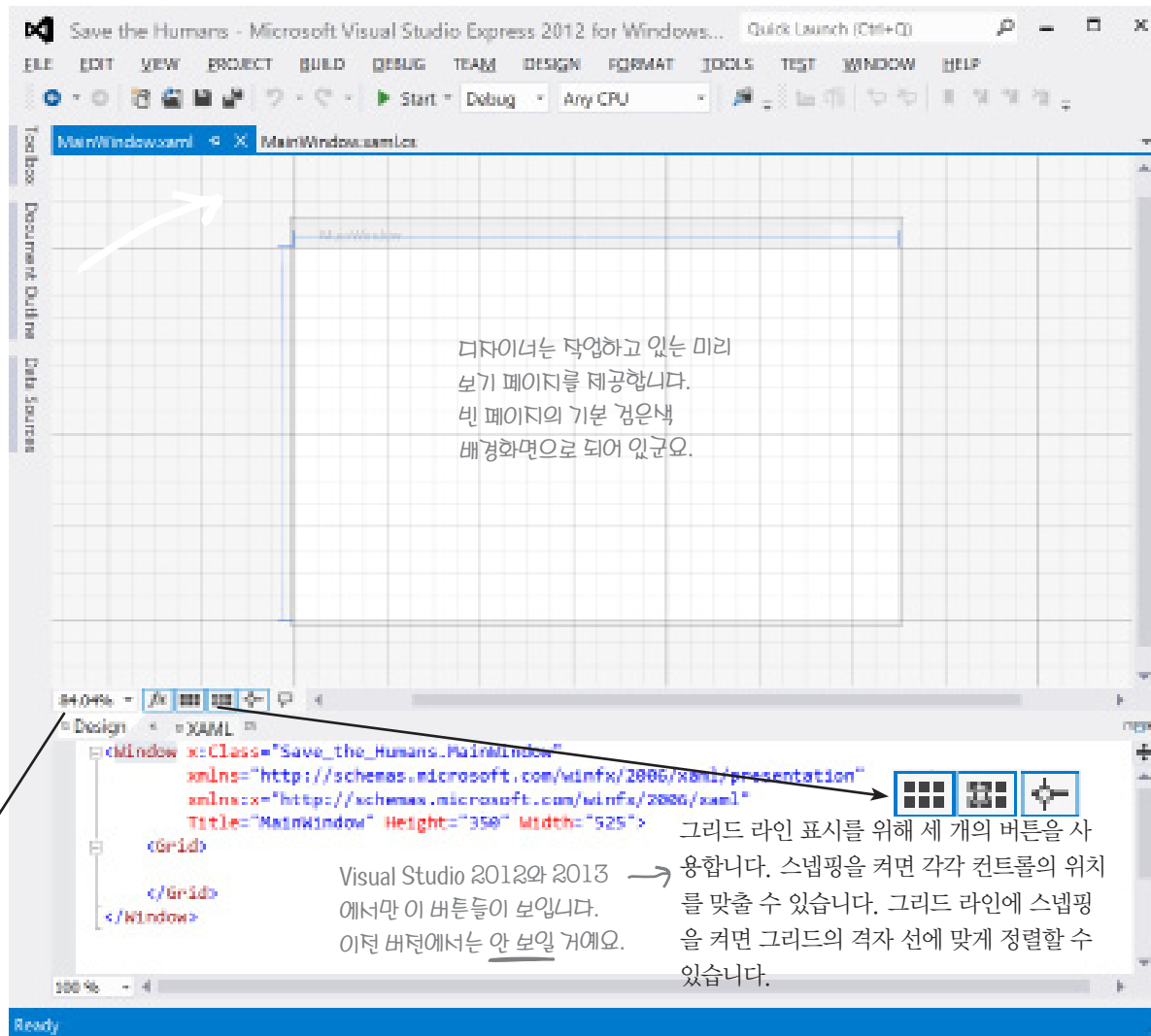
여러분이 C#의 중요한 개념을 학습하는데 집중할 수 있도록 이리저리 페이지를 넘기는 것을 최소화하는데 많은 노력을 기울였습니다. 망에서 55쪽까지 읽고, 부록을 읽으면 됩니다. 2장에서 부록에서 다섯 페이지만 읽고, 종이책을 읽으면 됩니다. 이 책의 3장에서 91쪽까지는 이전 버전의 윈도우에서 구축할 수 있는 데스크톱 응용 프로그램에 집중하고 있습니다.

빈 응용 프로그램으로 시작하기

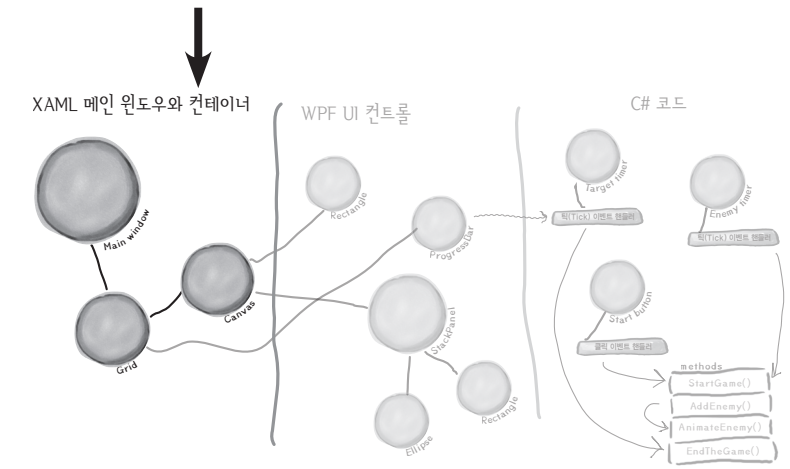
모든 앱은 새 프로젝트로 시작합니다. 파일(File) 메뉴에서 새 프로젝트(New Project)를 선택하세요. 창에서 Visual C# > Windows를 누른 뒤 **WPF 응용 프로그램**을 선택하세요. 이름에 Save the Humans을 입력하고 확인 버튼을 누르면 됩니다.

파일 이름이 ".cs"로 끝나지 않는다면 여러분은 실수로 JavaScript, Visual Basic 혹은 Visual C++로 프로젝트를 생성했을 것입니다. 이 창을 닫고 다시 Visual C#을 선택해서 다시 실행하세요. 프로젝트 이름인 "Save the Humans"를 사용하고 싶으면 이전에 실수로 만들어진 프로젝트를 지우시면 됩니다.

- 1 프로젝트가 생성되면 **디자이너 창**이 보입니다. 솔루션 탐색기(Solution Explorer)에 있는 MainPage.xaml을 더블-클릭합니다. 디자이너의 왼쪽 코너에 있는 줌 드롭-다운을 찾아 모두 맞춤(Fit All)을 선택해서 화면을 축소합니다.

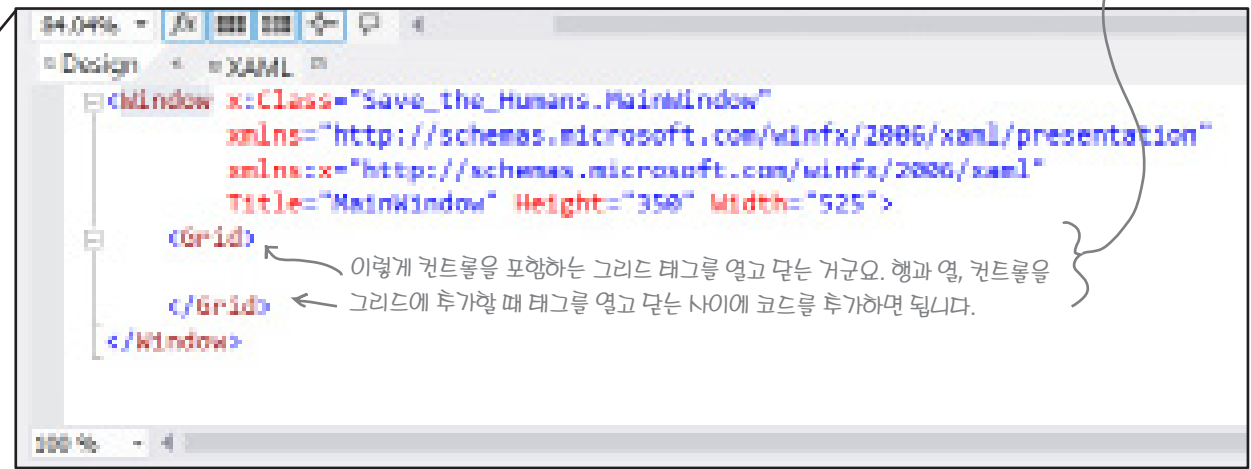


현재 위치



디자이너 아랫부분은 XAML 코드입니다. 디자이너의 "빈" 페이지는 빈 게 아닙니다(XAML 그리드 코드 부분 포함되어 있습니다). 그리드는 HTML 페이지 혹은 워드 문서의 테이블과 비슷합니다. XAML을 이용하여 창을 다양한 크기로 늘리거나 줄여볼 거예요.

그리드를 위한 XAML 코드는 IDE가 생성해 줍니다. 이 부분을 잘 살펴보세요. 행과 열을 추가할 거예요.



- 67%
- 800%
- 400%
- 200%
- 150%
- 100%
- 66.67%
- 50%
- 33.33%
- 25%
- 12.5%
- Fit all
- Fit selection

Visual Studio 2012와 2013에서 이 버튼들이 보입니다. 이전 버전에서는 안 보일 거예요. 그리드 라인 표시를 위해 세 개의 버튼을 사용합니다. 스냅핑을 켜면 각각 컨트롤의 위치를 맞출 수 있습니다. 그리드 라인에 스냅핑을 켜면 그리드의 격자 선에 맞게 정렬할 수 있습니다.

이 프로젝트는 ①부터 ③까지의 과정이 있습니다. 다음 페이지로 넘겨 주세요.

이 프로젝트는 책의 1장과 거의 비슷합니다. 책에서 56~92 페이지를 대체하는 내용을 부록에 담았습니다. 부록에 있는 다른 프로젝트들도 마찬가지로 책을 대체하는 정보를 제공합니다. 책을 이 한 페이지, 한 페이지 다 대체하지는 않지만, 프로젝트를 만드는 데 필요한 모든 정보를 여러분에게 제공합니다.

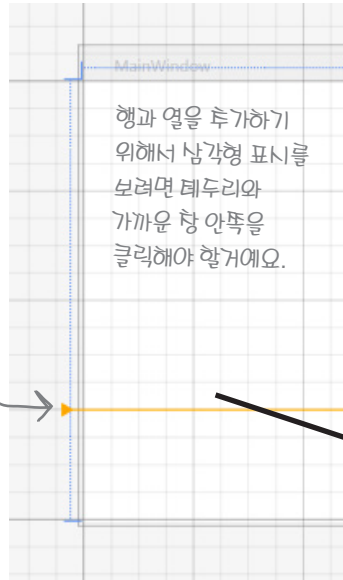


한 걸음, 한 걸음

- ② 빈 페이지의 템플릿에 제목 행을 포함한 2개의 행과 3개의 열을 가진 그리드를 만들어 봅시다. 여기에 플레이 영역이 포함된 중간 부분에 큰 공간이 있습니다. 테두리에서 선과 삼각형이 보일 때 행과 열을 조정해 봅시다.

오렌지 색 삼각형과 선이 나타날 때 그리드 테두리를 조정하면 됩니다.

그리고 여기를 클릭해서 그리드 안에 아래쪽 행을 만드세요.

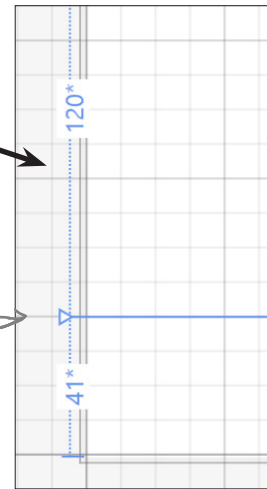


WPF 앱의 어떤 해상도에서라도 크기에 맞게 창이 조절되어 보여야 합니다.

다음 몇몇 페이지에서는 강력한 도구인 비주얼 스튜디오 IDE가 제공하는 많은 기능을 살펴보며 학습할 것입니다. 이 책을 통해 IDE를 사용하며 C#을 탐험하게 됩니다. 이것은 정말 여러분의 뇌를 자극하는 효과적인 방법입니다.

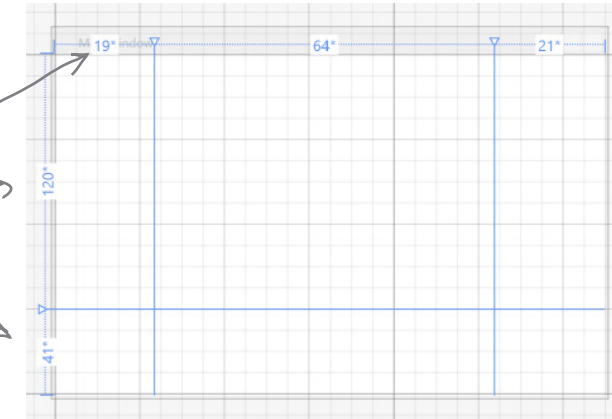
앱은 그리드의 열과 행을 이용하여 자동으로 레이아웃을 윈도우 창의 크기에 맞게 조정합니다.

행이 추가되면 선은 파란색으로 변하고, 테두리에서 행 높이의 숫자가 보입니다. 각 행의 높이는 *에 있는 숫자가 됩니다. 여기에서는 숫자에 신경 쓰지 않아도 됩니다.



- ③ 창의 위쪽 테두리를 이전 페이지에서 했던 것처럼 해 봅시다. 이번에는 왼편과 오른편에 각각 선을 넣어서 두 열을 만들어 봅시다. 행의 높이와 열의 너비는 신경 쓰지 마세요. 여러분이 어디에 클릭하느냐에 따라 크기가 다릅니다. 아래에서 고칠 거예요.

행의 높이와 열의 너비가 다르다고 걱정하지 마세요. 다음 페이지에서 고칠 거예요.



작업이 완료되면, XAML 윈도우 창으로 가서 이전 페이지와 같은 간격의 그리드를 만들어 봅시다. 열 너비와 행 높이가 창의 상단과 측면에 있는 숫자와 일치합니다.



이것이 3번 과정의 왼쪽 열의 너비입니다. 이 너비는 디자이너에서 보이는 너비와 일치합니다. IDE에서 이 XAML 코드를 생성해 줍니다.

바보 같은 질문이란 없습니다

Q: 그리드에 많은 행과 열이 있는데, 회색 선은 뭐죠?

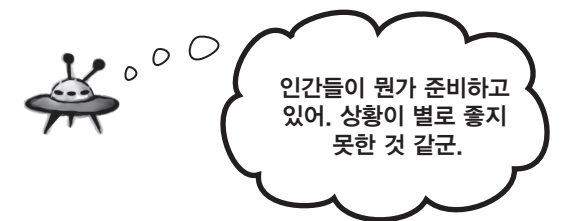
A: 회색 선은 비주얼 스튜디오가 여러분의 컨트롤을 페이지에 맞게 잘 배치할 수 있도록 도와주는 역할을 합니다. 이 회색 선을 버튼으로 켜고 끌 수 있습니다. 비주얼 스튜디오 밖에서 앱을 실행할 때는 디자이너에서 이 선들을 볼 수 없습니다. 그리고 여러분이 클릭을 해서 새로운 행을 추가할 때, XAML 코드가 실제로 변경됩니다. 컴파일하고 실행할 때 추가된 내용이 적용될 것입니다.

Q: 잠시만요. 전 C#을 배우고 싶은데, 왜 XAML을 배우는 데 시간을 투자해야 하나요?

A: 왜냐하면 WPF 앱을 만들때, 대부분 XAML로 디자인된 사용자 인터페이스로 시작합니다. 또한 비주얼 스튜디오는 사용자 인터페이스를 쉽게 만들어 주는 멋진 XAML 에디터가 있습니다. 이 책의 전반에 걸쳐 다양한 유형의 프로그램(XAML을 사용하는 윈도우 스토어 앱과 데스크톱 앱과 XAML을 사용하지 않는 콘솔 응용 프로그램을) 만들면서 C#을 조금 더 깊이 이해할 수 있도록 도와줍니다.

이제 그리드의 행과 열을 추가했습니다.

XAML 그리드는 다른 컨트롤을 담을 수 있는 컨테이너 컨트롤입니다. 그리드는 셀로 정의된 행과 열로 구성되어 있습니다. 각각의 셀은 버튼, 텍스트, 도형을 나타내는 다른 XAML 컨트롤을 담을 수 있습니다. 그리드는 창을 쉽게 꾸며 줍니다. 화면의 크기에 맞춰 행과 열을 다시 조정할 수 있죠.



그리드 설정하기

프로그램은 다양한 크기의 화면에서 작동될 수 있어야 합니다. 그리드는 이런 일을 유연하게 해 주는 좋은 도구입니다. 그리드는 행과 열을 특정 Pixel의 높이에 맞게 설정할 수 있습니다. 그리고 Star(*)를 이용해서, 화면의 크기나 위치에 상관없이 서로 다른 페이지의 적정 크기를 유지할 수 있습니다.

왼쪽 열의 너비를 설정합니다.

1

드롭-다운 메뉴가 나타날 때까지 첫 번째 열 위의 번호를 가리키세요. 픽셀을 Pixel을 선택해서 Star(*)를 자물쇠 표시로 변경합니다. 그리고 숫자를 클릭한 다음 140을 입력하세요. 열의 숫자는 아래와 같아야 합니다

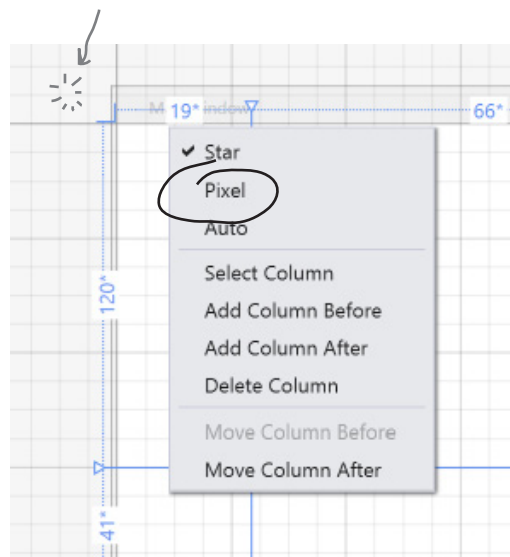


오른쪽 열과 아래쪽 행에도 위와 같이 합니다.

2

오른쪽 열에서 160, 아래쪽 행에서 150을 입력하고, Pixel을 선택하세요.

항의 테두리에서 120*과 19* 같은 숫자가 보이지 않는다면, 디자이너에서 항의 바깥쪽을 클릭해 보세요.



열을 Pixel로 변경할 때, 너비에 비례한 픽셀에서 실제 너비의 픽셀로 바뀝니다.

행과 열을 Pixel로 설정하면 너비와 높이가 고정되도록 설정합니다. Star(*)로 설정하게 되면 그리드의 나머지 부분에 맞게 행과 열이 확대되거나 축소될 수 있습니다. 디자이너에서 너비와 높이 속성을 XAML로 알맞게 변경해 보세요. 만약 너비와 높이의 속성을 제거한다면, 그것은 1*로 설정하는 것과 같습니다.



앱을 꾸미는 데 능숙하지 않아도 괜찮아요.

앞으로 앱을 디자인하는 것에 대해서 많이 이야기 할 거예요. 지금

부터는 게임을 만드는 과정을 살펴봅니다. 이 책의 마지막 부분에서 여러분은 모든 것을 정확하게 이해할 수 있을 거예요.

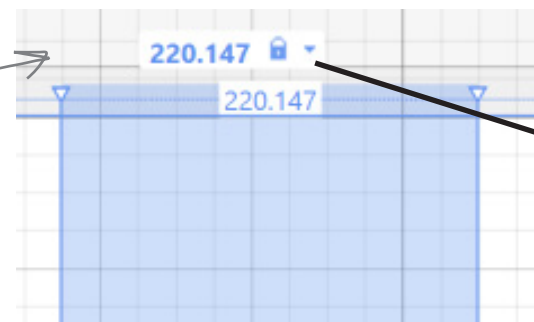
가운데 열과 행을 기본 크기로 만듭니다.

3

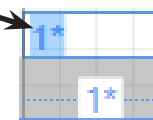
가운데 열의 너비가 1*로 되어 있어야 합니다. 그렇지 않은 경우 중간에 있는 열의 숫자를 클릭해서 1을 입력하세요. 드롭-다운을 사용하지 마세요(Star로 놔두세요). 아래와 같은 그림을 볼 수 있습니다. 그런 다음 IDE가 자동으로 크기를 조정하지 않았는지, 다른 열을 한번 확인해 보세요. 만약 다른 열의 크기가 변경 되었다면 과정 1, 2에서 설정했던 너비로 바꿔줍니다.

XAML과 C#은 대소문자를 구분합니다! 예제 코드와 대문자와 소문자가 일치하는지 확인하세요.

실수로 가운데 열의 너비를 Pixel로 바꿨을 경우, 다시 1*로 바꿀 수 있습니다.



1*을 입력할 때, IDE는 기본 너비에 맞게 열의 크기를 설정합니다. 다른 열도 그렇게 적용될 것입니다. 만약 그렇다면 160 픽셀로 다시 바꿔 두세요.



4

XAML 코드를 봅시다.

그리드의 코드 부분으로 갑니다. 그리고 XAML 윈도우 창에서 코드가 잘 만들어 졌는지 확인해 봅시다.

```
<Window x:Class="Save_the_Humans.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="140"/>
<ColumnDefinition/>
<ColumnDefinition Width="160"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition/>
<RowDefinition Height="150"/>
</Grid.RowDefinitions>
</Grid>
</Window>
```

꼭대기에 있는 <Grid> 태그는 그리드 전체를 의미합니다. 그 아래에 그리드의 일부들이 있죠.

XAML에서 그리드의 열이 정의되는 부분입니다. 3개의 열과 2개의 행을 추가해서, 3개의 ColumnDefinition 태그와 2개의 RowDefinition 태그가 있습니다.

디자이너를 이용해서 아래 행의 높이를 150으로 설정했었죠.

행과 열의 드롭-다운을 설정해서 너비와 높이의 속성을 변경했었죠.

잠시 후, 그리드에 컨트롤을 추가할 것입니다. 행과 열을 정의한 후 여기에 컨트롤을 배치할 거예요.

Visual Studio 2010을 쓴다면 IDE가 다르게 보입니다. 열 크기 부분에 마우스를 올려 놓으면, Pixel과 Star를 선택하는 이런 상자가 보입니다.



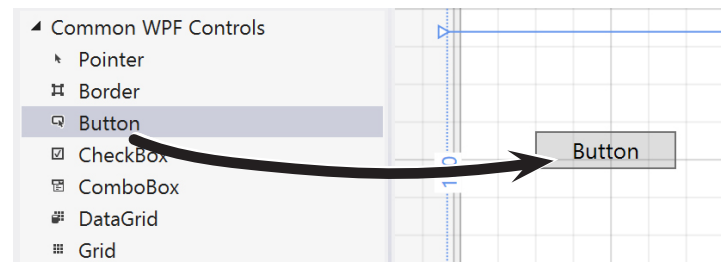
구 버전 IDE의 디자이너에서 열의 크기를 바꿀 순 있지만, 우리가 실습했던 것처럼 잘 되지 않습니다. 만약 구 버전의 IDE를 사용한다면, 행과 열을 만든 후 XAML에서 행과 열을 수동으로 입력하는 것을 추천합니다.

컨트롤을 추가해 봅시다

그리드에 컨트롤 추가하기

버튼과 텍스트, 사진, 프로그래스 바, 슬라이더, 드롭-다운과 메뉴를 여러분의 앱에 어떻게 채워 넣을까요? 이것들을 컨트롤(control)이라 부르죠. 이제 컨트롤을 앱에 추가할 때가 왔습니다. 그리드의 행과 열로 이루어진 셀 안에 컨트롤을 추가해 볼까요?

- ① 도구 상자의 **Common WPF Controls** (Common WPF 컨트롤)을 펼친 후 **Button** 제일 **왼쪽 아래의 셀**로 끌어 놓습니다.

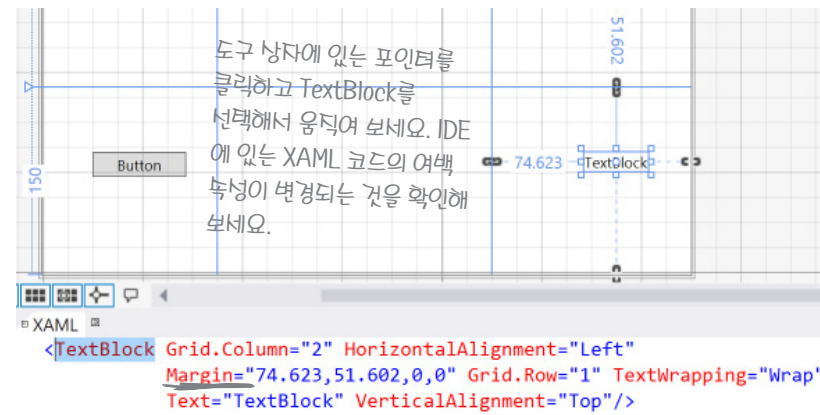


디자이너 아래로 가서 IDE가 생성한 XAML 태그를 봅시다. 여러분이 컨트롤을 끌어놓은 위치에 따라 여백 숫자가 다르게 보이고, 속성의 순서가 다를 수도 있습니다.

XAML 버튼 코드는 여는 태그와 같이 여기서부터 시작합니다.

```
<Button Content="Button" HorizontalAlignment="Left"
        Margin="40,52,0,0" Grid.Row="1" VerticalAlignment="Top" Width="75" />
```

- ② Drag a **TextBlock** 를 그리드의 맨 **오른쪽 아래의 셀**에 끌어 놓습니다. XAML 코드는 아래와 같이 보일 거예요. 컨트롤이 어느 행과 열에 배치되어 있는지 어떻게 알 수 있을까요?



```
<TextBlock Grid.Column="2" HorizontalAlignment="Left"
           Margin="74.623,51.602,0,0" Grid.Row="1" TextWrapping="Wrap"
           Text="TextBlock" VerticalAlignment="Top"/>
```

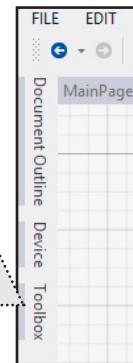
XAML을 읽기 쉽게 하려고 들여쓰기를 했어요. 여러분도 직접 해 보세요!

IDE에서 도구상자가 안 보이면, 보기(View) 메뉴로 가서 도구상자를 열 수 있습니다. 자동으로 숨겨지지 않도록 고정 핀을 사용하세요.

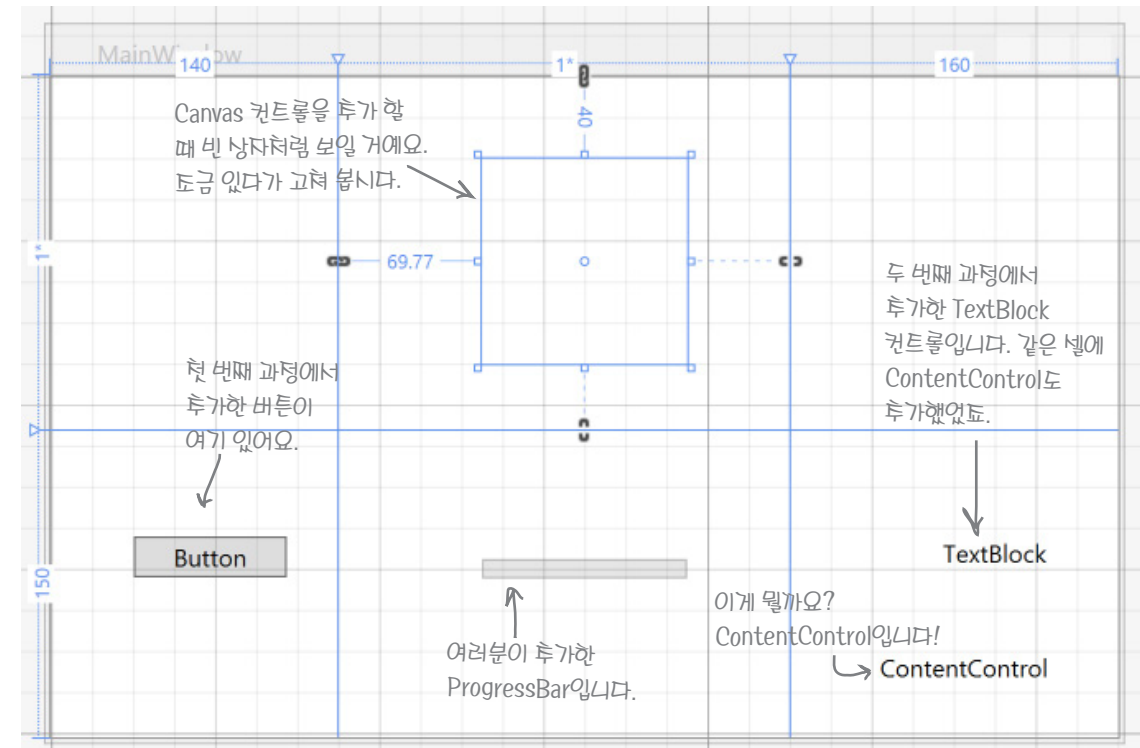


이것이 속성입니다. 속성은 이름과 =, 값으로 이루어져 있습니다(이름=값).

만약 도구 상자가 보이지 않는다면, IDE의 위쪽 왼쪽 모퉁이에 있는 도구 상자의 글씨를 클릭하세요. 만약 여기에도 없다면 보기(View) 메뉴에서 도구 상자(Toolbox)를 클릭하면, 도구 상자를 볼 수 있습니다.



- ③ 다음 도구 상자에 있는 **ALI WPF Controls**(All WPF 컨트롤)을 펼칩니다. 맨 아래의 중앙 셀에 **ProgressBar** 를 끌어 놓습니다. 그리고 **ContentControl** 을 맨 아래 오른쪽 셀에 놓습니다 (이미 배치한 TextBlock 밑에 놓으면 됩니다). 그리고 **Canvas** 를 중앙에 놓습니다. 이제 여러분의 창에 이렇게 컨트롤이 배치되어 있어야 합니다(아래 그림과 다르다고요? 걱정하지 마세요. 같이 수정해 봅시다).



- ④ 중앙에 있는 Canvas 컨트롤을 선택합니다(선택이 안 되면 도구상자의 포인터를 눌러서 선택해 보세요). XAML 창에는 아래와 같이 보입니다.

```
<Canvas Grid.Column="1" HorizontalAlignment="Left" Height="100" ...
```

Canvas 컨트롤에 대한 XAML 태그입니다. 이것은 <Canvas 로 시작하고 />로 끝납니다. 그 사이에 Grid.Column="1" (Canvas를 가운데 열에 배치)과 Height="100" 속성이 있습니다. XAML창에 그리드를 클릭해서 다른 컨트롤을 선택해 보세요.

이 버튼을 눌러 보세요. 문서 개요(Document Outline) 창이 나타납니다. 어떻게 사용하든지 아니냐요? 나중에 배워 봅시다.



도구상자에서 컨트롤을 끌어놓을 때, IDE는 자동으로 여러분이 끌어놓은 위치의 XAML 코드를 생성합니다.

속성값을 넣어 봅시다

속성값으로 컨트롤이 변하는 모습을 봅시다

비주얼 스튜디오 IDE는 컨트롤을 미세하게 조절할 수 있습니다. IDE의 속성 창은 컨트롤의 외형과 동작을 변경할 수 있습니다.

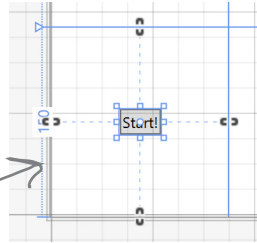
1 Button 컨트롤의 텍스트를 바꿔 봅시다.

여러분이 그리드로 끌어 놓은 Button 컨트롤을 오른쪽 클릭한 뒤, 메뉴에서 텍스트 편집(Edit Text)을 선택합니다. Start!로 변경한 후 XAML의 Button 태그를 확인해 보세요.

```
<Button Content="Start!" HorizontalAlignment="Left" VerticalAlignment="Top" ...
```

Button 컨트롤에 텍스트를 입력할 때, XAML의 Content 속성이 업데이트됩니다.

텍스트를 편집할 때 Esc 키를 눌러 편집을 끝내면 됩니다. 컨트롤들도 이렇게 하시면 됩니다.



속성 창의 이름 칸에 startButton을 입력합니다.

2 속성 창에서 Button을 수정합니다.

IDE에서 Button을 선택하고, 아래의 오른쪽 코너에 속성 창으로 갑니다. 컨트롤의 이름을 startButton으로 변경하고, 셀의 중앙에 컨트롤을 배치해 봅시다. Button의 속성이 맞는지 확인하려면, **마우스 오른쪽 버튼을 클릭해서 소스보기(View Source)를 선택**하면 됩니다. XAML 창에서 <Button> 태그를 확인할 수 있습니다.

이 작은 네모난 상자는 속성이 설정되었는지 알려 줍니다. 색깔된 상자는 속성이 변경되었다는 뜻이고, 빈 상자는 기본값으로 설정되어 있다는 것을 의미합니다.

오른쪽 메뉴를 이용해서 텍스트 편집을 눌러 Button의 텍스트를 변경했을 경우, IDE는 이 속성의 내용을 업데이트합니다.

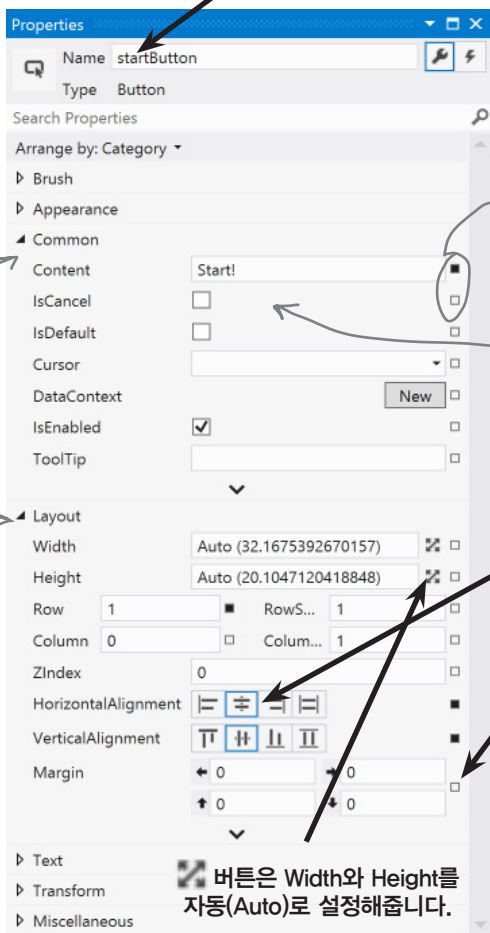
HorizontalAlignment와 VerticalAlignment 속성의 버튼을 컨트롤을 셀의 중앙에 배치하세요.
 구 버전의 IDE에서는 아이콘 대신 "Center"가 있습니다.

페이지에 Button을 끌어 놓을 때 IDE는 Margin(여백) 속성을 이용해서 셀의 정확한 위치에 놓습니다. 작은 네모 상자를 클릭하고, 메뉴에서 다시 설정(Reset)을 선택하면 Margin이 0값으로 재설정됩니다.

```
<Button x:Name="startButton" Content="Start!" Grid.Row="1" HorizontalAlignment="Center" VerticalAlignment="Center"/>
```

속성들이 조금 다르게 정렬되어 있어도 괜찮아요!

공용과 레이아웃 섹션을 확장합니다.



버튼은 Width와 Height를 자동(Auto)로 설정해줍니다.

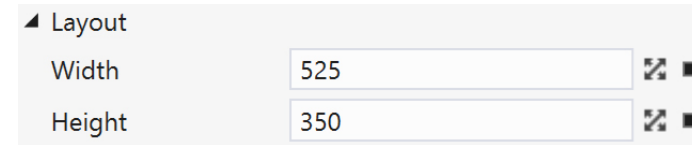
이전 내용을 편집(Edit) > 실행 취소(Undo 혹은 Ctrl-z)로 돌릴 수 있습니다. 실행 취소를 몇 번 해 보면 취소한 만큼의 이전 내용으로 돌아갑니다. 여러분이 컨트롤을 잘못 선택했을 경우 편집 메뉴에 있는 몇 항목들이 비활성화됩니다. 또한 Esc키를 눌러 컨트롤의 선택을 해제할 수 있습니다. 만약 StackPanel과 Grid와 같은 컨테이너 안에 컨트롤이 선택되어 있다면, Esc키로 컨테이너를 선택할 수 있습니다. Esc키를 여러 번쳐야 하는 경우도 있죠.

3 창의 크기와 제목을 바꿔봅시다.

아무 컨트롤을 선택한 다음, XAML 편집기에서 <Window> 태그가 보일 때까지 Esc키를 누릅니다.

```
<Window x:Class="Save_the_Humans.MainWindow" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="MainWindow" Height="350" Width="525">
```

XAML 편집기에서 <Window>를 클릭합니다. <Window> 태그에 Height와 Width 속성이 있습니다. 창에서 이 속성들을 변경했을 때 잘 반영되는지 살펴봅시다.



TextBlock과 ContentControl은 그리드의 오른쪽 아래의 셀에 있습니다.

Width와 Height를 각각 1000과 700으로 설정해 줍니다. 창은 바로 입력한 크기를 반영하죠. 디자이너에서 전체 창을 보기 위해서 확대/축소 드롭-다운에서 "모두 맞춤(Fit all)" 옵션을 선택합니다. 다른 행과 열의 픽셀이 유지되면서, 크기가 창에 맞게 바뀐 것을 볼 수 있습니다. 속성 창의 공용(Common) 섹션을 확장해서, **Title 속성**에 Save The Humans를 입력합니다. 창의 제목이 바뀐 걸 볼 수 있습니다.

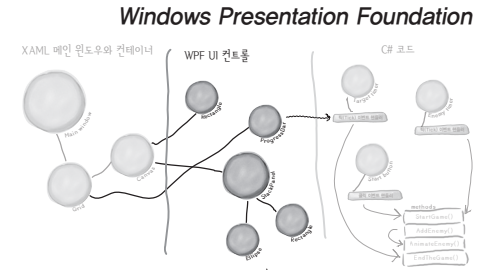
4 TextBlock을 수정해서 텍스트와 글꼴 크기를 바꿔 봅시다.

마우스 오른쪽 버튼을 눌러 텍스트 편집을 이용해서 TextBlock를 Avoid These로 변경합니다(텍스트 입력이 끝나면 Esc키를 눌러 편집을 끝냅니다). 속성 창의 텍스트 섹션을 확장해서 글꼴 크기에 18px를 입력합니다. 이렇게 설정하면 텍스트가 두 줄로 되는 경우가 있습니다. 그렇다면, TextBlock 컨트롤을 옆으로 조금 늘려줍니다.

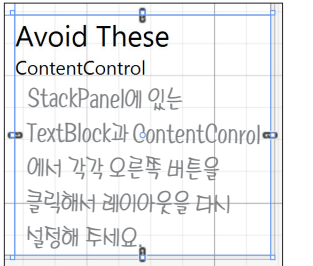
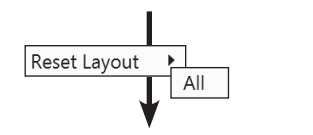
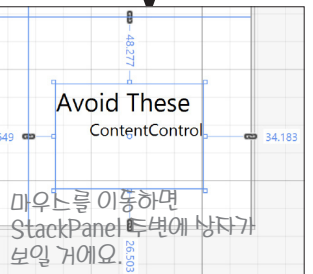
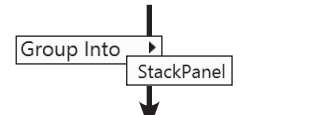
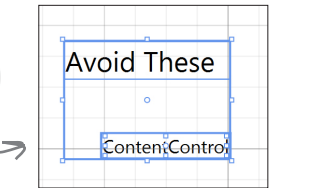
5 StackPanel을 사용해서 TextBlock과 ContentControl을 묶어 봅시다.

셀에서 TextBlock을 위로 배치하고 ContentControl을 아래로 배치합니다. Shift키로 두 컨트롤을 선택한 후 마우스 오른쪽 버튼을 클릭한 후, 팝업 메뉴에서 그룹으로 묶기(Group Into) > StackPanel을 선택합니다. 이제 페이지에 새로운 StackPanel 컨트롤이 추가되었습니다. 두 컨트롤을 클릭해서 StackPanel을 선택할 수 있습니다.

StackPanel은 Grid, Canvas와 많은 부분이 비슷합니다. 공통점은 다른 컨트롤을 묶어 주죠(이것을 컨테이너라고 부릅니다). 컨테이너는 페이지에서 보이지 않습니다. 여기에서는 셀에서 위 부분의 TextBlock과 아랫부분의 ControlPanel을 묶은 후 StackPanel을 만들면서 Grid의 셀을 대부분 채워 줍니다. StackPanel을 선택한 후, 마우스 오른쪽 버튼을 눌러 레이아웃 > 모두 다시 설정을 클릭하면 속성들이 빠르게 다시 설정됩니다(셀의 수평과 수직의 레이아웃이 Stretch로 정렬됩니다). 그리고 TextBlock과 ContentControl에 오른쪽 버튼을 클릭해서 레이아웃을 모두 다시 설정해 줍니다. ContentControl을 설정할 때, 수평과 수직 정렬을 이용해 컨트롤을 가운데로 배치합니다.



지금 여기에 있어요!

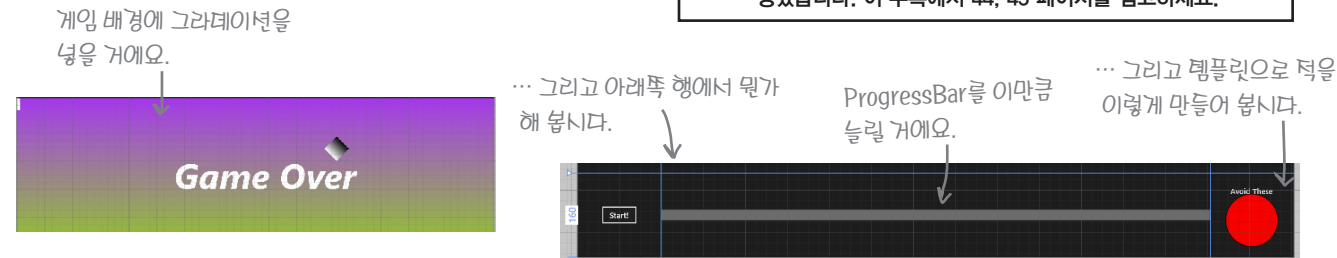


지금 여기에요 ▶

컨트롤로 게임 동작하게 하기

컨트롤은 제목과 캡션과 같은 보이는 것들만 있는 게 아닙니다. 컨트롤은 게임이 동작할 수 있도록 하는 중심 역할을 합니다. 컨트롤을 추가해서 플레이어들이 게임을 할 수 있도록 해 봅시다. 여러분이 다음에 만들 것은 다음과 같습니다.

구 버전의 비주얼 스튜디오에서 색상을 편집하는 사용자 인터페이스가 달라도, 색상을 편집할 수 있습니다. 문서 개요 창에서 원시적인 방법으로 편집할 수 있지만, Visual Studio 2010에서는 템플릿을 시각적으로 생성하는 게 쉽지 않습니다. 구 버전의 IDE에서 템플릿을 생성하는 가장 쉬운 방법은 헤드 퍼스트 랩에서 소스 코드를 내려 받아서 <Windows.Resources>에서 </Windows.Resources> 태그까지 복사합니다. 그리고 이것을 <Grid> 시작 태그 위에 붙여 넣습니다. 웹 사이트의 WPF 폴더에서 내려 받아야 합니다. 그리고 ContentControl을 선택한 뒤, 속성 창에서 Template 속성을 EnemyTemplate로 설정합니다. 적군은 약탈한 외계인 같이 생겼습니다. 이 부록에서 44, 45 페이지를 참고하세요.

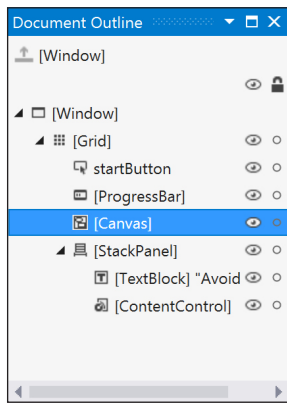


1 ProgressBar를 수정해 봅시다.

그리드의 중앙 아랫부분의 셀에서 ProgressBar를 마우스 오른쪽 클릭합니다. 레이아웃 > 모두 다시 설정을 선택합니다. 그리고 속성 창의 레이아웃 섹션으로 가서 Height에 20을 입력합니다. XAML 창에서 레이아웃과 관련된 속성을 찾아서 Height가 추가되었는지 확인해 봅시다.

```
<ProgressBar Grid.Column="1" Grid.Row="2" Height="20"/>
```

보기(View) > 다른 창 (Other Windows) > 문서 개요(Document Outline)에서 볼 수 있습니다.



2 Canvas 컨트롤을 게임 플레이 영역으로 바꿔봅시다.

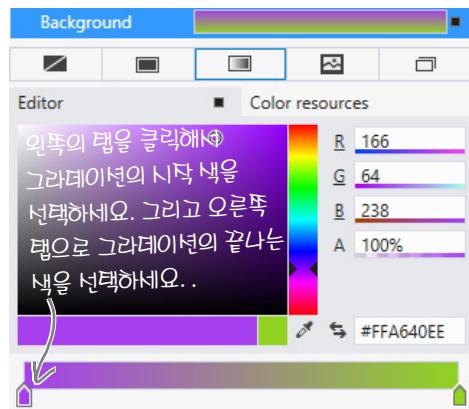
중앙에 Canvas 컨트롤 놓은 거 기억하시죠? 그런데 지금 찾기가 힘드네요. 도구 상자에서 Canvas Control을 놓을 때, 잘 보이지 않습니다. 그러나, 찾기 쉬운 방법이 있습니다. XAML 창에서 툴 팁 박스에서 문서 개요(Document Outline)라고 쓰인 버튼을 클릭합니다. 그리고 [Canvas] 버튼을 클릭해서 Canvas 컨트롤을 선택합니다.

Canvas 컨트롤을 선택했으면, 속성 창의 이름 칸에 "playArea"를 입력합니다.

이름이 변경되면 문서 개요 창에 [Canvas] 대신 playArea라고 뜹니다.

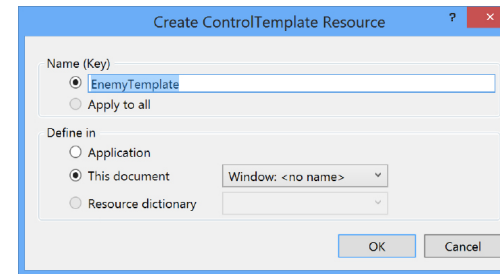
Canvas 컨트롤에 이름을 붙인 후, 문서 개요 창을 닫으세요. 그리고 속성 창에 있는 [Stretch] 버튼을 눌러 수평과 수직으로 정렬하고, Margin(여백)을 다시 설정합니다. 그리고 Width와 Height에 있는 [Auto] 버튼을 각각 누릅니다. 그리고 Column에 0과 ColumnSpan(Column의 오른쪽)에 3을 입력합니다.

마지막으로, 속성 창의 브러시(Brush) 섹션에서 Canvas에 그래데이션을 주기 위해 [Gradient] 버튼을(그래데이션 브러시)을 클릭합니다. 편집기 아랫부분의 왼쪽과 오른쪽 탭을 클릭해서 그래데이션의 시작 색과 끝나는 색을 선택하고 색상을 클릭하세요.



3 적 템플릿을 생성해 봅시다.

게임 화면에서 같은 모양의 많은 적군들이 움직여야 합니다. 다행히도, XAML에서 많은 컨트롤을 같은 적군 모양으로 만들어 주는 템플릿(Template)을 제공해 줍니다. 다음, 문서 개요(Document Outline) 창에서 [StackPanel]을 펼친 후, [ContentControl]에서 마우스 오른쪽 클릭합니다. 그리고 템플릿 편집(Edit Template) > 빈 항목 만들기(Create Empty...) 메뉴를 선택합니다. EnemyTemplate으로 이름을 입력하고 확인을 누르면, XAML 창에 템플릿이 추가됩니다.



움직이는 적들은 여기서 보이지 않습니다. 적군이 표시되도록 컨트롤을 추가하고 크기를 설정하기 전까지 디자이너 창에는 어떤 템플릿도 표시되지 않습니다. 뭔가 잘못됐다고 걱정하지 마세요. 취소(Undo)하고 다시 하면 됩니다.

템플릿이 선택되지 않을 경우, 문서 개요 창에서 그리드를 선택하세요.

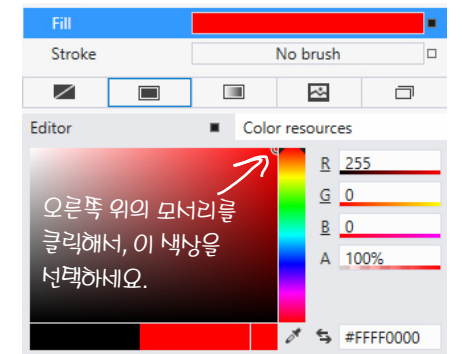
현재 새로 만든 템플릿이 선택되어 있습니다. 그런데 템플릿이 보이지 않네요. 다음 과정에서 템플릿을 보이게 합니다. 만약 컨트롤 템플릿의 바깥 부분을 클릭했다면, 여러분은 문서 개요(Document Outline)를 열어 다시 선택할 수 있습니다. ContentControl을 오른쪽 클릭해서 템플릿 편집(Edit Template) > 현재 항목 편집(Edit Current)을 선택하면 됩니다.

4 적 템플릿을 편집해 봅시다.

빨간색 원을 템플릿에 추가해 봅시다.

디자이너에서 원형이 보일 때까지 아무데나 클릭하지 마세요. 템플릿이 선택되는 것을 유지하려고 합니다.

- ★ 도구 상자에 있는 Ellipse 를 더블 클릭해서 원을 추가하세요.
- ★ Ellipse의 Height와 Width의 속성을 100으로 설정하세요. 그러면 셀에서 원이 보일 거예요.
- ★ Margin, HorizontalAlignment, VerticalAlignment 속성의 옆에 작은 네모 상자를 눌러, 다시 설정을 선택합니다.
- ★ 속성 창의 브러시 섹션으로 가서 단색 브러시(Solid-color brush) 를 선택합니다.
- ★ 긴 색상 바 안의 표식을 윗부분으로 끌어서, 원형에 빨간색을 칠합니다. 그런 다음 정사각형의 색상 선택 부분을 클릭해서 오른쪽 위의 코너로 마우스를 끌어 놓습니다.



ContentControl의 XAML 코드는 이제 이렇게 보일 겁니다.

```
<ContentControl Content="ContentControl" Grid.Column="2" HorizontalAlignment="Center" Grid.Row="2" VerticalAlignment="Center" Style="{StaticResource EnemyTemplate}"/>
```

XAML 윈도우 창을 스크롤해서 EnemyTemplate가 정의된 것을 확인해 보세요. AppName 리소스가 밑줄과 같아야 합니다.

5 문서 개요를 이용해 StackPanel, TextBlock, Grid 컨트롤을 수정해 봅시다.

문서 개요로 돌아옵니다(문서 개요 창의 윗부분에 EnemyTemplate (ContentControl Template) 이 보이면, 를 클릭해서 Window로 돌아옵니다). 컨트롤을 선택해서, 수평과 수직이 중앙으로 정렬되어 있는지, 여백이 설정되었는지 확인합니다. 마찬가지로 TextBlock도 확인합니다. 그리고 속성 창에서 색상 편집기를 통해서 Foreground 속성을 흰색으로 설정해 줍니다.



마지막으로 Grid를 선택합니다. 속성 창의 브러시(Brush) 섹션을 열어서, 단색 브러시 를 선택한 후, Background를 검정색으로 만들어 주세요.

이 부분을 클릭하면 색상 편집기가 나옵니다. TextBlock을 흰색으로 바꿔주세요.

Document Outline → IDE의 오른쪽 부분에서 문서 개요를 클릭해서 열 수 있습니다.

6 Canvas에 사람을 추가해 봅시다.

사람을 추가하기 위한 두 가지 옵션이 있습니다. 첫 번째는 다음 세 단계를 따라 하면 됩니다. 두 번째는 조금 더 빠른 방법인데 XAML 코드에 4줄만 추가하면 됩니다. 여러분이 선택하세요!

Ellipse 컨트롤을 Canvas에 추가하기 위해서 셀 중앙의 Canvas 컨트롤을 선택한 후, 도구 상자의 모든 XAML 컨트롤 섹션에서 Ellipse를 더블-클릭합니다. Canvas 컨트롤을 다시 선택해서 Rectangle을 더블 클릭합니다. Rectangle은 Ellipse의 바로 위에 추가될 것입니다. Rectangle을 Ellipse 아래로 끌어 놓습니다.

Shift 키를 누른 채 Ellipse를 클릭한 후, 두 컨트롤 모두 선택되게 합니다. 선택한 부분에서 오른쪽 버튼을 눌러 **그룹으로 묶기(Group Into)** > StackPanel을 선택합니다. Ellipse를 선택해서 단색 브러시 속성을 이용해서 흰색으로 바꿉니다. 그리고 Width와 Height 속성을 10으로 합니다. 다음 Rectangle를 선택해서, 역시 흰색으로 바꾼 후 Width를 10, Height를 25로 합니다.

문서 개요 창에서 playArea에 있는 StackPanel을 선택하세요(속성 창에서 형식이 StackPanel인지 확인하세요). 여백을 다시 설정한 후, **Width와 Height 속성에 있는 [] 버튼(자동으로 설정)을 클릭하세요.** 그리고 이름 칸에 human을 입력합니다. 여기에 생성된 XAML 코드가 있습니다.

```
<StackPanel x:Name="human" Orientation="Vertical">
  <Ellipse Fill="White" Height="10" Width="10"/>
  <Rectangle Fill="White" Height="25" Width="10"/>
</StackPanel>
```

만약 XAML 탭에 이 코드를 입력한다면 </Canvas> 태그 바로 위에 코드를 입력하세요. 이것은 Canvas 안에 사람이 있다는 것을 의미합니다.

Ellipse와 Rectangle의 Stroke 속성이 검은색으로 설정되어 있을 겁니다(만약 보이지 않는다면, 한번 추가해 보세요. 무슨 일이 일어날까요?)

문서 개요 창으로 돌아와 새로운 컨트롤이 어떻게 나타나는지 확인해 봅시다.



과정 2에서 Canvas 컨트롤의 이름을 playArea로 해서, 문서 개요 창에 이렇게 보입니다. 컨트롤에 마우스를 올려보세요.

만약 human이 playArea 아래에 있지 않다면, human을 마우스로 끌어서 위의 그림처럼 만들어 주세요.

7 "Game Over" 텍스트를 추가합니다.

게임이 끝났을 때, 게임이 끝났다는 메시지를 표시해야 합니다. TextBlock 컨트롤을 추가해서 글꼴을 설정하고 이름을 입력해 봅시다.

- ★ Canvas를 선택하고 도구상자의 TextBlock을 끌어다 놓습니다.
- ★ 속성 창의 TextBlock의 이름 칸에 gameOverText를 입력합니다.
- ★ 속성 창의 텍스트 섹션을 펼친 후 글꼴을 Arial Black, 글꼴 크기를 100px로 하고, Bold와 기울임꼴을 선택합니다.
- ★ TextBlock을 끌어서, Canvas의 중앙에 놓습니다.
- ★ 그리고, 텍스트에 Game Over를 입력합니다.

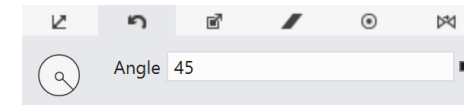
Canvas에 컨트롤을 끌어놓을 때 위와 왼쪽의 속성값은 끌어놓은 위치 값으로 설정됩니다. 만약 위와 왼쪽의 속성값을 바꾸고 싶으면 컨트롤을 움직이면 됩니다.

8 사람을 끌어 넣을 타깃 포탈을 추가해 봅시다.

Canvas에 마지막 컨트롤을 추가합니다. 플레이어가 사람을 포탈에 끌어 넣을 겁니다(Canvas가 어디에 있어도 상관 없습니다).

Canvas 컨트롤을 선택해서 Rectangle 컨트롤을 위에 놓습니다. 속성 창의 브러시 섹션에서 [] 버튼을 클릭하여 그라데이션을 줍니다. 그리고 Height와 Width 속성을 50으로 합니다.

Rectangle을 45도로 회전해서 다이아몬드 모양으로 바꿔봅시다. 속성 창의 변형 섹션을 열어서 [] 회전 버튼을 눌러 Angle에 45를 입력합니다.



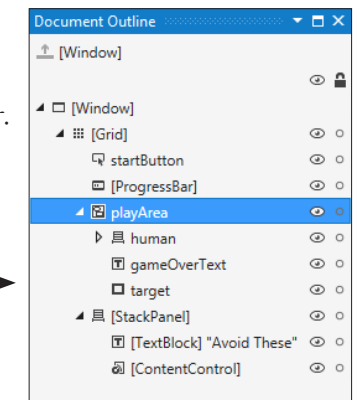
마지막으로, 속성 창의 이름 칸에 target을 입력합니다.

9 몇 가지 사항을 점검해 봅시다.

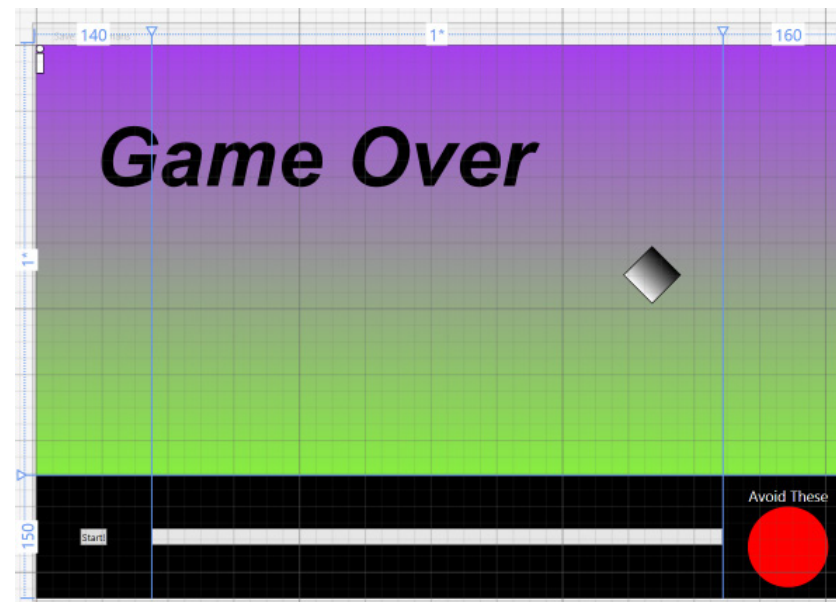
문서 개요 창을 열어 human의 StackPanel, gameOverText의 TextBlock, target의 Rectangle이 PlayArea의 Canvas([Grid] 아래에 있음) 아래에 있는지 확인해 봅시다. 그리고 playArea의 Canvas 컨트롤을 선택해서, Height와 Width가 자동(Auto)으로 설정되어 있는지 확인해 봅시다. 이 모든 사항은 찾기 힘든 버그를 발생시킵니다. 여러분의 문서 개요 창은 왼쪽 그림과 같아야 합니다.

마지막으로, 속성 창의 이름 칸에 target을 입력합니다.

축하합니다. 메인 윈도우를 완성했습니다!



human과 gameOverText, target 컨트롤이 playArea 컨트롤 아래에 있는지, playArea를 펼쳐서 확인했습니다. 여기에서 컨트롤의 눈서는 상관 없습니다(컨트롤들을 위 아래로 움직여 보세요. 대신 위의 그림처럼 들여쓰기가 맞아야 합니다). 컨테이너 컨트롤 안에 컨트롤의 포함 관계를 문서 개요 창을 통해 알 수 있습니다.



나는 누구일까요?

지금까지 사용자 인터페이스를 만들어 보면서 컨트롤이 무슨 역할을 하는지에 대한 감각을 키웠습니다. 컨트롤의 속성이 무엇을 하는지, IDE의 속성 창에 어디에 있는지 찾아서 연결해 보세요.

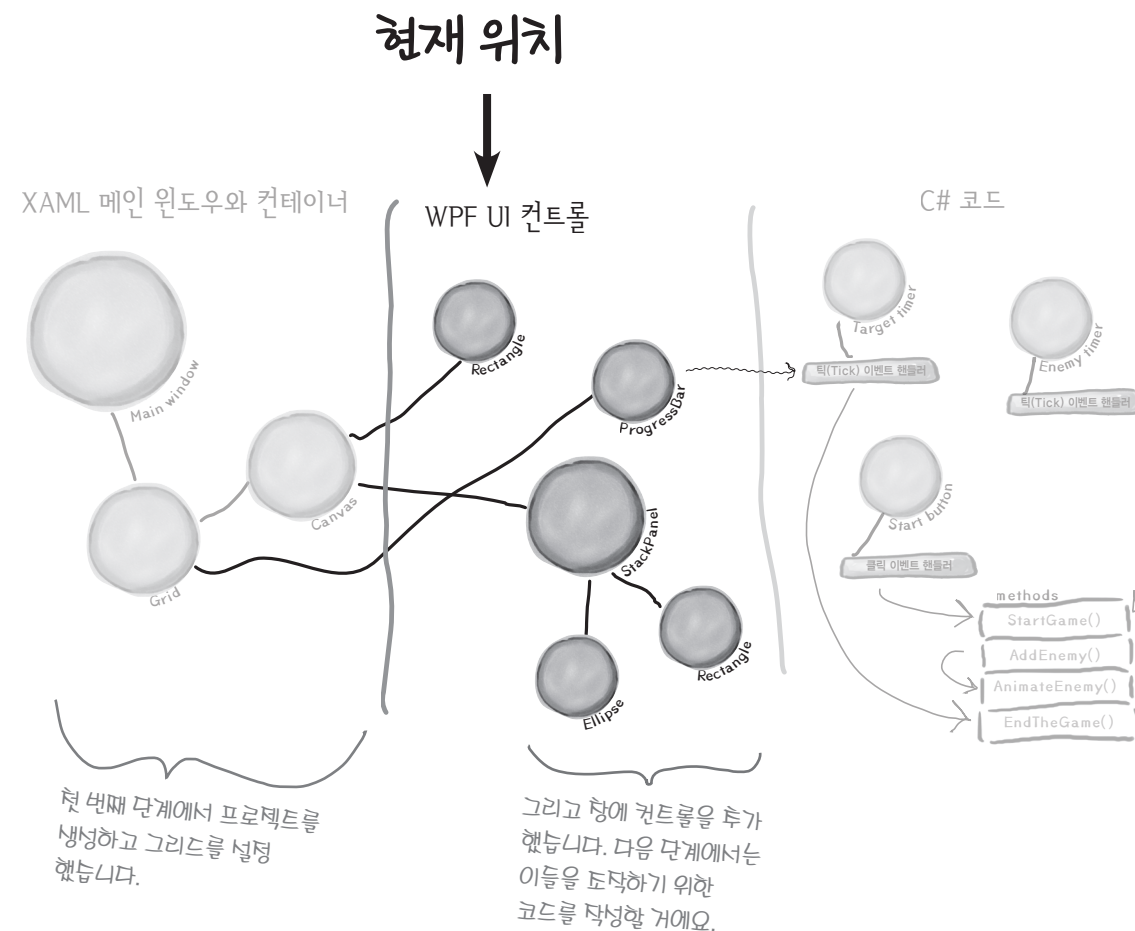
XAML 속성	IDE의 속성창 어디에서 찾을 수 있나요?	무엇을 하죠?
Content	속성 창 꼭대기	컨트롤의 크기를 알 수 있습니다.
Height	▶ Brush	컨트롤의 각도를 설정할 수 있어요.
Rotation	▶ Appearance	C# 코드로 특정 컨트롤을 조작하기 위해 사용합니다.
Fill	▶ Common	컨트롤의 색상
x:Name	▶ Layout	컨트롤 안에 보이는 텍스트를 바꿀 때 사용합니다.
	▶ Transform	

해답은 79페이지에 있습니다. →

힌트: 속성을 찾기 위해 윈도우 속성 창에 있는 검색 속성을 이용할 수 있습니다. 검색한 속성 중의 일부는 모든 컨트롤의 속성에 존재하지 않습니다.

게임 만들 준비가 됐습니다

이제 코딩을 할 수 있습니다. 여러분은 창의 기초가 되는 그리드를 설정하고, 컨트롤을 추가했습니다.



비주얼 스튜디오는 창을 꾸미기 위한 유용한 도구를 제공해 줍니다. XAML 코드를 자동으로 생성해 주죠. 이것로 뭔가 하기엔 부족합니다. 이제 여러분이 나서야 할 차례입니다.

다음에 해야 할 것

이제부터 재미있는 부분입니다. 게임이 동작하는 코드를 추가하기 위한 세 가지 단계가 있습니다. 첫 번째 적군을 움직이게 한 다음 플레이어와 상호동작할 수 있도록 만듭니다. 마지막으로 게임을 조금 다듬어서 멋지게 만들어 봅시다.

먼저, 적을 움직여 봅시다.

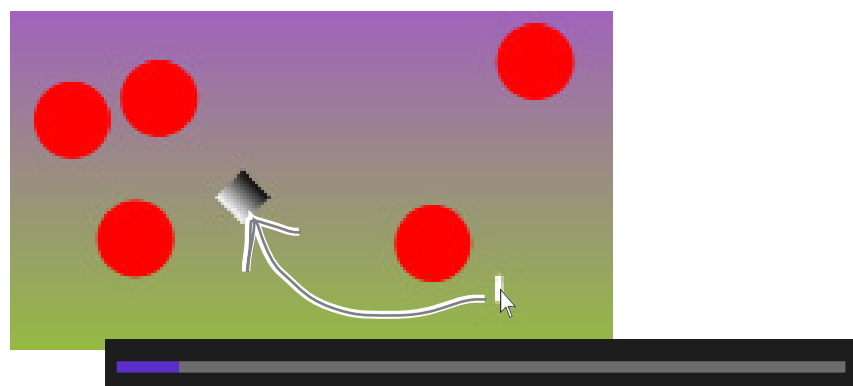


첫 번째 할 일은 시작 버튼을 눌렀을 때 적들이 계속 화면을 가로질러 움직이게 하는 C# 코드를 작성하는 겁니다.

많은 프로그래머들이 코드를 조금씩 점진적으로 작성하며 프로젝트를 진행합니다. 다음 단계로 넘어가기 전에 요구사항을 확인합니다. 즉, 이 프로그램의 나머지 부분도 이런 방식으로 진행합니다. 여러분은 Canvas 컨트롤에 움직이는 적을 AddEnemy()의 메서드에 추가하는 코드를 작성할 겁니다. 페이지에 움직이는 적들을 채우는 것으로 시작해 봅시다. 게임의 나머지 부분을 만드는 데 도움이 될 겁니다.

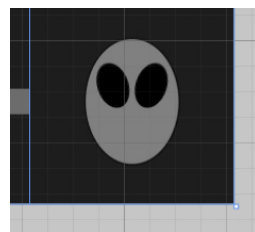
... 그다음, 게임을 해 봅시다.

게임이 동작하려면 프로그램의 바깥 카운트 다운되어야 합니다. 사람이 움직여서 적에 부딪히거나 정해진 시간이 다 되었을 때 게임을 종료해야 합니다.



템플릿을 사용해서 빨간색 원을 만듭니다. 템플릿을 수정해서 나쁜 외계인처럼 만들어 줍니다.

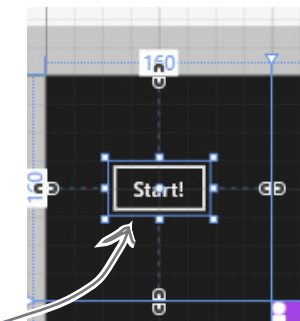
... 마지막으로, 적을 외계인 모습으로 만들어 볼까요?



메서드 추가하기

이제 코딩할 때가 왔습니다. 먼저 메서드(method)를 추가해 봅시다. IDE가 생성해준 코드로 쉽게 시작할 수 있습니다.

페이지를 편집할 때, 컨트롤을 더블-클릭하면 IDE는 코드를 자동으로 생성해 줍니다. 페이지 디자이너에서 Start 버튼을 더블-클릭합니다. 여러분은 플레이어가 Start 버튼을 클릭할 때, 게임을 시작하는 코드를 추가할 것입니다. 다음 코드를 팝업 창에서 볼 수 있습니다.



Button 컨트롤을 더블-클릭 할 때, IDE는 이 메서드를 생성합니다. 플레이어가 "Start!"를 클릭할 때 게임을 시작합니다.

```
private void startButton_Click(object sender, RoutedEventArgs e)
{
}
}
```

Click="startButton_Click"

필요한 메서드를 만들어 봅시다

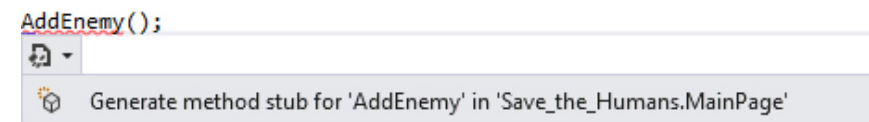
{ } 중괄호 안을 클릭해서 괄호()와 세미콜론;을 포함하는 문장을 입력해 주세요.

```
private void startButton_Click(object sender, RoutedEventArgs e)
{
    AddEnemy();
}
}
```

또한 IDE는 이것을 XAML에 추가합니다. 한번 찾아보세요. 2장에서 이것이 무엇인지 알아봅시다.

그냥 텍스트를 입력했는데 빨간색 물결 모양의 선이 왜 나올까요? 이것은 IDE가 뭔가 잘못되었다는 것을 알려 주는 것입니다. 물결 모양의 선을 클릭하면, 오류를 고치는 데 도움을 주는 작은 파란색 상자가 나타납니다.

작은 파란색 작은 상자 위에서 팝업되는 이 아이콘을 클릭합니다. 그러면 메서드 스텝을 생성하라는 상자를 볼 수 있습니다. 한번 클릭해 보세요.



바보 같은 질문이란 없습니다

Q: 메서드가 뭐예요?

A: 메서드는 코드 블록의 이름입니다. 2장에서 메서드에 대해서 조금 더 이야기해 봅시다.

Q: IDE가 메서드를 생성해 주나요?

A: 네... 현재까지는요. 메서드는 프로그램의 한 부분을 만드는 기본 단위입니다. 여러분은 코딩을 하면서 많은 메서드를 작성하고, 메서드를 사용할 겁니다.

메서드에 코드 채워 넣기

이제 프로그램이 뭔가 해야 할 차례입니다. IDE에서 메서드 스텝을 생성해서 Start 버튼의 메서드를 만들었습니다. 메서드에 코드를 입력해 봅시다.



조심하세요

여기에 있는 C# 코드와 정확하게 일치해야 합니다.

오타로 실수하기 쉽습니다. C# 코드를 입력할 때, 대소문자가 정확하게 일치해야 합니다. 괄호와 콤마, 세미콜론을 잘 확인해 보세요. 하나라도 빠지면 프로그램은 동작하지 않습니다.

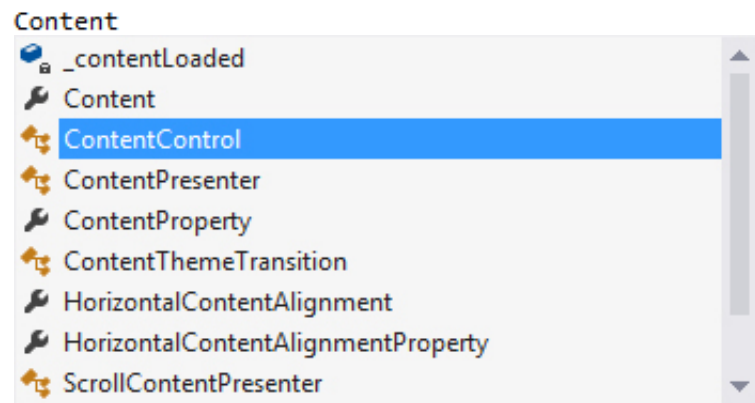
- 1 IDE가 생성한 메서드 스텝의 내용을 지웁니다.

```
private void AddEnemy()
{
    throw new NotImplementedException();
}
```

이것을 선택해서 지우세요. 12장에서 예외에 대한 것을 배울 겁니다.

- 2 코드를 추가합니다. Content를 메서드 몸체에 입력합니다. IDE는 키워드를 제안하는 인텔리센스 창을 띄울 것입니다. 리스트에서 ContentControl을 선택하세요.

```
private void AddEnemy()
{
    Content
}
```



- 3 마지막으로 첫 번째 줄에서 이 코드를 추가하세요. new를 입력할 때 다른 인텔리센스 창이 나타납니다.

```
private void AddEnemy()
{
    ContentControl enemy = new ContentControl();
}
```

이 라인에 새로운 ContentControl 객체를 생성합니다. 3장에서 객체와 new 키워드를, 4장에서 enemy와 같은 참조 변수를 배웁니다.

- 4 AddEnemy() 메서드를 채우기 전에, 파일의 위쪽에 한 줄의 코드를 추가합니다. public sealed partial MainWindow : Window로 시작하는 줄을 찾아서 중괄호{ 이후에 내용을 입력합니다.

```
/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : Window
{
    Random random = new Random();
}
```

이것을 필드(Field)라고 부릅니다. 4장에서 조금 더 자세히 배워 봅시다.

- 5 아래 내용을 AddEnemy 메서드에 입력해 봅시다. 빨간색 물결의 밑줄이 보입니다. 여기에 메서드 스텝을 추가하면 AnimateEnemy()에 있는 밑줄은 사라 집니다.

PlayArea에 물결 모양의 밑줄이 보인다구요? XAML 편집기로 돌아가서 Canvas 컨트롤의 이름이 playArea로 되어 있는지 확인해 보세요.

```
private void AddEnemy()
{
    ContentControl enemy = new ContentControl();
    enemy.Template = Resources["EnemyTemplate"] as ControlTemplate;
    AnimateEnemy(enemy, 0, playArea.ActualWidth - 100, "(Canvas.Left)");
    AnimateEnemy(enemy, random.Next((int)playArea.ActualHeight - 100),
        random.Next((int)playArea.ActualHeight - 100), "(Canvas.Top)");
    playArea.Children.Add(enemy);
}
```

이 라인은 적 컨트롤 객체에 Children (자식)이라고 불리는 컬렉션을 추가합니다. 8장에서 컬렉션에 대해 배워 봅시다.

XAML과 C#의 코드 위치를 바꾸고 싶다면 윈도우 위의 탭을 이용하세요.



- 6 작은 파란색 상자를 이용하여 [?] 버튼을 누르면 AnimateEnemy() 메서드 스텝이 생성됩니다. AddEnemy()에서 했던 것처럼 말이죠. 이번에는 enemy, p1, p2, p3이라고 불리는 4개의 매개변수(parameter)가 추가됩니다. 메서드의 위쪽에 있는 3개의 매개변수를 수정해 봅시다. p1을 from으로, p2를 to, p3을 propertyToAnimate로 수정합니다. 그리고 int 유형을 double로 수정합니다.

```
private void AnimateEnemy(ContentControl enemy, int p1, double p2, string p3)
{
    throw new NotImplementedException();
}

private void AnimateEnemy(ContentControl enemy, double from, double to, string propertyToAnimate)
```

2장에서 메서드와 매개변수에 대해 배워 봅시다.

IDE에서 메서드 이름을 "int" 유형으로 생성할 겁니다. 이것을 "double"로 바꿔 주세요. 4장에서 유형에 대해 배워 봅시다.

다음 페이지에서 프로그램을 실행해 봅시다.

메서드를 끝내고, 프로그램을 실행해 봅시다

여러분의 프로그램은 실행할 준비가 거의 다 되었습니다. 이제 남은 부분은 AnimateEnemy() 메서드를 마무리하는 겁니다. 아직 안 된다고 좌절하지 마세요. 아마 프로그래밍을 할 때 콤마나 괄호 같은 것이 빠져 있을 수도 있어요. 빠져 있는 부분이나 오타를 조심해야 합니다!



아직 물결 모양의 빨간색 선이 보인다고요? IDE는 그 문제가 뭔지 알고 있습니다.

아직 물결 모양의 빨간색 밑줄이 보인다면, 그것은 코드의 일부가 잘못 입력되었다는 것을 의미합니다. 이 책은 수많은 사람의 테스트를 거쳤습니다. 잘못된 부분이 없는지 꼼꼼하게 확인했습니다. 그러므로 프로그램이 돌아가게 하려면 이 책과 똑같이 입력하시면 됩니다.

IDE에서 창들을 다시 설정하고 싶을 때, 창(Window) 메뉴 > 창 레이아웃 다시 설정(Reset Window Layout)을 하면 됩니다.

1 파일의 윗부분에 using문을 추가해 주세요.

파일 맨 위쪽으로 이동합니다. IDE는 using으로 시작하는 몇 개의 라인들을 생성해줍니다. 아래의 리스트 밑에서 한 라인을 추가해 주세요.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Media.Animation;
```

C#에서 .NET 라이브러리의 코드를 사용할 수 있는 이러한 구문들은 2장에서 배워 봅니다.

이 틀을 조금 더 쉽게 작성할 수 있습니다. 인텔리센스 항목을 이용해서 코드를 입력해 줍시다. 마지막 부분에 세미콜론 입력하는 것을 잊지 마세요.

이 using문은 .NET 프레임워크에서 적들이 움직이는 애니메이션 코드를 사용할 수 있게 해줍니다.

4장에서 객체 이니셜라이저에 대해 배워 봅니다.

2 적이 움직이는 애니메이션 코드를 추가해 봅시다.

이전 페이지에서 AnimateEnemy() 메서드를 만들었습니다. 이제 코드를 추가할 차례입니다. 적들이 화면을 가로지르며 움직이게 만들어 봅시다.

```
private void AnimateEnemy(ContentControl enemy, double from, double to, string propertyToAnimate)
{
    Storyboard storyboard = new Storyboard() { AutoReverse = true, RepeatBehavior = RepeatBehavior.Forever };
    DoubleAnimation animation = new DoubleAnimation()
    {
        From = from,
        To = to,
        Duration = new Duration(TimeSpan.FromSeconds(random.Next(4, 6))),
    };
    storyboard.SetTarget(animation, enemy);
    storyboard.SetTargetProperty(animation, new PropertyPath(propertyToAnimate));
    storyboard.Children.Add(animation);
    storyboard.Begin();
}
```

애니메이션에 대해서는 16장에서 배울 거예요.

이 코드는 적이 playArea를 가로지르며 움직이게 해줍니다. 만약 여러분이 숫자 4, 6을 수정한다면, 적들을 느리게나 빠르게 만들 수 있습니다.

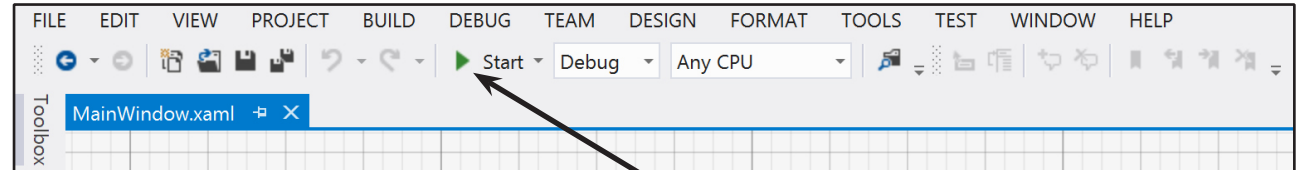
오류 목록 항목 볼 수 없다면 메뉴에서 보기(View) > 오류 목록(Error List)을 선택하세요. 2장에서 오류 항목과 코드 디버깅에 대해 배워 봅니다.

3 코드를 살펴 봅시다.

오류가 있으면 안 됩니다. 오류 목록 창이 비어 있어야 하죠. 오류 목록이 보인다면, 그 오류를 더블-클릭하세요. IDE는 문제를 추적하기 위해 여러분의 커서가 오류가 발생한 부분으로 이동합니다.

4 프로그램을 실행해 봅시다.

IDE에서 ▶ 버튼을 찾아 프로그램을 실행해봅시다.



이 버튼을 누르면 프로그램이 시작합니다.

5 이제 프로그램이 실행됩니다!

프로그램을 실행할 때, 메인 윈도우가 뜹니다. 그리고 "Start!" 버튼을 몇 번 클릭해 보세요. 클릭할 때마다 원 모양의 도형이 캔버스를 가로지르며 움직일 겁니다.

훌륭하네요! 우리가 약속했던 것처럼 오래 걸리진 않았죠? 조금만 더 작업을 해서, 게임을 만들어 봅시다.



만약에 적들이 움직이지 않거나 플레이 영역을 벗어난다면, 코드를 다시 확인해 보세요. 단어나 괄호가 빠져 있을 수도 있습니다.

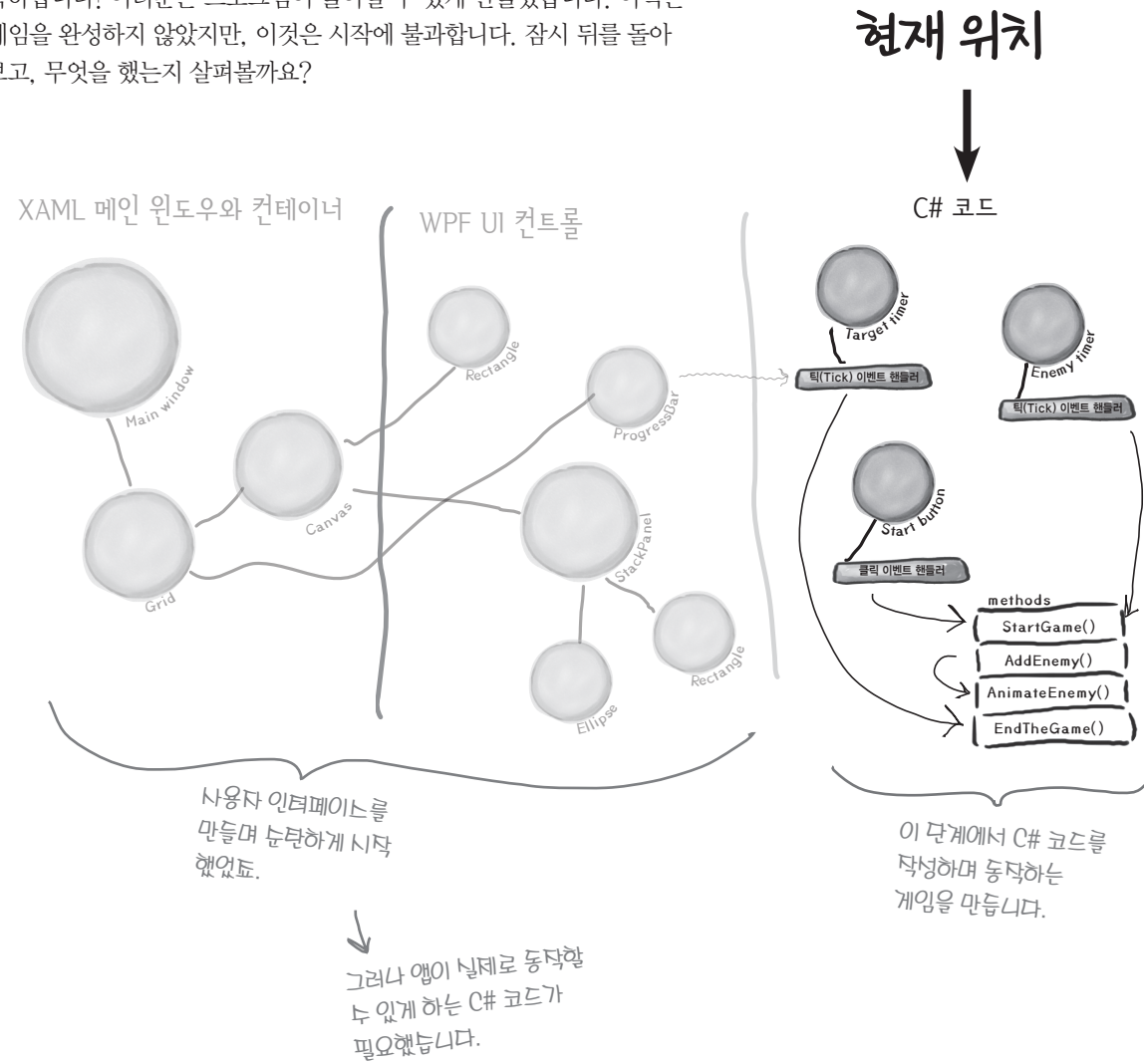
6 프로그램을 멈춰 봅시다.

Alt + Tab을 눌러 IDE로 돌아와서 도구 모음(Toolbar)에 있는 ▶ 버튼의 위치에 ||, ■, ↺ 버튼으로 바뀐 모두 중단(break), 디버깅 중지(stop), 다시 시작(restart)을 확인하세요. 디버깅 중지를 눌러 프로그램을 멈춰 봅시다.

여기는 어디고, 뭘 해야 하죠?

지금까지 한 일

축하합니다! 여러분은 프로그램이 돌아갈 수 있게 만들었습니다. 아직은 게임을 완성하지 않았지만, 이것은 시작에 불과합니다. 잠시 뒤를 돌아 보고, 무엇을 했는지 살펴볼까요?



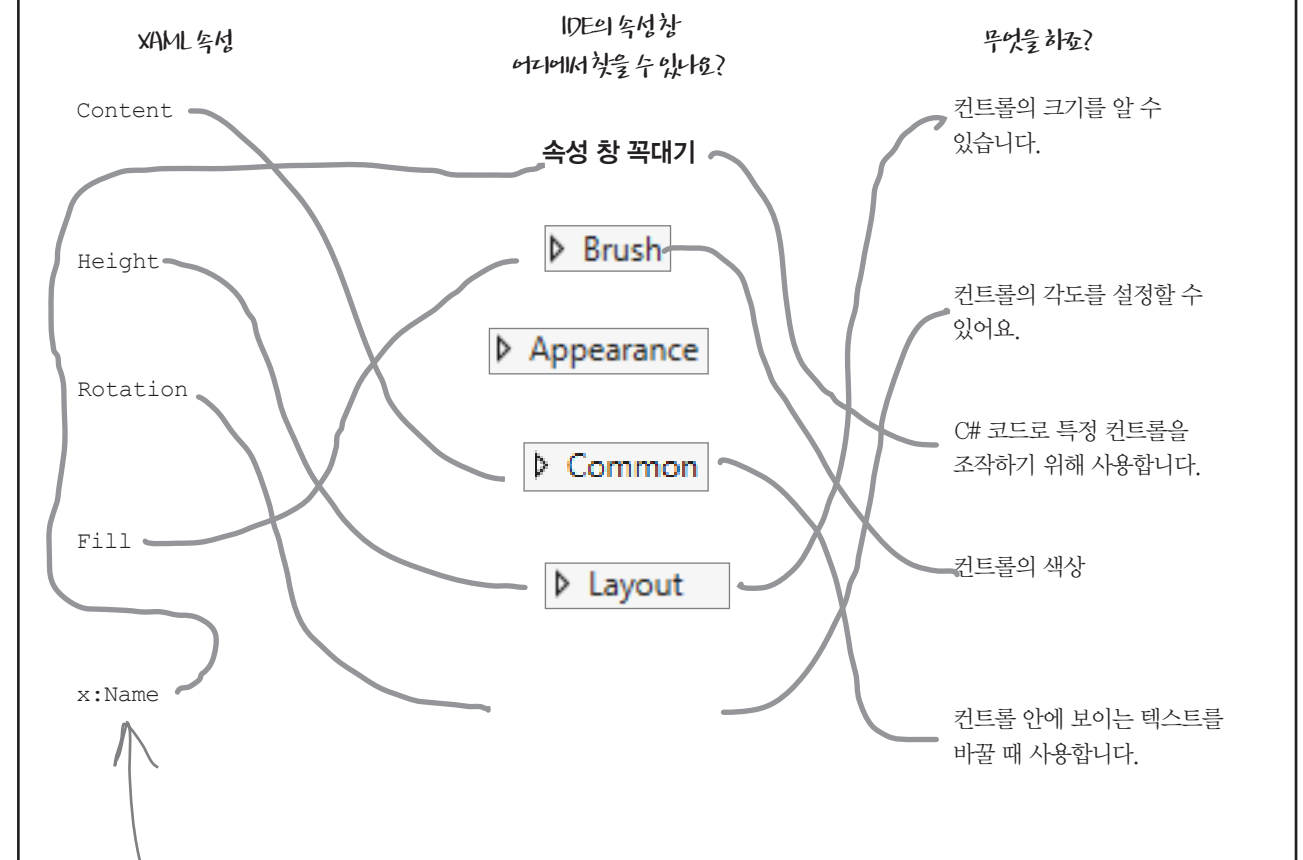
비주얼 스튜디오는 일부 코드를 자동으로 생성해 줍니다. 자동으로 생성한 코드만으로 아무것도 할 수가 없죠. 앱을 만들기 전에 뭘 해야 할지 생각하고, 코드를 작성해야 합니다.

70페이지의 “누가 무엇을 하나요?”의 연습문제 해답이 여기 있습니다. 앞으로, 여러분에게 십자퍼즐과 연습문제에 대한 답을 제공합니다.

나는 누구일까요?

정답

지금까지 사용자 인터페이스를 만들어 보면서 컨트롤이 무슨 역할을 하는지에 대한 감각을 키웠습니다. 컨트롤의 속성이 무엇을 하는지, IDE의 속성 창에 어디에 있는지 찾아서 연결해 보세요.



Canvas 컨트롤의 이름에 “playArea”로 입력한 걸 기억하나요? XAML의 “x:Name” 속성에 있습니다. 이것은 Canvas에서 C# 코드를 쉽게 작성할 수 있도록 도와줍니다.

타이머 추가하기

게임을 동작하는 하나의 중요한 요소를 추가해 봅시다. 이 게임은 적들이 계속 늘어나야 합니다. 그리고 플레이어가 사람을 목표 지점으로 이동시키는 중에 프로그래스 바가 천천히 채워져야 합니다. 타이머(timers)로 두 문제를 해결해 봅시다.

여러분이 작성한 MainWindow. Xaml.cs 파일에는 MainWindow 클래스 코드가 포함되어 있습니다. 클래스에 대해서는 3장에서 배워 봅시다.

1. 윗부분에 C# 코드를 몇 라인 더 추가해 봅시다.

몇 페이지 전에 추가했던 using문을 하나 더 추가해야 합니다.
using System.Windows.Media.Animation; DispatcherTimers를 사용하기 위한 using문입니다.
using System.Windows.Threading;

그리고 나서 여러분이 추가한 Random 라인으로 올라갑니다. 아래 3줄을 추가해 주세요.

```
public partial class MainWindow : Window
{
    Random random = new Random();
    DispatcherTimer enemyTimer = new DispatcherTimer();
    DispatcherTimer targetTimer = new DispatcherTimer();
    bool humanCaptured = false;
```

Random 아래 3줄을 추가하세요. 이것을 필드라 합니다. 4장에서 배워 봅시다.

2. 타이머에 대한 메서드를 추가합니다.

IDE가 생성한 이 코드를 찾아 보세요.

```
public MainWindow()
{
    InitializeComponent();
}
```

커서를 맨 마지막 줄의 세미콜론 바로 뒤에 놓은 뒤, 엔터키를 두 번 칩니다. 그리고 enemyTimer.를 입력합니다(점을 포함합니다). 점을 입력하는 순간, 인텔리센스 창이 나타날 겁니다. 여기서 Tick을 선택하고 아래와 같이 입력하세요. +=를 입력하면 아래의 팝업 상자가 뜹니다.

```
enemyTimer.Tick += enemyTimer_Tick; // (TAB) 키를 눌러 삽입합니다.
```

Tab을 누르면 IDE에서 다른 상자가 나타날 겁니다.

```
enemyTimer.Tick += enemyTimer_Tick;
// enemyTimer_Tick(object sender, EventArgs e)
// (TAB) 키를 눌러 이 클래스에 'enemyTimer_Tick' 처리기를 생성합니다.
```

Tab을 한 번 더 누르면, IDE는 아래의 코드를 생성해 줍니다.

```
public MainWindow()
{
    InitializeComponent();

    enemyTimer.Tick += enemyTimer_Tick;
}

void enemyTimer_Tick(object sender, EventArgs e)
{
    throw new NotImplementedException();
}
```

IDE는 이벤트 핸들러에 대한 메서드를 생성해줍니다. 15장에서 이벤트 핸들러에 대해 배워 봅시다.

타이머의 "tick"은 시간간격마다 메서드를 호출해 줍니다. 첫 번째 타이머는 몇 초마다 적들을 추가합니다. 두 번째 타이머는 시간이 만료되면 게임을 종료합니다.

3. MainWindow() 메서드를 마무리해 봅시다.

두 번째 타이머를 위한 Tick 이벤트 핸들러를 추가해 봅시다. 두 줄의 코드를 추가해야 합니다. 여러분이 작성한 MainWindow() 메서드와 IDE가 생성한 2개의 메서드가 있습니다.

```
public MainWindow()
{
    InitializeComponent();

    enemyTimer.Tick += enemyTimer_Tick;
    enemyTimer.Interval = TimeSpan.FromSeconds(2);

    targetTimer.Tick += targetTimer_Tick;
    targetTimer.Interval = TimeSpan.FromSeconds(.1);
}

void targetTimer_Tick(object sender, EventArgs e)
{
    throw new NotImplementedException();
}

void enemyTimer_Tick(object sender, EventArgs e)
{
    throw new NotImplementedException();
}
```

게임을 다 만들었을 때, 여기에 있는 숫자를 바꿔봅시다. 어떻게 게임이 바뀔까요?

여러분이 Tab을 눌러 이벤트 핸들러를 추가할 때마다 IDE는 여기에 코드를 생성합니다. 여러분은 타이머의 Tick마다 실행되는 코드로 수정해야 합니다.

4. EndTheGame() 메서드를 추가합니다.

targetTimer_Tick() 메서드로 가서, IDE가 생성해준 코드를 지우고, 아래의 코드를 추가합니다. EndTheGame()을 입력하고, 전에 했던 것처럼 메서드 스텝을 생성해 주세요.

```
void targetTimer_Tick(object sender, object e)
{
    progressBar.Value += 1;
    if (progressBar.Value >= progressBar.Maximum)
        EndTheGame();
}
```

progressBar를 입력할 때 계속해서 대문자 P로 바뀌나요? IDE가 코드에 노문자 p의 progressBar가 없어서 이와 비슷한 컨트롤 유형을 찾아 두기 때문입니다.

왜 progressBar에 오류가 날까요? 여러분이 컨트롤을 사용할 때, 이름이 없거나 오타가 발생할 경우 어떻게 되는지 보여 주기 위해서 일부러 이렇게 코드를 입력했습니다. XAML 코드 창(IDE의 다른 탭에 있는)으로 가서, 아래 줄에 추가한 ProgressBar 컨트롤을 찾아보세요. 그리고 이름을 progressBar로 수정해 주세요.

다음, 코드 창으로 돌아가서 EndTheGame() 메서드 스텝을 생성합니다. 조금 전에 했던 AddEnemy()를 생성한 것처럼 말이죠. 여기에 메서드 코드가 있습니다.

```
private void EndTheGame()
{
    if (!playArea.Children.Contains(gameOverText))
    {
        enemyTimer.Stop();
        targetTimer.Stop();
        humanCaptured = false;
        startButton.Visibility = Visibility.Visible;
        playArea.Children.Add(gameOverText);
    }
}
```

gameOverText가 오류로 뜬다면, TextBlock의 이름 속성이 gameOverText로 되어 있지 않다는 것을 의미합니다. 지금 돌아가서 바로 수정하세요.



브레인 파워

지금 바로 시작 버튼을 눌러 움직이는 적을 플레이 영역에 추가해 봅시다. 시작 버튼을 눌러 적을 추가하는 대신에 적을 자동으로 생성하려면 무엇이 필요할까요?

XAML 코드가 있는 디자이너 탭이 닫혀 있다면, 눌러서 탐색기 탭에 있는 MainWindow.xaml을 더블-클릭 하세요.

이 메서드는 타이머를 멈추고, 시작 버튼을 다시 보이게 합니다. 그리고 게임을 종료하며, "GAME OVER" 텍스트를 playArea에 보여 줍니다.

시작 버튼 동작하게 하기

시작 버튼을 누르면 캔버스에 원이 나오게 만든 걸 기억하나요? 이제 진짜 게임이 시작되도록 고쳐 봅시다.

1 시작 버튼이 동작하도록 만들어 봅시다.

이미 추가한 시작 버튼을 누르면 적이 생성되는 코드를 찾습니다. 다음과 같이 수정하세요.

```
private void startButton_Click(object sender, RoutedEventArgs e)
{
    StartGame();
}
```

이 라인은 시작 버튼으로 적을 playArea 캔버스에 생성하는 대신 게임을 시작하게 합니다.

2 StartGame() 메서드를 추가합니다.

메서드 스텝을 생성해서 StartGame() 메서드를 만드세요. IDE가 추가한 메서드 스텝에 다음과 같이 코드를 입력하세요.

```
private void StartGame()
{
    human.IsHitTestVisible = true;
    human.Captured = false;
    progressBar.Value = 0;
    startButton.Visibility = Visibility.Collapsed;
    playArea.Children.Clear();
    playArea.Children.Add(target);
    playArea.Children.Add(human);
    enemyTimer.Start();
    targetTimer.Start();
}
```

15장에서 IsHitTestVisible에 대해 배워 봅시다.

포팅(target)의 Rectangle과 사람(human)의 StackPanel 컨트롤의 이름 설정을 했습니다. 컨트롤의 이름을 맞게 설정했는지 확인해 보세요.


3 EnemyTimer에 적을 추가합니다.

IDE가 생성한 enemyTimer_Tick() 메서드를 찾아서 다음과 같이 수정하세요.

```
void enemyTimer_Tick(object sender, object e)
{
    AddEnemy();
}
```

이해할 수 없는 오류들이 오류 목록 창에 보입니까? 하나의 점이나 세미콜론이 잘못되었을 경우, 하나의 오류가 두 개, 세 개, 네 개 혹은 그 이상의 오류를 발생시킵니다. 모든 오타를 찾는 데 시간을 낭비하지 마세요! 아래의 웹 페이지로 가면 이 프로그램의 코드를 쉽게 찾아서 복사/붙여 넣기를 할 수 있습니다.

<http://www.hanbit.co.kr/exam/2165>



코드 구울 준비

이 책에서는 여러분이 작성할 많은 코드를 제공합니다.

이 책의 마지막에서 이 코드가 무엇을 하게 되는지 알게 될 겁니다. 지금처럼 아무 생각 없이 그냥 코드를 입력할 수 있습니다.

현재 여러분이 해야 할 일은 코드를 정확하게 입력하고, 지시에 정확히 따르는 겁니다. 이렇게 코딩을 하면서 IDE에 익숙해질 수 있습니다.

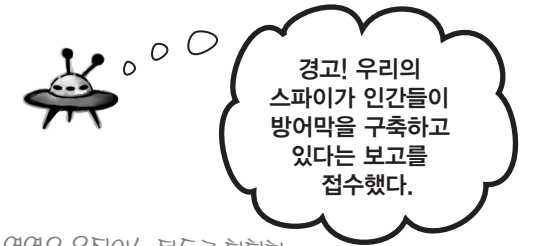
만약 진행이 잘 되지 않는다면, MainWindow.xaml과 MainWindow.Xaml.cs 파일을 내려받으세요. 혹은 각각의 메서드에 XAML과 C#의 코드를 복사/붙여넣기하세요. 아래의 URL에 있습니다.

<http://www.hanbit.co.kr/exam/2165>

한 가지 더! 부록에 있는 프로젝트를 내려받을 때, WPF 폴더에서 받았는지 꼭 확인해야 합니다. 윈도우 스토어 앱은 WPF 프로젝트에서 실행되지 않습니다.

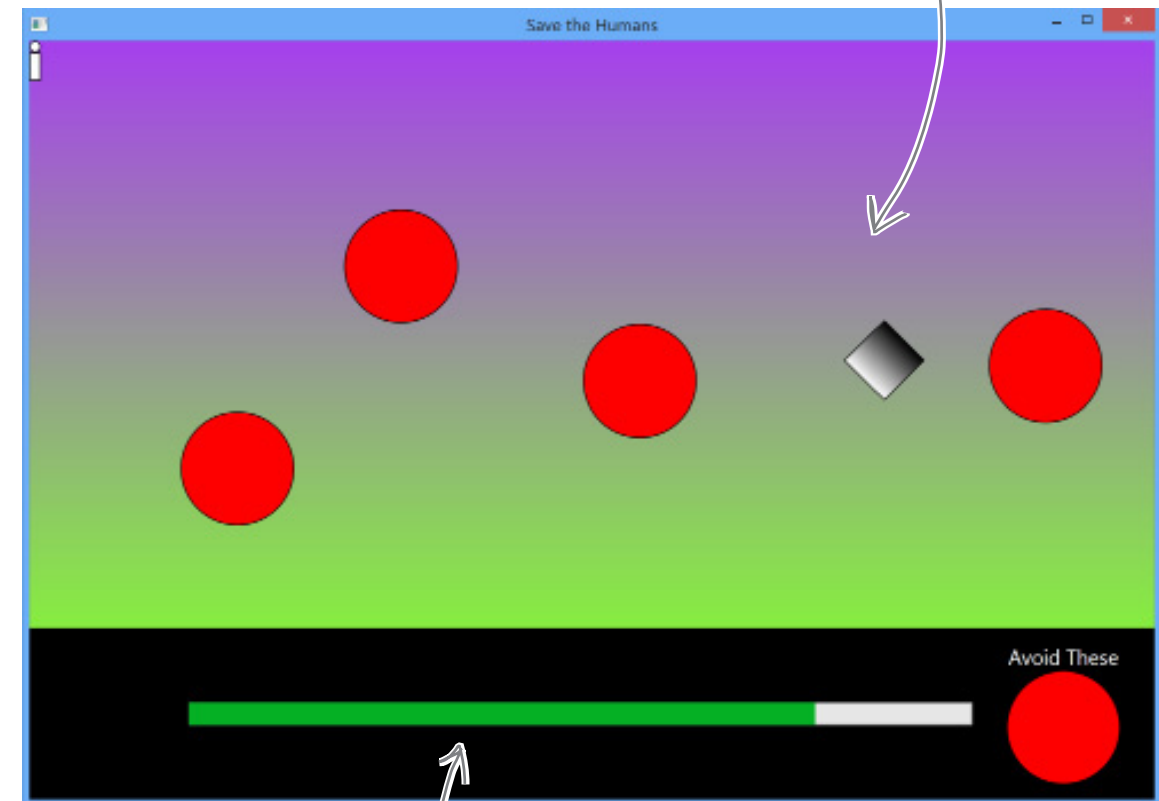
진행 상황을 볼까요?

게임이 거의 완성되어 갑니다. 게임을 실행해서 어떻게 동작하는지 살펴봅시다.




“Start!” 버튼을 누를 때, 버튼이 사라지고, 적군들이 나타나며, 프로그래스 바가 채워지기 시작합니다.

플레이 영역은 움직이는 적들로 현현히 채워지기 시작합니다.



프로그래스 바가 다 찼을 때, 게임이 끝나고 Game Over 텍스트가 표시됩니다.

타기 타이머가 현현히 채워지면서, 적군들은 3초마다 생성됩니다. 만약 타이머가 동작하지 않는다면, MainWindow() 메서드에서 추가한 모든 코드를 확인해 보세요.



브레인 파워


게임을 완성하기 위해서 어떤 일들이 남았을까요?

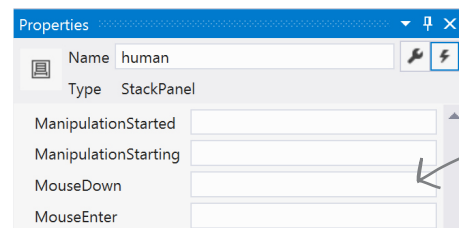
나머지 해야 할 일들은 다음 페이지에 있습니다. →

플레이어가 컨트롤을 조정하는 코드를 작성해 봅시다

플레이어가 사람을 포탈로 끌어야 합니다. 사람이 포탈에 닿을 때, 적절한 반응을 해야 합니다. 이러한 코드를 작성해 봅시다.

코드를 작성하기 전에 앱을 멈추고, IDE로 전환하세요.

- ① XAML 디자이너로 가서, 문서개요 창의 human을 선택합니다(기억하시죠? Circle과 Rectangle으로 이루어진 StackPanel입니다). 그리고 속성 창에서  버튼을 누르면 이벤트 핸들러가 보입니다. MouseDown 행을 찾아 빈 상자에 더블-클릭하세요.
- 4장에서 녹성 창에 있는 이벤트 핸들러를 배워 봅시다.



이 상자를 더블-클릭하세요.

이제 XAML 창에서 IDE가 StackPanel을 위해 무엇을 추가했는지 확인해 보세요.

```
<StackPanel x:Name="human" Orientation="Vertical" MouseDown="human_MouseDown">
```

메서드 스텝이 추가됐습니다. XAML 창에서 human_MouseDown을 오른쪽 클릭합니다. “정의로 이동”을 선택해 C# 코드로 바로 넘어갑니다.

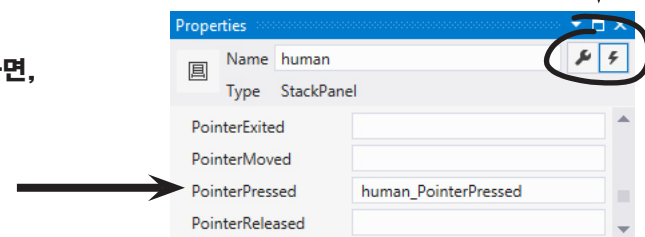
```
private void human_MouseDown(object sender, MouseButtonEventArgs e)
{
}
```

- ② 그리고 코드를 입력하세요.

```
private void human_PointerPressed(object sender, PointerRoutedEventArgs e)
{
    if (enemyTimer.IsEnabled)
    {
        humanCaptured = true;
        human.IsHitTestVisible = false;
    }
}
```

속성 창에서 속성과 이벤트 핸들러가 있는 이 버튼을 이용하면 됩니다.

디자이너에서 StackPanel을 다시 클릭한다면, 속성 창에서 새로운 이벤트 핸들러 메서드의 이름이 보입니다. 더 많은 이벤트 핸들러를 같은 방법으로 추가할 수 있습니다.

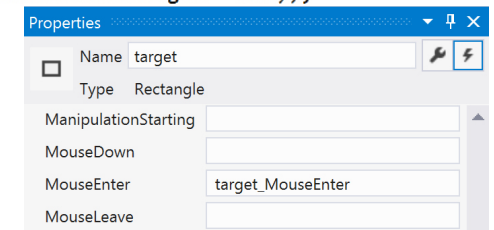


- ③ 이벤트 핸들러를 빠르게 추가했는지 확인하세요! 이전엔 사람 컨트롤에 Windows Presentation Foundation MouseDown 이벤트 핸들러를 추가했습니다. 여기에서는 타깃에 MouseEnter 이벤트 핸들러를 추가해 봅시다.

문서 개요 창의 이름이 target으로 된 Rectangle을 선택합니다. 속성 창의 “선택한 요소의 이벤트 처리기(event handlers view)”에서 MouseEnter 이벤트 핸들러를 추가합니다. 여기에 메서드 코드가 있습니다.

```
private void target_MouseEnter(object sender, MouseEventArgs e)
{
    if (targetTimer.IsEnabled && humanCaptured)
    {
        progressBar.Value = 0;
        Canvas.SetLeft(target, random.Next(100, (int)playArea.ActualWidth - 100));
        Canvas.SetTop(target, random.Next(100, (int)playArea.ActualHeight - 100));
        Canvas.SetLeft(human, random.Next(100, (int)playArea.ActualWidth - 100));
        Canvas.SetTop(human, random.Next(100, (int)playArea.ActualHeight - 100));
        humanCaptured = false;
        human.IsHitTestVisible = true;
    }
}
```

속성 창에서 이벤트 핸들러가 보인다면, 빈 이벤트 핸들러 상자에 더블-클릭하세요. IDE는 메서드 스텝을 생성해 줍니다.



이벤트 핸들러와 녹성을 보여 주는 녹성 창에서 컨트롤 이름을 바꿀 수 있습니다.

- ④ 이번에는 playArea Canvas 컨트롤에 2개의 이벤트 핸들러를 추가합니다. 문서 개요에서 [Grid]를 선택한 후, 이름을 grid로 설정합니다. 그리고 Canvas의 MouseMove와 MouseLeave 이벤트 핸들러를 처리하는 메서드를 추가합니다.

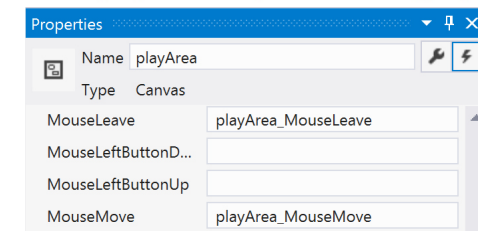
```
private void playArea_MouseMove(object sender, MouseEventArgs e)
{
    if (humanCaptured)
    {
        Point pointerPosition = e.GetPosition(null);
        Point relativePosition = grid.TransformToVisual(playArea).Transform(pointerPosition);
        if ((Math.Abs(relativePosition.X - Canvas.GetLeft(human)) > human.ActualWidth * 3)
            || (Math.Abs(relativePosition.Y - Canvas.GetTop(human)) > human.ActualHeight * 3))
        {
            humanCaptured = false;
            human.IsHitTestVisible = true;
        }
        else
        {
            Canvas.SetLeft(human, relativePosition.X - human.ActualWidth / 2);
            Canvas.SetTop(human, relativePosition.Y - human.ActualHeight / 2);
        }
    }
}
```

괄호가 정말 많네요! 실수하지 않게 조심하세요.

두 개의 수직 바는 논리 연산을 의미합니다. 2장에서 배워 봅시다.

숫자 3보다 작거나 크게 해서, 감도를 높이거나 낮출 수 있습니다.

이벤트 핸들러에 코드를 똑바로 입력했는지 확인하세요. 두 내용이 바뀌면 안 됩니다.



사람과 적이 부딪히면 게임이 끝납니다

플레이어가 사람을 적으로 끌어 놓으면, 게임이 종료하는 코드를 추가해 봅시다. AddEnemy() 메서드의 마지막 부분에서 한 줄의 코드를 추가합니다. 인텔리센스 창을 이용해서 enemy.MouseEnter를 입력합니다.

```
private void AddEnemy()
{
    ContentControl enemy = new ContentControl();
    enemy.Template = Resources["EnemyTemplate"] as ControlTemplate;
    AnimateEnemy(enemy, 0, playArea.ActualWidth - 100, "(Canvas.Left)");
    AnimateEnemy(enemy, random.Next((int)playArea.ActualHeight - 100),
        random.Next((int)playArea.ActualHeight - 100), "(Canvas.Top)");
    playArea.Children.Add(enemy);
}
```

enemy를 입력하고 점(.)을 누르면 인텔리센스 창이 뜹니다. 그리고 "Enter"를 입력해 보세요. "Enter"를 포함하는 목록으로 건너 뜁니다.

← AddEnemy() 메서드의 마지막 줄에 커서를 두고, 엔터키를 친 후 코드를 입력하세요.

← MouseEventArgs UIElement.MouseEnter 마우스 포인터가 요소의 경계에 들어올 때 발생합니다.

목록에서 MouseEnter를 선택하세요(잘못 선택해도 괜찮아요. 점(.)까지 다시 지우면 됩니다. 다시 점을 입력해서 인텔리센스 창을 띄우세요).

다음, 전에 했던 것처럼 이벤트 핸들러를 추가합니다. +=를 입력하고 Tab 키를 누르세요.

```
enemy.MouseEnter += enemy_MouseEnter;
// enemy_MouseEnter; (TAB) 키를 눌러 삽입합니다.)
```

다시 Tab 키를 눌러 이벤트 핸들러를 위한 스텝을 생성합니다.

```
enemy.MouseEnter += enemy_MouseEnter;
// (TAB) 키를 눌러 이 클래스에 enemy_MouseEnter 처리기를 생성합니다.
```

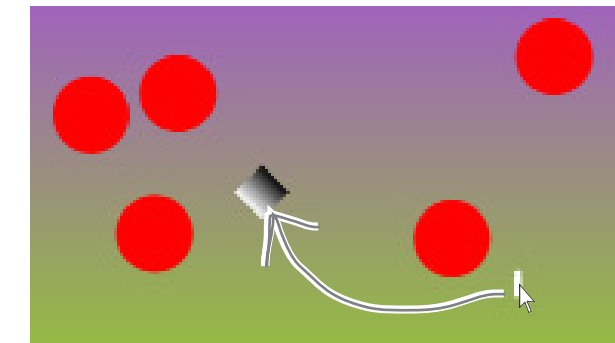
이제 새로운 메서드가 생성되었습니다. 다음 코드를 입력하세요.

```
void enemy_MouseEnter(object sender, MouseEventArgs e)
{
    if (humanCaptured)
        EndTheGame();
}
```

이제 게임을 할 수 있습니다

게임을 실행해 보세요. 이제 거의 끝났습니다. 시작 버튼을 누를 때, 플레이 영역에서 동그란 적이 나타 납니다. 그리고 사람과 포탈이 놓여 있습니다. 프로그레스 바가 가득 차기 전에 사람을 포탈로 옮겨야 합 니다. 처음엔 쉽지만, 시간이 지날수록 적이 늘어나서 게임이 어려워집니다.

사람을 안전한 곳으로 대피시켜 주세요!



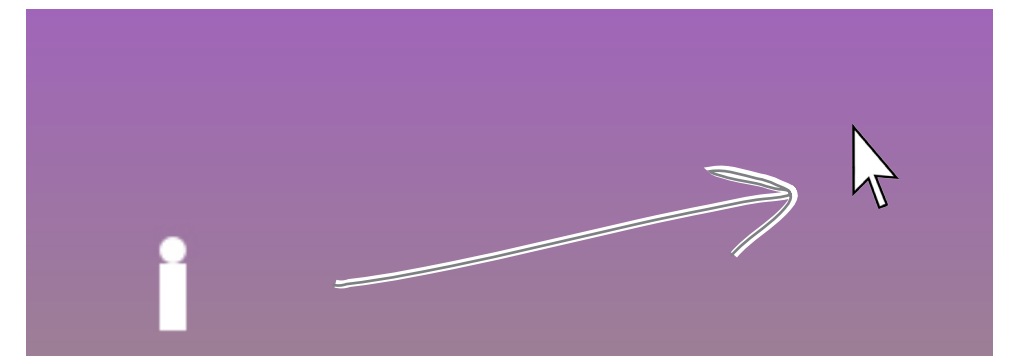
← 외계인들은 이리저리 움직이며 사람을 눈찰합니다. 사람과 적이 부딪히는 순간 게임은 끝납니다. 외계인을 피해 인간을 잘 움직인다면, 적으로부터 일시적으로 안전합니다.

← human에 적용된 IsHitTestVisible 특성을 살펴보세요. IsHitTestVisible 특성이 true일 때, 사람은 MouseEnter 이벤트를 차단합니다. 사람(human)의 StackPanel 컨트롤이 포탈에 도착했을 때, 사람과 포탈이 새로운 위치로 이동하기 때문이죠.

시간이 가기 전에 사람을 피신시켜 주세요.



너무 빨리 끌면 사람을 잃을 수 있어요!



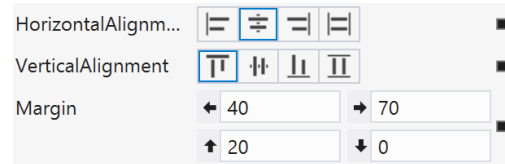
적을 외계인처럼...

빨간색 원은 뭔가 이상합니다. 템플릿을 사용해서 빨간색 원을 조금 더 멋지게 수정해 봅시다.

- 1 문서 개요에서 ContentControl 오른쪽 클릭 > 템플릿 편집 > 현재 항목 편집을 선택합니다. XAML 창에서 템플릿을 볼 수 있습니다. Ellipse 태그에서 width를 75로 Fill을 Gray로 설정합니다. 그리고 Stroke="Black"을 추가해서 바깥 테두리 선을 만듭니다. 그리고 수평과 수직으로 다시 정렬합니다. 코드는 다음과 같습니다(작업을 하면서 실수로 추가된 속성들을 지워도 됩니다).

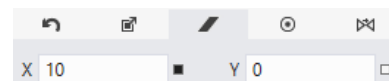
```
<Ellipse Fill="Gray" Stroke="Black" Height="100" Width="75"/>
```

- 2 도구 상자에서 이미 만든 타원 위에 또 다른 Ellipse를 추가합니다. Fill="black", width=25, height=35로 수정합니다. 그리고 정렬과 여백 설정을 다음과 같이 합니다.

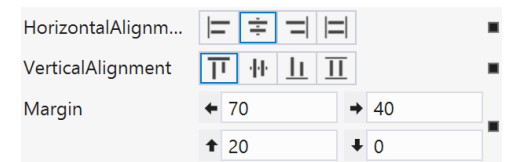


마우스나 방향키를 이용해 Ellipse를 눈으로 이동시켜 눈알을 만들 수도 있습니다. 기존에 있는 Ellipse를 복사/붙여넣기해서 눈 위에 놓고, 조금만 수정하면 됩니다.

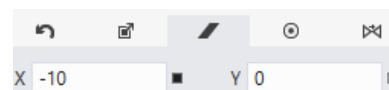
- 3 속성 창에서 변형 섹션의 / (기울이기)를 선택하고 다음과 같이 합니다.



- 4 이미 만든 타원 위에 Ellipse를 하나 더 추가합니다. Fill="black", width=25, height=35로 수정합니다. 그리고 정렬과 여백 설정을 다음과 같이 합니다.




그리고 기울이기를 추가합니다.




이제 여러분의 적은 인간을 잡아 먹는 외계인처럼 생겼네요.





속성 대신 이벤트가 보인다고요?

속성 창에 있는 토글 버튼을 이용해서 여러분이 선택한 컨트롤의 속성과 이벤트를 볼 수 있습니다. 스페너와 번개 모양의 아이콘을 클릭해 보세요.



여기에 새롭게 만든 적 템플릿의 XAML 코드가 있습니다.

```
<ControlTemplate x:Key="EnemyTemplate" TargetType="{x:Type ContentControl}">
  <Grid>
    <Ellipse Fill="Gray" Stroke="Black" Height="100" Width="75"/>
    <Ellipse Fill="Black" Stroke="Black" Height="35" Width="25"
      VerticalAlignment="Top" HorizontalAlignment="Center"
      Margin="40,20,70,0" RenderTransformOrigin="0.5,0.5">
      <Ellipse.RenderTransform>
        <TransformGroup>
          <ScaleTransform/>
          <SkewTransform AngleX="10"/>
          <RotateTransform/>
          <TranslateTransform/>
        </TransformGroup>
      </Ellipse.RenderTransform>
    <Ellipse Fill="Black" Stroke="Black" Height="35" Width="25"
      VerticalAlignment="Top" HorizontalAlignment="Center"
      Margin="70,20,40,0" RenderTransformOrigin="0.5,0.5">
      <Ellipse.RenderTransform>
        <TransformGroup>
          <ScaleTransform/>
          <SkewTransform AngleX="-10"/>
          <RotateTransform/>
          <TranslateTransform/>
        </TransformGroup>
      </Ellipse.RenderTransform>
    </Ellipse>
  </Grid>
</ControlTemplate>
```

XAML 코드를 참고해서 사람과 포탄, 플레이 영역과 외계인을 창의적으로 바꿔보세요.

그리고 복습하는 것을 잊지 마세요. 여러분은 임무를 훌륭하게 완수했습니다. 굿 님!

한 가지 더 해야 할 것은...
그냥 여러분이 만든 게임을 즐기세요!

Chapter 2

2장의 첫 부분에 나오는 몇몇 프로젝트들은 XAML과 윈도우 스토어 앱을 사용합니다. 이것을 WPF 프로젝트로 바꿔봅시다.

WPF 프로젝트와 함께 코딩을 시작해 봅시다.

두 번째 장은 C# 코드를 작성하는 것부터 시작합니다. 그리고 몇몇의 예제가 윈도우 스토어 앱으로 만들어졌습니다.

우리는 아래의 방법을 추천합니다.

- ★ 책 112쪽까지 2장의 주요 부분을 읽어 주세요.
- ★ 부록에서는 책 113쪽의 내용을 대체합니다. 그리고 책 114, 115, 116쪽을 읽어 주세요.
- ★ 프로그램을 만드는 117, 118쪽의 내용을 부록에서 대체합니다. 그리고 프로젝트의 나머지 부분은 이 책을 참고하면 됩니다.
- ★ 그리고 책 126쪽까지 읽어 주세요.
- ★ 다음 127쪽의 연습문제와 129쪽의 해답이 부록에 있습니다.

연습문제를 다 풀고 나서, 9장과 두 번째 실습 전까지는 윈도우 8 혹은 윈도우 스토어 앱 프로젝트가 아닙니다. 10장이 되기 전까지는 책을 읽어 주세요.



변수 값의 추이를 지켜보려면 디버거를 사용하세요

디버거는 여러분이 만든 프로그램이 동작하는 방식을 이해하는 최상의 도구입니다. 이전 페이지에 있는 코드가 실제로 어떻게 움직이는지 확인하는 데 디버거를 사용할 수 있죠.

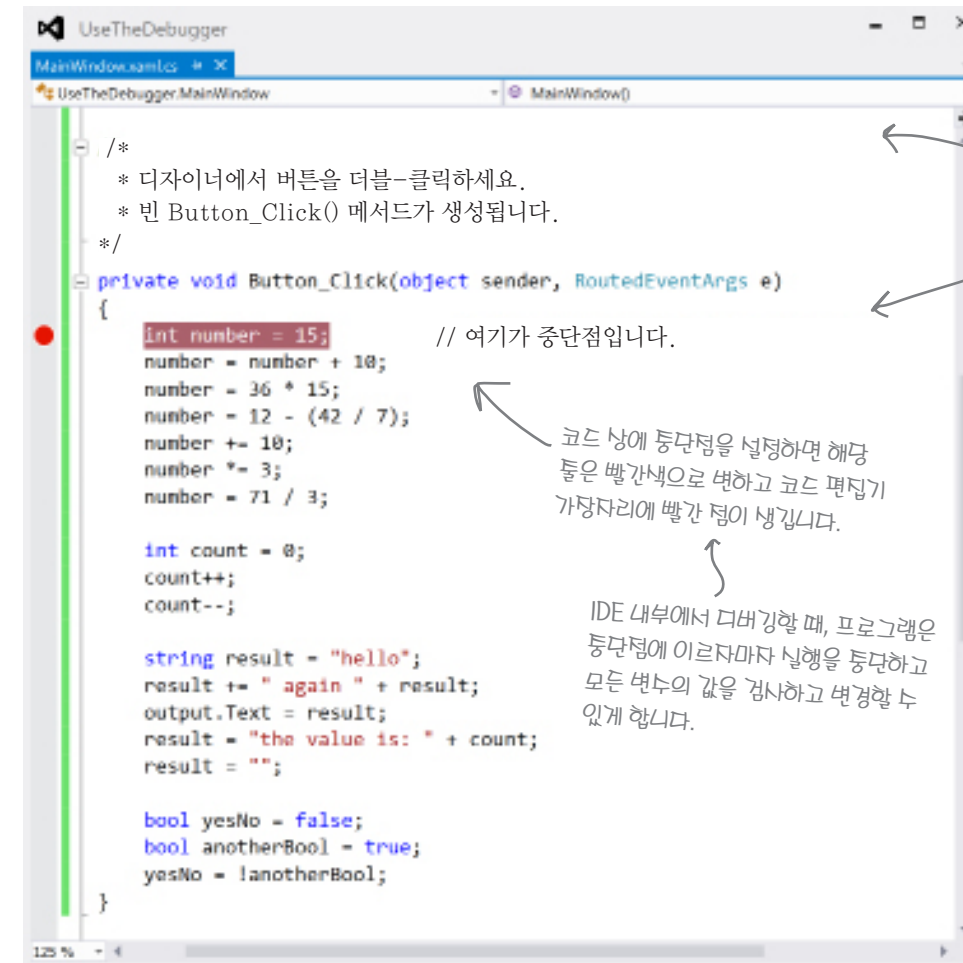


1 새 WPF 응용 프로그램 프로젝트를 생성합니다.

페이지에다 TextBlock을 하나 끌어다 놓고, 이름을 output으로 입력합니다. 그리고 Button을 추가한 뒤 더블-클릭을 해서, Button_Click() 메서드를 추가하세요. IDE는 자동으로 코드 편집 창에 있는 메서드로 갑니다. 이전 페이지에 있는 모든 코드를 메서드 안에 작성하세요.

2 코드 첫 번째 줄에 중단점(breakpoint)을 추가하세요.

코드 첫 번째 줄(int number = 15;)에서 오른쪽 버튼을 클릭하고 중단점 메뉴에서 중단점 삽입(Insert Breakpoint)을 선택하세요(디버그(Debug) 메뉴 > 중단점 설정/해제(Toggle Breakpoint)를 선택하거나 F9키를 입력해도 됩니다).



두석(두 개 이상의 줄레시 다음이나 /*와 */ 사이에 오는 내용)은 IDE에서 녹색으로 표시됩니다. 이 마크 사이에 무엇을 입력해야 할지는 고민할 필요가 전혀 없어요. 컴파일러는 항상 두석을 무시합니다.

코드 상에 중단점을 설정하면 해당 줄은 빨간색으로 변하고 코드 편집기 가장자리에 빨간 점이 생깁니다.

IDE 내부에서 디버깅할 때, 프로그램은 중단점에 이르자마자 실행을 중단하고 모든 변수의 값을 검사하고 변경할 수 있게 합니다.

새 WPF 프로젝트를 생성하는 것은 IDE에서 빈 페이지 하나와 새 프로젝트를 만들라고 얘기해 주는 것입니다. 여러분은 프로젝트 이름을 UseTheDebugger와 같은 이름으로 지을 수도 있습니다(이 페이지에서의 제목과 맞추기 위해서). 많은 프로그램이 이 책으로 만든 후, 필요할 때 다시 사용할 수 있습니다.

책 114쪽으로 가서 계속 읽어 주세요!

부록에서 두 페이지 모드를 유지하기 위해서 일부러 페이지를 비웠습니다. 연습문제와 연습문제 정답이 서로 마주하는 면에 나오면 안 되기 때문이죠. 그리고 두 페이지 모드의 짝수 쪽에는 오른쪽 위에, 홀수 쪽에는 왼쪽 위에 작은 캡션이 있습니다.

이 프로젝트에 맞는 이름을 지으세요.
여러분이 나중에 다시 볼 수도 있으니까요.

기초부터 만들기

모든 프로그램에서 실제로 일을 하는 것은 선언문입니다. 지금까지 어떻게 선언문들이 페이지와 함께 동작 하는지 살펴봤습니다. 이번 실습에서는 코드의 내용을 모두 이해할 수 있습니다. **Windows > WPF 응용 프로그램 생성해 봅시다.** 메인 윈도우를 열어서, 3개의 행과 2개의 열을 그리드에 추가해 봅시다. 그리고 4개의 Button 컨트롤과 TextBlock 컨트롤을 셀에 추가해 봅시다.

Windows Presentation Foundation

운동화 그림은 코드 실행을 의미합니다.



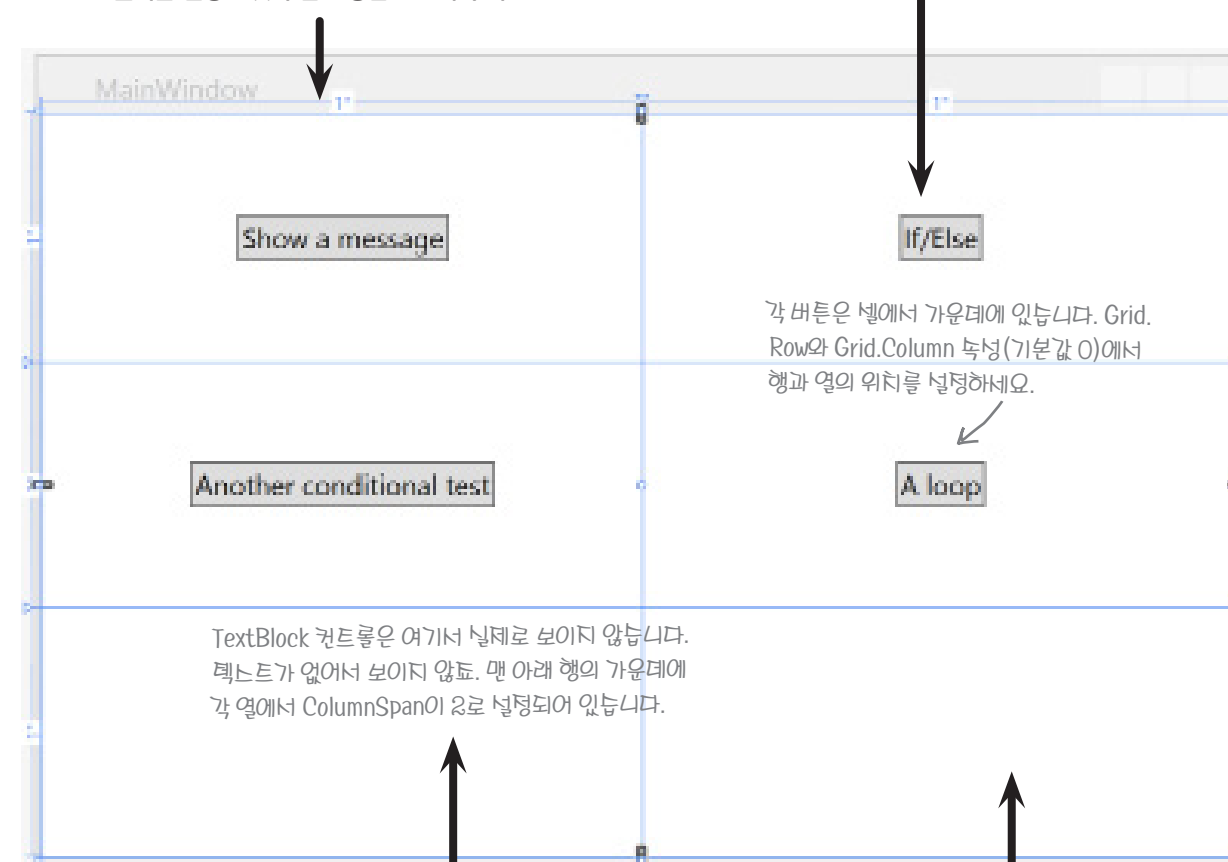
연습문제

윈도우를 만들어

봅시다

페이지에는 3개의 행과 2개의 열이 있습니다. 각각의 행의 높이는 1*로 합니다. 아무런 속성 없이 <RowDefinition/>으로만 되어 있습니다. 열에서도 높이를 설정한 것과 같은 방법으로 해 주세요.

윈도우의 각 행에는 4개의 Button 컨트롤이 있습니다. Content 속성에서 텍스트를 Show a message, if/else, Another conditional test, A loop로 입력하세요.



각 버튼은 셀에서 가운데에 있습니다. Grid. Row와 Grid.Column 속성(기본값 0)에서 행과 열의 위치를 설정하세요.

TextBlock 컨트롤은 여기서 실제로 보이지 않습니다. 텍스트가 없어서 보이지 않죠. 맨 아래 행의 가운데에 각 열에서 ColumnSpan이 2로 설정되어 있습니다.

아랫부분의 Text 컨트롤의 이름을 myLabel로 합니다.

x:Name 속성을 이용해서 버튼의 이름을 button1, button2, button3, button4로 수정하세요. 이름을 바꾼 후 각 버튼을 더블-클릭해서 이벤트 핸들러 메서드를 추가하세요.



연습문제 정답

여기에 연습문제 정답이 있습니다. 여러분이 한 것과 비슷하나요? 틀 바꿈이나 녹성이 조금 다르다고요? 그래도 괜찮아요!

IDE를 사용하지 않고 XAML을 손으로 코드를 작성하는 프로그래머도 있습니다. 여러분도 IDE를 사용하지 않고 코드를 작성할 수 있나요?

여기에 네 응용 프로그램에서 IDE가 생성한 <Window> 와 <Grid> 태그가 있습니다.

여기에 행과 열의 정의 (definition)가 있습니다. 3개의 행과 2개의 열이 있네요.

디자이너에서 각 버튼을 더블-클릭했을 때, IDE는 버튼의 이름에 _Click이 따라오는 메시지를 생성합니다.

이 버튼은 두 번째 열과 두 번째 행에 있습니다. 그리고 이들의 녹성은 1로 설정되어 있습니다.

버튼에서 HorizontalAlignment 나 VerticalAlignment 속성을 지워 보세요. 이 속성이 설정되어 있지 않으면, 버튼이 수평 혹은 수직으로 셀 전체를 채웁니다.



브레인 파워

왜 왼쪽 열과 위쪽 행은 숫자 1이 아니라 0일까요? 그리고 왼쪽 위의 셀("Show a message")에서는 Grid.Row와 Grid.Column 속성이 설정되어 있지 않을까요?

이 책에는 이런 연습문제들이 많이 있습니다. 답은 몇 페이지 뒤에 있어요. 혹시 중간에 막히게 되거든 너무 부담 갖지 말고 막히는 부분만 참고해도 괜찮아요.

이 책을 보면서 수많은 애플리케이션을 만들게 될 텐데, 각각의 애플리케이션의 이름은 서로 다르게 붙여야 합니다. 애플리케이션 이름은 "PracticeUsingIfElse" 같은 식으로 당 번호의 같은 폴더에 관리하면 편리합니다.



연습문제

if/else문 연습을 해 봅시다. 아래의 프로그램을 만들어보세요.

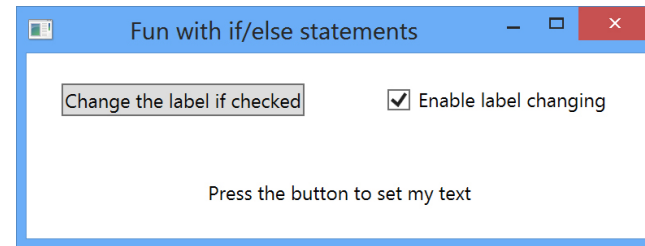
이 창을 만드세요.

그리드에 두 개의 행과 두 개의 열이 있습니다. 창의 높이는 150픽셀이고 너비는 450픽셀입니다. 창 제목을 Fun with if/else statements로 해 주세요.

IDE에서 두 개의 행을 만들고 하나의 행 높이에 1*을 설정하면, 한 행이 없어진 것처럼 보입니다. 작게 축소되었기 때문이죠. 다른 행을 1*로 설정하면, 축소된 행이 보일 거예요.

Button과 CheckBox를 추가합니다.

도구 상자에서 Button 컨트롤 아래에 CheckBox 컨트롤이 있습니다. Button의 이름을 changeText로, CheckBox의 이름을 enableCheckbox로 설정하세요. 두 컨트롤의 텍스트를 바꾸기 위해, 오른쪽을 클릭하고 텍스트 편집(Edit Text) 메뉴를 선택해서 바꿔주세요(텍스트 편집을 끝내려면 Esc를 누르면 됩니다). 각 컨트롤에서 오른쪽을 클릭하고, 레이아웃 > 모두 다시 설정을 선택한 후, 속성 창에서 VerticalAlignment와 HorizontalAlignment를 Center로 설정하세요.



TextBlock을 추가합니다.

이전 프로젝트에서 창 아래에 TextBlock을 추가한 것과 같습니다. 이름을 labelToChange로 하고 Grid.Row 속성을 1로 설정해 주세요.

Checkbox가 체크되지 않은 상태에서 버튼을 클릭하면, 이런 메시지가 뜹니다.

Checkbox가 체크되었는지 확인하기 위한 조건 테스트는 다음과 같습니다.

```
enableCheckbox.IsChecked == true
```

Text changing is disabled

테스트가 true가 아니면, 프로그램은 두 개의 선언문을 실행해야 합니다.

```
labelToChange.Text = "Text changing is disabled";
labelToChange.HorizontalAlignment = HorizontalAlignment.Center;
```

힌트: 이것을 else 블록에 넣으면 돼요.

Checkbox가 체크된 상태에서 Button을 클릭하면, TextBlock이 왼쪽에서 Left가 보이고, 오른쪽에서 Right가 보입니다.

레이블의 Text 속성이 현재 "Right"면, 텍스트를 "Left"로 바꾸고 HorizontalAlignment 속성을 HorizontalAlignment.Left로 바꿔야 합니다. 그렇지 않다면 텍스트를 "Right"로 바꾸고 HorizontalAlignment 속성을 HorizontalAlignment.Right로 바꿔야 합니다. 여러분이 Checkbox가 체크된 상태에서 버튼을 클릭했을 때, 레이블이 앞뒤로 움직여야 합니다.



연습문제 정답

if/else문 연습을 해 봅시다. 아래의 프로그램을 만들어 보세요.

가독성을 위해 들여쓰기를 했습니다.

그리드에 대한 XAML 코드입니다.

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>

  <Button x:Name="changeText" Content="Change the label if checked"
    HorizontalAlignment="Center" VerticalAlignment="Center"
    Click="changeText_Click"/>

  <CheckBox x:Name="enableCheckbox" Content="Enable label changing"
    HorizontalAlignment="Center" VerticalAlignment="Center"
    IsChecked="true" Grid.Column="1"/>

  <TextBlock x:Name="labelToChange" Grid.Row="1" TextWrapping="Wrap"
    Text="Press the button to set my text"
    HorizontalAlignment="Center" VerticalAlignment="Center"
    Grid.ColumnSpan="2"/>
</Grid>
```

디자이너에서 이름을 설정하기 전에 Button을 더블-클릭했다면, changeText_Click() 대신, Button_Click_1()이라는 이벤트 핸들러 메서드가 생성됩니다.

이벤트 핸들러 메서드에 대한 C# 코드입니다.

```
private void changeText_Click(object sender, RoutedEventArgs e)
{
  if (enableCheckbox.IsChecked == true)
  {
    if (labelToChange.Text == "Right")
    {
      labelToChange.Text = "Left";
      labelToChange.HorizontalAlignment = HorizontalAlignment.Left;
    }
    else
    {
      labelToChange.Text = "Right";
      labelToChange.HorizontalAlignment = HorizontalAlignment.Right;
    }
  }
  else
  {
    labelToChange.Text = "Text changing is disabled";
    labelToChange.HorizontalAlignment = HorizontalAlignment.Center;
  }
}
```

책의 다음 부분에서는 XAML을 사용하지 않습니다.

2장의 나머지 부분에서는 윈도우 8을 요구하지 않습니다. 윈도우 2003을 사용하거나 Visual Studio 2010으로 실습할 수 있죠. 그리고 3장부터 9장까지는 윈도우 폼 응용 프로그램(혹은 원폼) 프로젝트를 사용합니다. 이 프로젝트는 데스크톱 앱을 만들기 위해서 스토어 앱보다 오래된 기술을 사용합니다. 여러분이 IDE를 통해서 C#과 XAML을 배우고, 경험한 것처럼 잠시 동안 책으로 돌아가서 원폼에 대해 배워 봅시다.

← 책 131쪽을 살펴보세요. C#의 기본 개념을 익히기 위해서 원폼으로 전환하는 이유에 대해서 설명합니다.

↑ WPF도 마찬가지입니다. 원폼 프로젝트를 통해 C#의 핵심 개념을 빠르게 익힐 수 있습니다. 그리고 WPF를 학습하는 지름길이기도 하죠.

10장이 되기 전까지는 윈도우 8과 WPF가 필요 없단 말이죠? 왜 최신 기술을 사용하지 않는 거죠?



때론 오래된 기술이 좋은 학습 도구로 사용됩니다.

WPF는 데스크톱 앱을 만드는 좋은 도구입니다. 하지만, 조금 더 간단한 기술이 C#의 개념을 쉽게 익히는 데 도움을 줍니다. 그리고 원폼을 사용하는 또 다른 이유가 있습니다. 여러 가지 기술을 익히면서 공통점과 차이점을 비교하며, 기술에 대한 통찰력을 얻을 수 있습니다. 책 132쪽부터 원폼 앱을 만들며, C#의 기초를 다져 봅시다. 그리고 10장에서 다시 XAML로 돌아왔을 때, 여러분이 학습한 기초를 기반으로 조금 더 쉽게 WPF에 접근할 수 있습니다.



조심하세요

일부 몇몇 장은 Visual Studio 2008에서 지원되지 않는 .NET 4.0에 있는 C# 기능들을 사용합니다.

Visual Studio 2008을 사용하는 경우, 이 책의 3장 끝에 도달했을 때, 몇 가지 문제가 발생할 수 있습니다. 2008의 .NET 프레임워크의 최신 버전은 3.5입니다. 책에 있는 예제들은 .NET 4.0의 C# 기능들을 사용합니다. 3장에서 객체 인터페이스, 8장에서 컬렉션 인터페이스와 공변성(covariance)을 배웁니다. Visual Studio 2008에서는 이 기능들이 아직 추가되지 않았기 때문에, 책에 있는 예제 코드들은 컴파일되지 않습니다. 2008보다 더 새로운 비주얼 스튜디오 버전을 설치할 수 없다면, 대부분의 예제는 실행할 수 있지만, 위에서 언급한 C#의 기능들을 사용할 수 없습니다.

부록에서 두 페이지 모드를 유지하기 위해서 일부러 페이지를 비웠습니다. 연습문제와 연습문제 정답이 서로 마주하는 면에 나오면 안 되기 때문이죠. 그리고 두 페이지 모드의 짝수 쪽에는 오른쪽 위에, 홀수 쪽에는 왼쪽 위에 작은 캡션이 있습니다.

Chapter 10

이번 장에서는 책에 있는 예제들을 WPF로 재설계하여 개발해 봅시다.



권종 앱을 WPF 앱으로 포팅할 수 있습니다.

3~9장의 연습문제와 실습을 통해 많은 양의 코드를 작성했습니다. 이번 장에서는 책에서 본 일부 코드를 사용해서 WPF를 익혀 봅시다.

10장에서 여러분이 참고해야 할 사항입니다.

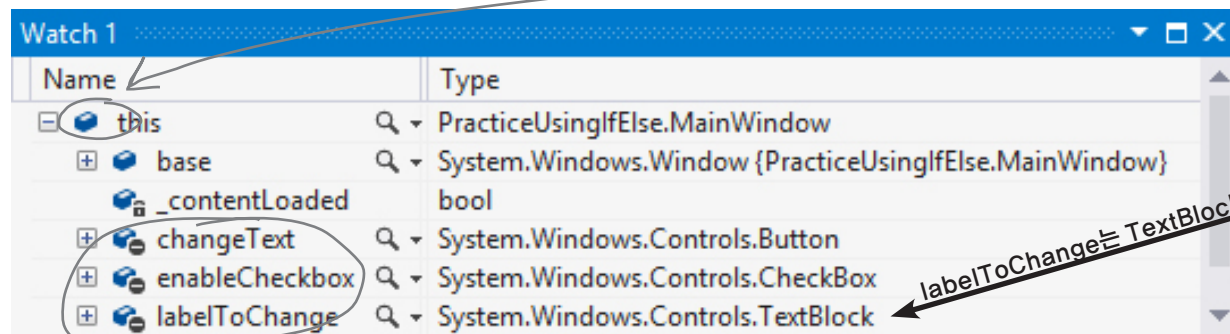
- ★ 책의 10장 처음부터 541쪽까지 읽어 주세요. “직접 해 봅시다!”, “연필을 깎으며”와 같은 문제들이 있습니다.
- ★ 이 부록은 책의 542-549쪽을 대체합니다.
- ★ 책 550쪽은 윈도우 스토어 프로젝트만 해당합니다. WPF에는 해당되지 않지만, 읽어도 됩니다.
- ★ 그리고, 책의 553-555쪽을 대체합니다.
- ★ 마지막으로 책 558, 559쪽을 읽고 난 후, 부록으로 돌아와서 나머지 부분(560-577쪽)을 읽으면 됩니다.

윈도우 스토어 앱은 XAML로 UI 객체를 만듭니다

WPF 응용 프로그램의 사용자 인터페이스를 구축하기 위해 XAML을 사용하면 객체 그래프를 만들 수 있습니다. 원본과 같이 IDE의 조사식 창을 볼 수 있습니다. 2장의 "기초부터 만들기"에서 연습문제로 한 Program2를 열어 봅시다. 그리고 MainWindow.xaml.cs를 열고 생성자에서 InitializeComponent()를 호출하는 곳에 중단점을 설정해서 앱의 UI 객체를 탐험해 봅시다.

** 직접 해
봅시다!*

1 디버깅을 시작해 봅시다. F10키 눌러 프로시저 단위로 실행해 봅시다. 디버그(Debug) > 창(Windows) > 조사식(Watch) > 조사식 1(Watch 1)을 누르세요. 조사식 창에서 this를 살펴봅시다.



labelToChange는 TextBlock의 인스턴스입니다.

2 페이지에 정의된 XAML 코드를 살펴봅시다.

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>

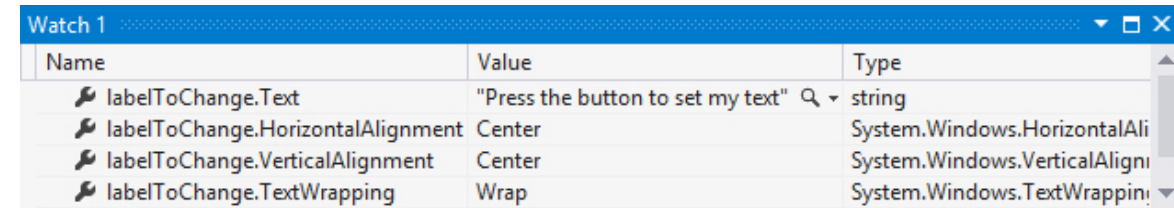
  <Button x:Name="changeText" Content="Change the label if checked"
    HorizontalAlignment="Center" Click="changeText_Click"/>

  <CheckBox x:Name="enableCheckbox" Content="Enable label changing"
    HorizontalAlignment="Center" IsChecked="true"
    Grid.Column="1"/>

  <TextBlock x:Name="labelToChange" Grid.Row="1" TextWrapping="Wrap"
    Text="Press the button to set my text"
    HorizontalAlignment="Center" VerticalAlignment="Center"
    Grid.ColumnSpan="2"/>
</Grid>
```

*윈도우의
컨트롤을
정의하는 XAML
은 필드와니
컨트롤에 대한
참조를 포함하는
속성을 가진
Window 객체에
설정되어
있습니다.*

3 몇몇의 labelToChange 속성을 조사식 창에 추가해 봅시다.



앱은 XAML에 있는 기본 속성들을 자동으로 설정해 줍니다.

```
<TextBlock x:Name="labelToChange" Grid.Row="1" TextWrapping="Wrap"
  Text="Press the button to set my text"
  HorizontalAlignment="Center" VerticalAlignment="Center"
  Grid.ColumnSpan="2"/>
```

그러나 조사식 창에서 labelToChange.Grid 혹은 labelToChange.ColumnSpan을 추가해 봅시다. 이 컨트롤은 Windows.UI.Controls.TextBlock 객체이고, 이 객체는 이 속성들을 가지고 있지 않습니다. 이 XAML 속성은 무엇일까요?

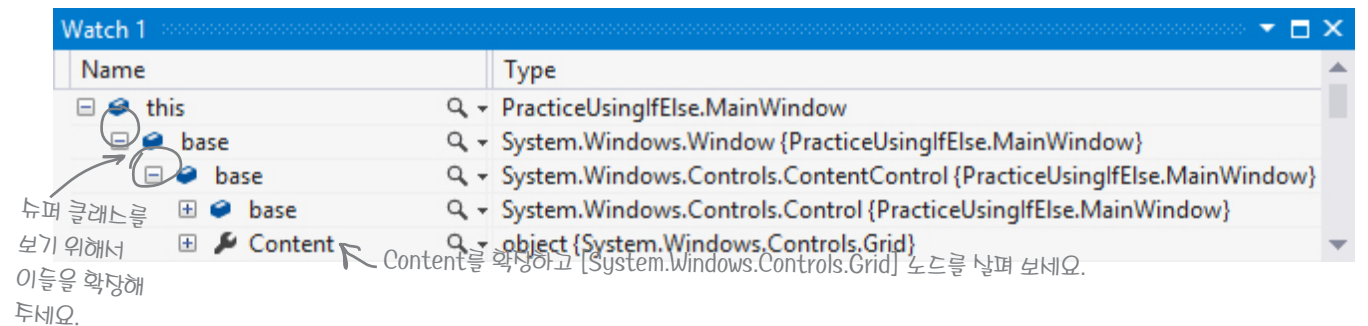
*클래스를 보기
위해 Window 위에
마우스를 올려 보세요.*

4 프로그램을 중단하고 MainWindow.xaml.cs 파일을 열어 클래스가 선언된 곳에서 MainWindow를 찾아보세요. 이 클래스는 Window의 서브 클래스입니다. IDE에서 Window라 적힌 곳에 마우스를 놓고 클래스의 전체 이름을 보세요.

```
public partial class MainWindow : Window
{
  public MainWindow()
  {
    InitializeComponent();
  }
}
```

창과 대화 상자의 수명을 생성, 구성, 표시 및 관리하는 기능을 제공합니다.

이제 프로그램을 다시 시작해 봅시다. InitializeComponent()를 호출하는 부분에 중단점을 설정하고 F10키를 눌러 프로시저 단위로 실행해 봅시다. 조사식 창으로 돌아와서 this > base > base를 확장하며 상속 계층구조를 살펴봅시다.



*슈퍼 클래스를
보기 위해서
이들을 확장해
주세요.*

XAML이 생성한 객체를 잠깐 동안 살펴봅시다. 이 객체들은 책의 뒤에서 자세히 살펴볼 겁니다. 여기에서는 앱 뒤에 숨어 있는 얼마나 많은 객체들이 있는지만 알고 넘어갑시다.

고피시 폼을 윈도우 WPF 응용프로그램으로 새롭게 단장해 봅시다

8장에서 만든 고피시(Go Fish!) 게임을 WPF 응용 프로그램으로 만들어 봅시다. Visual Studio 2013 for Desktop을 열어 새로운 WPF 응용 프로그램을 생성합니다("Save the Humans"를 생성했던 것처럼요). 다음에 나오는 몇 페이지들을 XAML로 새롭게 디자인해서, 원폼보다 더 유연한 메인 윈도우를 만들어 볼 겁니다. 폼의 윈도우 데스크톱 컨트롤을 사용하는 대신 WPF XAML 컨트롤을 사용해 봅시다.

*** 직접 해 봅시다! ***

이것은 <TextBox/> 가 될 거예요.

이것은 <Button/> 이 될 거고요.

TextBox와 Button 컨트롤을 그룹으로 묶기 위해 두명 StackPanel 을 사용합니다. 그래서 이 둘은 그리드의 같은 셀 안에 있군요.

이것은 <Listbox/> 가 될 거예요.

이것은 <Scrollview-er/>가 될 거예요.

이것도 <Scrollview-er/>가 될 거예요.

이것은 도구상자에 있는 또 다른 컨트롤입니다. 텍스트가 한 범위를 벗어났다면 수직/수평 방향의 스크롤 바를 추가해서 텍스트를 보여 줍니다.

이것은 <Button/> 이 될 거고요.



앱 메인 윈도우는 아래와 같습니다.

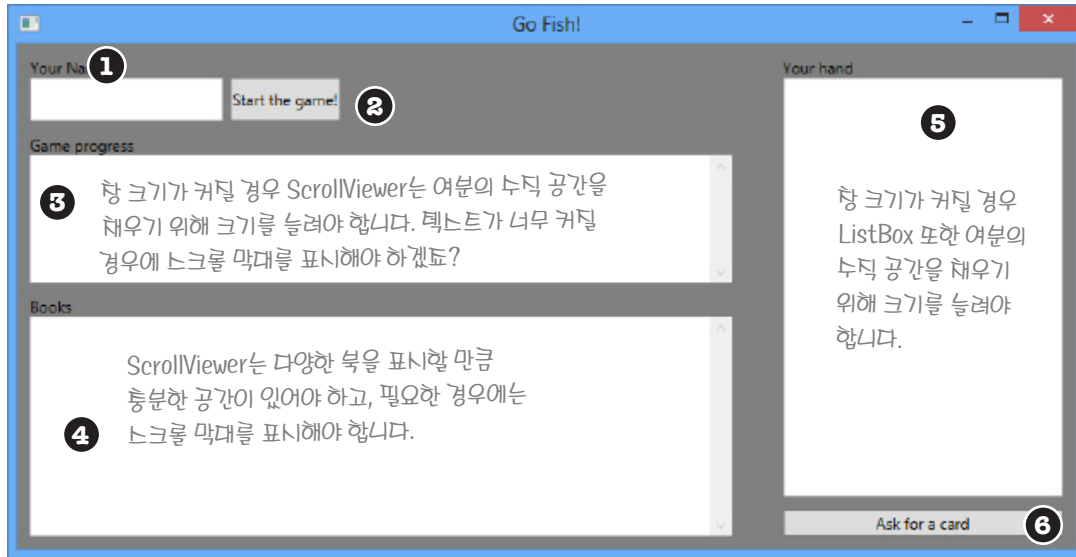
게임에서 코드의 대부분은 동일하게 유지되지만, UI 코드가 바뀝니다.

창의 크기에 따라 확장하거나 축소되는 그리드의 행과 열들은 컨트롤을 포함하고 있습니다. 창 크기에 맞추기 위해서 게임 화면을 늘리거나 줄일 수 있습니다.

창 크기에 상관 없이 게임이 실행됩니다.

창 레이아웃과 컨트롤

XAML과 WPF 앱은 하나의 공통점이 있습니다. 둘 모두 창 레이아웃에서 컨트롤에 의존하고 있습니다. 고피시 창에는 2개의 Button과 카드를 보여 주기 위한 ListBox, 사용자의 이름을 입력받기 위한 TextBox, 네 개의 TextBlock 레이블, 두 개의 ScrollViewer 컨트롤이 있습니다. 게임 진행 상황(Game progress)과 북(Books)의 컨트롤은 흰 배경입니다.



메인 윈도우에 대한 XAML은 <Window> 태그로 시작합니다. Title 속성은 창의 제목을 “Go Fish!”로, Height와 Width 속성은 창 크기를 설정해 주죠(디자이너에서 창 크기를 바꿀 때마다, 속성이 변하는 것을 볼 수 있습니다). 그리고 Background 속성을 회색인 Gray로 설정해 주세요.

XAML에 반영된 <Window>의 시작 태그입니다. 이 프로젝트의 이름을 “GoFish”로 지었죠. 만약 다른 이름을 입력했다면, x:Class 속성에 여러분이 입력한 이름이 보일 겁니다.

```
<Window x:Class="Go_Fish.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Go Fish!" Height="500" Width="525" Background="Gray">
  <Grid Margin="10" >
    StackPanel을 이용해서 플레이어 이름을 입력할 TextBox와 시작 Button을 한 셀에 놓습니다.
```

```
1 <TextBlock Text="Your Name" />
```

```
<StackPanel Orientation="Horizontal" Grid.Row="1">
  <TextBox x:Name="playerName" FontSize="24" Width="150" />
```

```
2 <Button x:Name="startButton" Margin="5,0"
Content="Start the game!"/>
</StackPanel>
```

Margin 속성에서 두 개의 숫자가 있는 경우 누평(왼쪽/오른쪽)과 누직(위/아래) 여백을 지정합니다. 여기에서는 누평 여백을 5, 누직 여백을 5로 설정했습니다. 그리고 5,0,0,0를 입력해서 왼쪽의 여백을 5, 오른쪽의 여백을 0으로 설정할 수도 있습니다.

창의 각 레이블은(“Your name”, “Game progress” 등) TextBlock입니다. Margin 속성에서 위쪽 여백을 10으로 설정해 주세요.

```
<TextBlock Text="Game progress" Grid.Row="2"
Margin="0,10,0,0"/>
```

ScrollViewer 컨트롤은 게임 진행 상황(Game Progress)을 표시해 줍니다. 텍스트가 길어질 경우 스크롤 바가 나타납니다.

```
3 <ScrollViewer Grid.Row="3" FontSize="24"
Background="White" Foreground="Black" />
```

북을 표시하기 위한 또 다른 TextBlock과 ScrollViewer 컨트롤이 있네요. ScrollViewer 컨트롤의 기본 수평 및 수직 정렬은 Stretch(늘림)로 되어 있습니다. 정말 유용한 컨트롤이죠. ScrollViewer 컨트롤이 다양한 화면 크기에 맞게 확장할 수 있도록, 행과 열을 설정해 봅시다.

```
4 <TextBlock Text="Books"
Margin="0,10,0,0" Grid.Row="4"/>
```

```
<ScrollViewer FontSize="24" Background="White" Foreground="Black"
Grid.Row="5" Grid.RowSpan="2" />
```

컨트롤 사이의 조그만 공간을 위해서 열의 여백을 40픽셀로 둡니다. ListBox와 Button의 컨트롤은 세 번째 열로 배치해야겠군요. ListBox의 행은 2행과 6행 사이에 걸쳐 있어서, Grid.Row="1"과 Grid.RowSpan="5"로 설정했습니다. 창이 커질 때 ListBox도 같이 커집니다.

행과 열의 인덱스는 0으로 시작해서 네 번째 열에 있는 컨트롤은 Grid.Column="2"과 Grid.RowSpan="5"로 설정했습니다.

```
5 <TextBlock Text="Your hand" Grid.Row="0" Grid.Column="2" />
```

```
<ListBox x:Name="cards" Background="White" FontSize="24"
Height="Auto" Margin="0,0,0,10"
Grid.Row="1" Grid.RowSpan="5" Grid.Column="2"/>
```

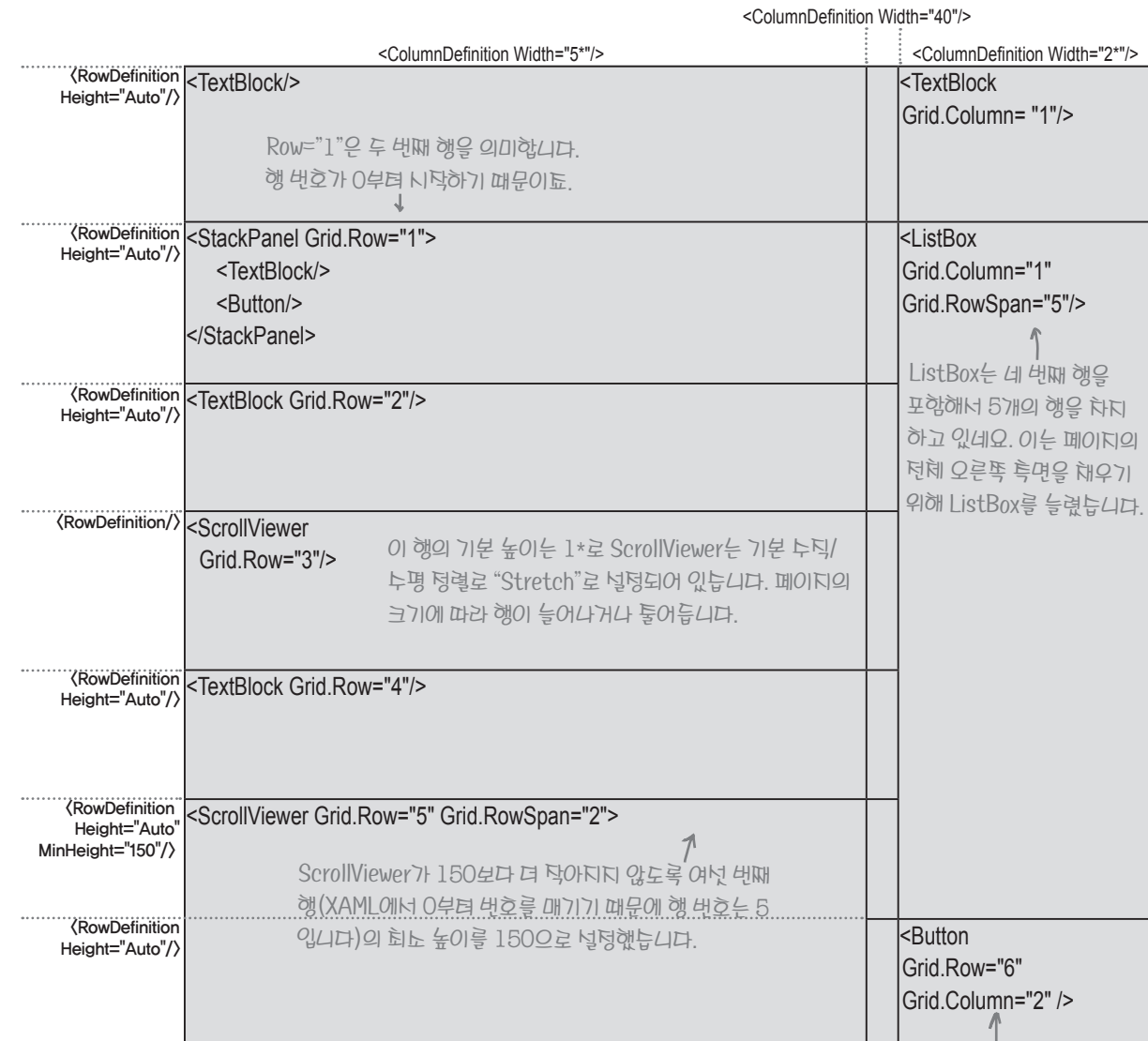
“Ask for a card” Button의 HorizontalAlignment와 VerticalAlignment는 Stretch로 설정했습니다. 그래서 버튼이 셀 안에 채워져 있군요. 그리고 ListBox 컨트롤 아래에 20픽셀의 작은 틈이 있습니다.

```
6 <Button x:Name="askForACard" Content="Ask for a card"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
Grid.Row="6" Grid.Column="2"/>
```

다음 페이지에서 그리드를 마무리해 봅시다. →

창 크기에 맞게 행과 열이 조절됩니다

그리드는 창 레이아웃을 위한 매우 효과적인 도구입니다. 다양한 장치의 페이지를 디자인하는 데 유용하죠. *로 끝나는 높이와 너비는 항상 **자동으로** 각각 다른 화면에 맞게 **조정합니다**. 고피시 페이지는 3개의 열이 있습니다. 첫 번째와 세 번째의 너비는 5*와 2*입니다. **그리드가 늘어나거나 줄어들어도** 항상 5:2의 비율을 유지하죠. 두 번째 열은 첫 번째와 세 번째의 분리를 유지하기 위해 40픽셀로 고정된 폭을 갖습니다. 페이지 안에 있는 컨트롤을 포함해서 행과 열을 배치하는 방법이 아래에 있습니다.



XAML에서의 행과 열 번호는 0부터 시작합니다. 이 버튼의 행은 6, 열은 2입니다. 그리고 수직/수평 정렬은 Stretch로 설정되어 셀의 공간을 모두 차지하죠. 이 행의 높이(RowDefinitionHeight)는 자동(Auto)으로 되어 있어서 내용의 길이에 따라 높이가 결정됩니다(버튼의 여백 포함).

창 레이아웃을 작업하기 위한 행과 열의 정의 부분입니다.

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="5*" />
  <ColumnDefinition Width="40" />
  <ColumnDefinition Width="2*" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
  <RowDefinition Height="Auto" />
  <RowDefinition Height="Auto" />
  <RowDefinition Height="Auto" />
  <RowDefinition Height="Auto" />
  <RowDefinition Height="Auto" MinHeight="150" />
  <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
</Grid>
</Window>
```

첫 번째 열은 항상 세 번째 열보다 2.5배 넓네요(5:2의 비율). 그리고 그 사이 40 픽셀의 열이 있습니다. 데이터를 표시하는 ScrollView와 ListBox 컨트롤의 HorizontalAlignment는 Stretch로 설정되어서 열을 채워 줍니다.

네 번째 행의 기본 높이는 1*입니다. 다른 행에서 사용하지 않는 공간을 채우기 위해서 행이 늘어나거나 줄어들 수 있죠. ListBox와 첫 번째 ScrollView가 이 행에 걸쳐 있어서, 늘어나거나 줄어들 수 있습니다.

대부분 모든 행의 높이는 Auto로 되어 있습니다. 어떤 하나의 행이 늘어나거나 줄어들면, 그 행에 속한 컨트롤 또한 늘어나거나 줄어듭니다.

그리드에서 컨트롤 위 혹은 아래에 행과 열의 정의를 추가할 수 있습니다. 여기에선 컨트롤 아래에 정의를 추가했습니다.

마지막으로 WPF 애플의 레이아웃을 마무리하는 그리드와 윈도우의 닫는 태그를 추가합니다.



XAML을 이용하여 원본 프로그램을 WPF 응용 프로그램으로 다시 만들어 봅시다. 새 WPF 응용 프로그램을 생성하고, 각 창의 그리드를 수정하고 컨트롤을 추가합니다. 앱이 실행될 필요는 없습니다. 그냥 XAML을 생성한 뒤, 아래 그림에 맞게 만들어 주세요.

```
<Window x:Class="BeehiveManagementSystem.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Beehive Management System"
Height="300" Width="525"
ResizeMode="NoResize">
    <StackPanel Margin="5">
        <TextBlock Text="Worker Bee Assignments" Margin="0,0,0,5" />
        <Border BorderThickness="1" BorderBrush="Black">
            <StackPanel Orientation="Horizontal" Margin="5">
                <StackPanel Margin="0,0,10,0">
                    <TextBlock Text="Job"/>
                    <ComboBox SelectedIndex="0">
                        <ComboBoxItem Content="Baby bee tutoring"/>
                        <ComboBoxItem Content="Egg care"/>
                        <ComboBoxItem Content="Hive maintenance"/>
                        <ComboBoxItem Content="Honey Manufacturing"/>
                        <ComboBoxItem Content="Nectar collector"/>
                        <ComboBoxItem Content="Sting patrol"/>
                    </ComboBox>
                </StackPanel>
                <StackPanel>
                    <TextBlock Text="Shifts" />
                    <TextBox/>
                </StackPanel>
                <Button Content="Assign this job to a bee"
                    VerticalAlignment="Bottom" Margin="10,0,0,0" />
            </StackPanel>
        </Border>

        <Button Content="Work the next shift" Margin="0,20,20,0"
            FontSize="18"
            HorizontalAlignment="Right" />

        <TextBlock Text="Shift report" Margin="0,10,0,5"/>
        <Border BorderBrush="Black" BorderThickness="1" Height="100">
            <ScrollViewer
                Content="
Report for shift #20&#13;
Worker #1 will be done with 'Nectar collector' after this shift&#13;
Worker #2 finished the job&#13;
Worker #2 is not working&#13;
Worker #3 is doing 'Sting patrol' for 3 more shifts&#13;
Worker #4 is doing 'Baby bee tutoring' for 6 more shifts
                "/>
        </Border>
    </StackPanel>
</Window>
```

우리가 둔 여백입니다. 숫자 하나(5)만 있다면, 위쪽, 왼쪽, 아래, 오른쪽의 여백이 같은 값으로 설정됩니다.

XAML 코드가 연습문제 정답과 다르다고요? XAML에서 매우 유사한 (또한 동일한) 창을 표시하는 여러 가지 방법이 있습니다.
그리고 XAML은 태그 순서 처리에 유연합니다. 같은 윈도우 객체 그래프에 있다면, 태그를 다른 순서로 배치할 수 있습니다.

Border 컨트롤은 ScrollViewer 컨트롤 두위에 테두리를 그려줍니다.

근무시간 보고서를 채우는 데 사용되는 빈 데이터입니다. Content 속성은 둘 바꿈 문자를 무시합니다. 대신 을 사용하면 됩니다.

```
<Window x:Class="BreakfastForLumberjacks.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Breakfast for Lumberjacks"
Width="525" Height="400"
MinWidth="300" MinHeight="350"
ResizeMode="CanResizeWithGrip" />
```

여기에 윈도우 속성이 설정되어 있습니다. 처음 창의 크기는 525 x 400이고, 최소 창 크기는 300 x 350입니다.

크기가 재조정되는 것을 막기 위해서 **ResizeMode** 속성을 **NoResize**로 설정할 수 있습니다. **CanMinimize**는 최소화만 할 수 있고, **CanResizeWithGrip**은 창의 오른쪽 아래의 코너에 크기를 조절할 수 있는 그림을 표시해 줍니다.

```
<Grid Grid.Row="1" Margin="5">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
```

```
<TextBlock Text="Lumberjack name" Margin="0,0,0,5" />
<TextBox Grid.Row="1"/>

<TextBlock Grid.Row="2" Text="Breakfast line" Margin="0,10,0,5" />
<ListBox Grid.Row="3" VerticalAlignment="Stretch">
    <ListBoxItem Content="1. Ed"/>
    <ListBoxItem Content="2. Billy"/>
    <ListBoxItem Content="3. Jones"/>
    <ListBoxItem Content="4. Fred"/>
    <ListBoxItem Content="5. Johansen"/>
    <ListBoxItem Content="6. Bobby, Jr." />
</ListBox>
```

연습문제의 일화로 품이 마치 동작되는 것처럼 보이는 외형을 만들기 위해서 더미 항목을 추가했습니다. 여기에 있는 ListBox와 같은 컨트롤을 클래스의 속성들과 어떻게 바인딩 하는지 배울 거예요.

```
<TextBlock Grid.Row="4" Text="Feed a lumberjack" Margin="0,10,0,5" />
<StackPanel Grid.Row="5" Orientation="Horizontal">
    <TextBox Text="2" Margin="0,0,10,0" Width="30"/>
    <ComboBox SelectedIndex="0" Margin="0,0,10,0">
        <ComboBoxItem Content="Crispy"/>
        <ComboBoxItem Content="Soggy"/>
        <ComboBoxItem Content="Browned"/>
        <ComboBoxItem Content="Banana"/>
    </ComboBox>
    <Button Content="Add flapjacks" />
</StackPanel>
```

```
<Border BorderThickness="1" BorderBrush="Gray" Grid.Row="6" Margin="0,5,0,0">
    <ScrollViewer Content="Ed has 7 flapjacks"
        BorderThickness="2" BorderBrush="White"
        MinHeight="50"/>
</Border>
```

더미 내용입니다.

```
<StackPanel Grid.Row="7" Orientation="Horizontal" Margin="0,10,0,0">
    <Button Content="Add Lumberjack" Margin="0,0,10,0" />
    <Button Content="Next Lumberjack" />
</StackPanel>
```

```
</Grid>
</Window>
```

데이터 바인딩으로 슬라피 조의 향상된 메뉴판을 만들어 봅시다

4장에서 나온 슬라피 조를 기억하시나요? 슬라피 조는 여러분이 XAML에 능숙하다고 알고 있습니다. 그는 샌드위치 메뉴를 위한 WPF 앱을 만들고 싶어 합니다.

우리가 만들려고 하는 페이지입니다.

ListView와 TextBlock의 내부에 있는 <Run> 태그를 채우기 위해 단방향 데이터 바인딩을 사용합니다. 그리고 실제 바인딩을 수행하기 위해서 <Run> 태그를 이용하여, TextBox에 양방향 바인딩을 사용합니다.

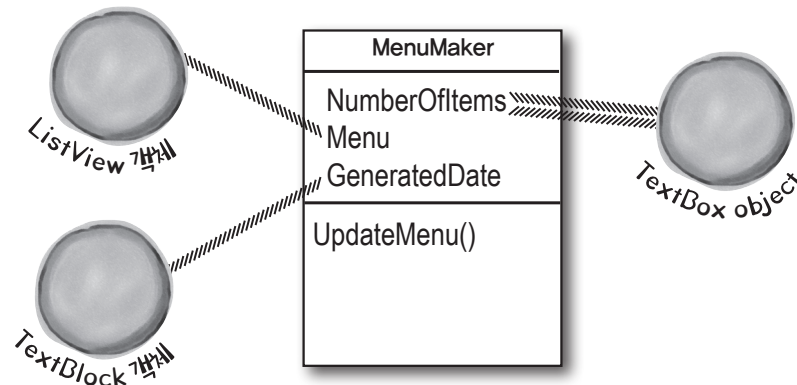
The screenshot shows a window titled "Welcome to Sloppy Joe's" with a "Number of items" input field set to 10 and a "Make a new menu" button. Below is a list of menu items and a timestamp. The XAML code snippet shows the following structure:

```

<StackPanel Grid.Row="1" Margin="120,0">
  <StackPanel Orientation="Horizontal">
    <TextBlock/>
    <TextBox Text="{Binding NumberOfItems, Mode=TwoWay}">
  </StackPanel>
  <Button/>
</StackPanel>
<ListView ItemsSource="{Binding Menu}">
  <TextBlock/>
  <Run/>
  <Run Text="{Binding GeneratedDate}">
</TextBlock>
</StackPanel>
    
```

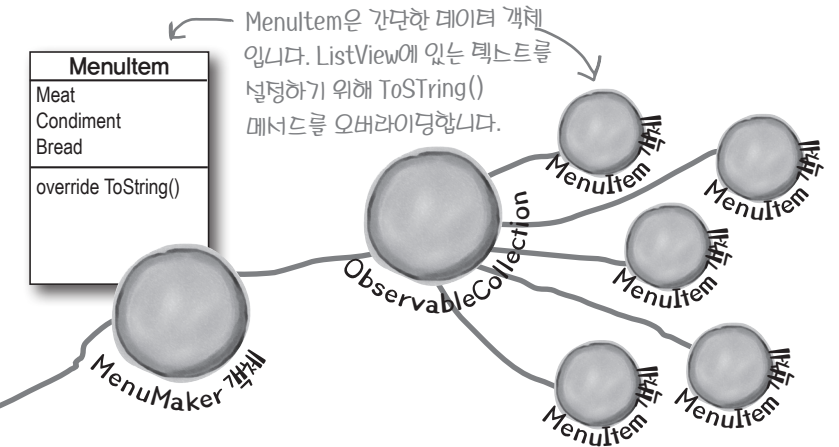
바인딩할 속성이 있는 객체가 필요합니다.

Windows 객체는 3개의 public 속성을 가진 MenuMaker 클래스의 인스턴스가 있습니다. int형의 NumberOfItems와 ObservableCollection 유형의 Menu, DateTime 유형인 GeneratedDate가 있죠.



Window 객체는 데이터 컨텍스트를 사용하기 위해 MenuMaker의 인스턴스를 생성합니다.

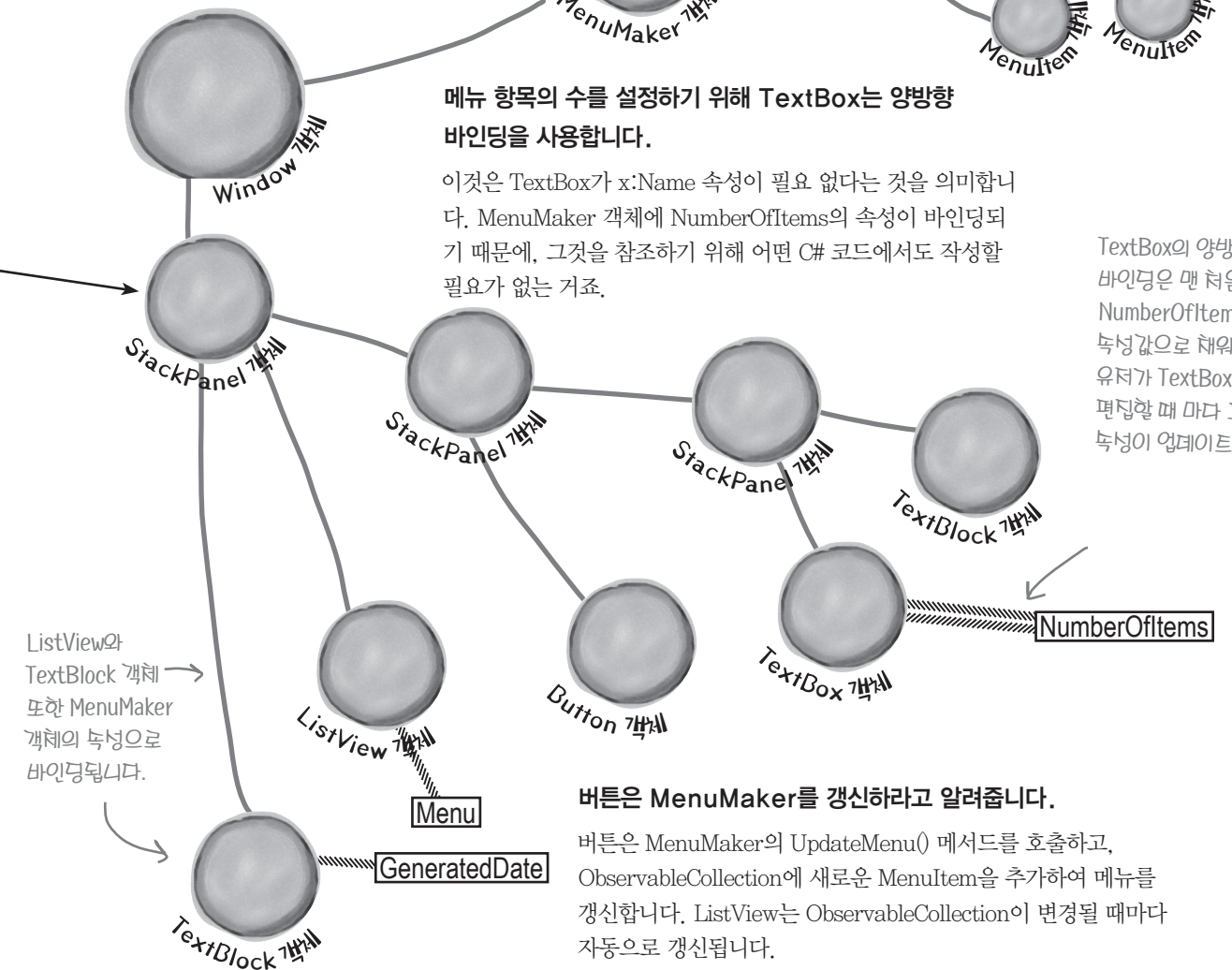
Window 객체의 생성자는 StackPanel의 DataContext 속성을 MenuMaker의 인스턴스로 설정합니다. 이제 바인딩은 XAML에서 실행됩니다.



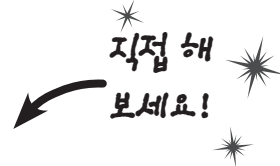
메뉴 항목의 수를 설정하기 위해 TextBox는 양방향 바인딩을 사용합니다.

이것은 TextBox가 x.Name 속성이 필요 없다는 것을 의미합니다. MenuMaker 객체에 NumberOfItems의 속성이 바인딩되기 때문에, 그것을 참조하기 위해 어떤 C# 코드에서도 작성할 필요가 없는 거죠.

TextBox의 양방향 바인딩은 맨 처음 NumberOfItems 속성값으로 채워지고, 유저가 TextBox 값을 편집할 때 마다 그 속성이 업데이트 됩니다.



이제 코딩할 차례입니다. 페이지를 넘겨 코드를 보기 전에, 지금까지 읽은 것을 바탕으로 새롭고 향상된 슬라피 조 앱을 만들 수 있나요?



1 먼저, 프로젝트를 생성합니다.

새로운 WPF 응용 프로그램을 생성합니다. 기본 창 크기로 유지하고, 제목을 "Welcome to Sloppy Joe's"로 합니다.

2 MenuMaker 클래스를 추가합니다.

4장에서부터 여기까지 먼 길을 왔습니다. 속성으로 항목 수를 설정하여 잘 캡슐화된 클래스를 만들어 봅시다. 생성자에 MenuItem의 ObservableCollection을 생성합니다. 이 Menu는 UpdateMenu() 메서드가 호출될 때마다 갱신됩니다. 이 메서드는 현재 Menu에 대한 타임 스탬프와 GeneratedDate라 불리는 DateTime 속성을 갱신합니다. 프로젝트에 MenuMaker 클래스를 추가해 봅시다.

다른 프로젝트에서 했던 것처럼 솔루션 탐색기에서 프로젝트 이름에 마우스 오른쪽 버튼을 클릭하고, 새 클래스를 추가하면 됩니다.

윈도우에 있는 이러한 속성들을 보여 주기 위해서 데이터 바인딩을 사용할 거예요. 또한 NumberOfItems를 갱신하기 위해 양방향 바인딩을 사용할 겁니다.

```
using System.Collections.ObjectModel;

class MenuMaker {
    private Random random = new Random();
    private List<String> meats = new List<String>()
        { "Roast beef", "Salami", "Turkey", "Ham", "Pastrami" };
    private List<String> condiments = new List<String>() { "yellow mustard",
        "brown mustard", "honey mustard", "mayo", "relish", "french dressing" };
    private List<String> breads = new List<String>() { "rye", "white", "wheat",
        "pumpernickel", "italian bread", "a roll" };

    public ObservableCollection<MenuItem> Menu { get; private set; }
    public DateTime GeneratedDate { get; set; }
    public int NumberOfItems { get; set; }

    public MenuMaker() {
        Menu = new ObservableCollection<MenuItem>(); 새롭게 추가된 CreateMenuItem() 메서드는
        NumberOfItems = 10; 문자열이 아닌 MenuItem 객체를 반환합니다.
        UpdateMenu(); 원하는 경우 쉽게 항목이 표시되는 방식을 바꿀
    }
    private MenuItem CreateMenuItem() {
        string randomMeat = meats[random.Next(meats.Count)];
        string randomCondiment = condiments[random.Next(condiments.Count)];
        string randomBread = breads[random.Next(breads.Count)];
        return new MenuItem(randomMeat, randomCondiment, randomBread);
    }
    public void UpdateMenu() {
        Menu.Clear();
        for (int i = 0; i < NumberOfItems; i++) {
            Menu.Add(CreateMenuItem());
        }
        GeneratedDate = DateTime.Now;
    }
}
```

이 using 라인이 필요할 거예요. ObservableCollection<T>가 이 네임스페이스에 있기 때문이죠.

← 누 있습니다.

어떻게 이 반복문이 동작하는지 자세히 살펴보세요. 새로운 MenuItem 컬렉션을 생성하지 않습니다. 새로운 항목을 추가하여 현재 상태를 갱신합니다.

NumberOfItems가 음수로 설정되면 무슨 일이 일어날까요?

날짜에 관한 건 DateTime을 사용하세요.
날짜를 저장하기 위해서 DateTime 유형을 사용했습니다. DateTime은 날짜와 시간을 생성하고 수정하는 데도 사용됩니다. DateTime에서 현재 시간을 반환해 주는 static 속성인 Now가 있습니다. 또한 DateTime은 AddSeconds() 메서드처럼 초 단위, 밀리 초 단위, 일별로 추가하거나 변환할 수 있습니다. 그리고 Hour와 DayOfWeek과 같은 속성으로 날짜를 분리할 수 있습니다.

3 MenuItem 클래스를 추가합니다.

데이터를 저장하기 위해 문자열 대신 클래스를 이용하는 경우 조금 더 유연한 프로그램을 만들 수 있다는 것을 살펴봤습니다. 메뉴 항목을 추가하는 간단한 클래스가 아래에 있네요. 여러분이 좋아하는 메뉴도 추가해 보세요.

```
class MenuItem {
    public string Meat { get; set; }
    public string Condiment { get; set; }
    public string Bread { get; set; }

    public MenuItem(string meat, string condiment, string bread) {
        Meat = meat;
        Condiment = condiment;
        Bread = bread;
    }

    public override string ToString() {
        return Meat + " with " + Condiment + " on " + Bread;
    }
}
```

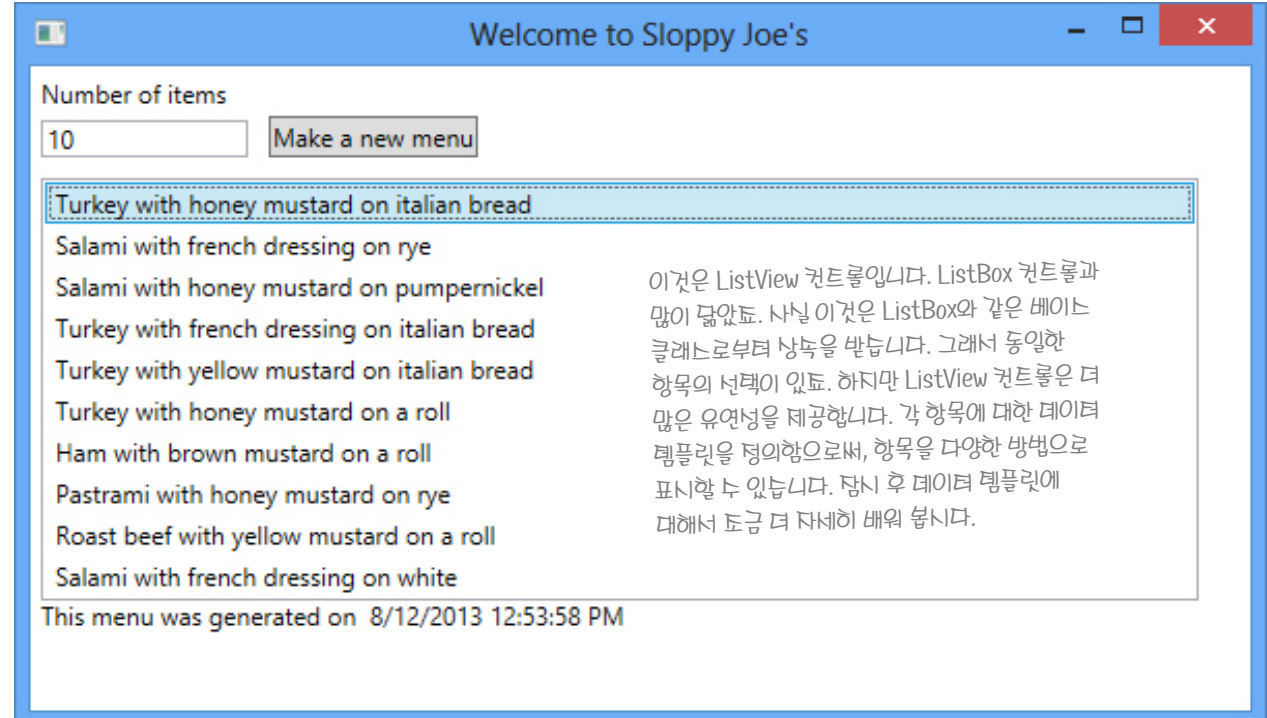
항목을 구성하는 이 세 문자열은 생성자에서 설정합니다. 그리고 읽기 전용의 자동 속성입니다.

ToString() 메서드를 오버라이드했네요. MenuItem 자체에서 문자열을 보여 줍니다.

4 XAML 윈도우를 만들어 봅시다.

여기에 스크린 샷이 있네요. StackPanel을 이용해서 아래처럼 만들 수 있나요? TextBox의 너비는 100입니다. 아래쪽의 TextBlock은 BodyTextBlockStyle의 스타일과 두 개의 <Run> 태그가 있습니다(두 번째 부분은 날짜를 표시합니다).

이번에는 더미 데이터를 넣지 마세요. 데이터 바인딩을 사용할 겁니다.



이것은 ListView 컨트롤입니다. ListBox 컨트롤과 많이 닮았죠. 사실 이것은 ListBox와 같은 베이스 클래스로부터 상속을 받습니다. 그래서 동일한 항목의 선택이 있죠. 하지만 ListView 컨트롤은 더 많은 유연성을 제공합니다. 각 항목에 대한 데이터 템플릿을 정의함으로써, 항목을 다양한 방법으로 표시할 수 있습니다. 잠시 후 데이터 템플릿에 대해서 조금 더 자세히 배워 봅시다.

XAML 코드를 보기 전에 스크린 샷만으로 이 창을 만들 수 있나요?

5 객체 이름과 데이터 바인딩을 XAML에 추가합니다.

MainWindow.xaml에 추가할 것들이 여기에 있군요. XAML에서 시작 태그 <Grid>와 닫는 태그 </Grid>를 StackPanel로 바꿔주세요. 그리고 버튼의 이름을 newMenu로 합니다. ListView, TextBlock, TextBox에서 데이터 바인딩을 사용합니다. 데이터 바인딩을 사용하는 컨트롤의 이름을 설정할 필요가 없습니다(심지어는 버튼에도 이름을 설정할 필요가 없죠. IDE에서 버튼을 더블-클릭했을 때, newMenu_Click이라는 이름의 이벤트 핸들러를 자동으로 추가해 줍니다).

```
<StackPanel Margin="5" x:Name="pageLayoutStackPanel">
  <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
    <StackPanel Margin="0,0,10,0">
      <TextBlock Text="Number of items" Margin="0,0,0,5" />
      <TextBox Width="100" HorizontalAlignment="Left"
        Text="{Binding NumberOfItems, Mode=TwoWay}" />
    </StackPanel>
    <Button x:Name="newMenu" VerticalAlignment="Bottom"
      Click="newMenu_Click" Content="Make a new menu"/>
  </StackPanel>
  <ListView ItemsSource="{Binding Menu}" Margin="0,0,20,0" />
  <TextBlock>
    <Run Text="This menu was generated on " />
    <Run Text="{Binding GeneratedDate}" />
  </TextBlock>
</StackPanel>
```

ListView 컨트롤입니다. ListBox로 바꿔서 형이 어떻게 바뀌는지 살펴보세요.

TextBox에서 항목 수를 설정하고 얻기 위해서(get/set), 양방향 데이터 바인딩을 사용합니다.

이 <Run> 태그가 편리할 것 보여주세요. 텍스트가 바인딩 되는 부분에서만 하나의 TextBlock을 설정할 수 있습니다.

6 MainWindow.xaml.cs에 윈도우의 코드-비하인드를 추가합니다.

윈도우 생성자는 메뉴 컬렉션과 MenuMaker의 인스턴스를 생성하고, 데이터 바인딩을 사용하기 위한 컨트롤의 데이터 컨텍스트를 설정해 줍니다. menuMaker라는 MenuMaker 필드가 있습니다.

```
MenuMaker menuMaker = new MenuMaker();
public MainPage() {
  this.InitializeComponent();
  pageLayoutStackPanel.DataContext = menuMaker;
}
```

MainWindow.xaml.cs의 메인 윈도우 클래스는 MenuMaker 필드를 가져옵니다. 이것은 바인딩된 컨트롤을 모두 포함하는 StackPanel의 데이터 컨텍스트로 사용되죠.

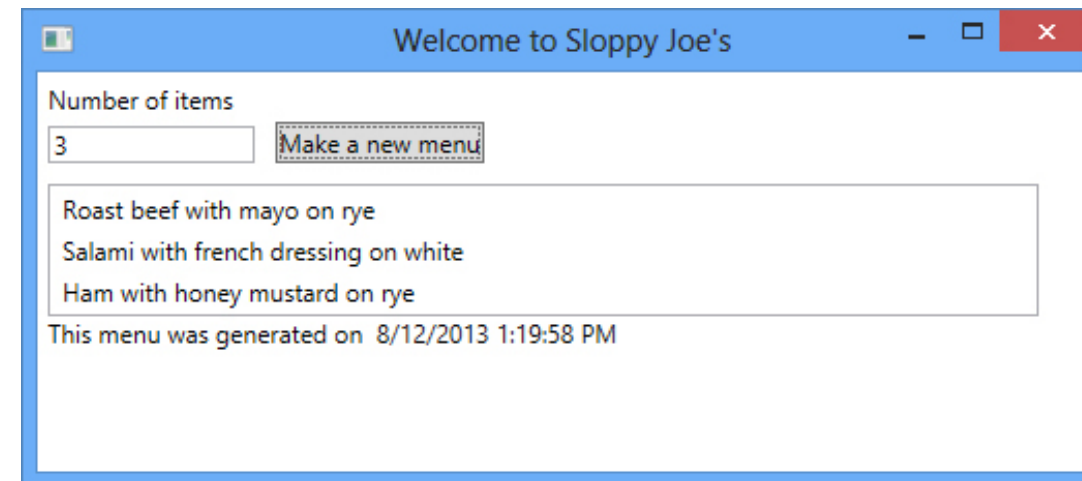
StackPanel의 바깥쪽에서 데이터 컨텍스트를 선언해야 합니다. 그래야 모든 컨트롤이 포함된 데이터 컨텍스트를 전달할 수 있기 때문이죠.

마지막으로 Click 이벤트 핸들러 메서드 스텝을 생성하기 위해서, 버튼을 더블-클릭합니다. 메뉴를 갱신하는 코드는 다음과 같습니다.

```
private void newMenu_Click(object sender, RoutedEventArgs e) {
  menuMaker.UpdateMenu();
}
```

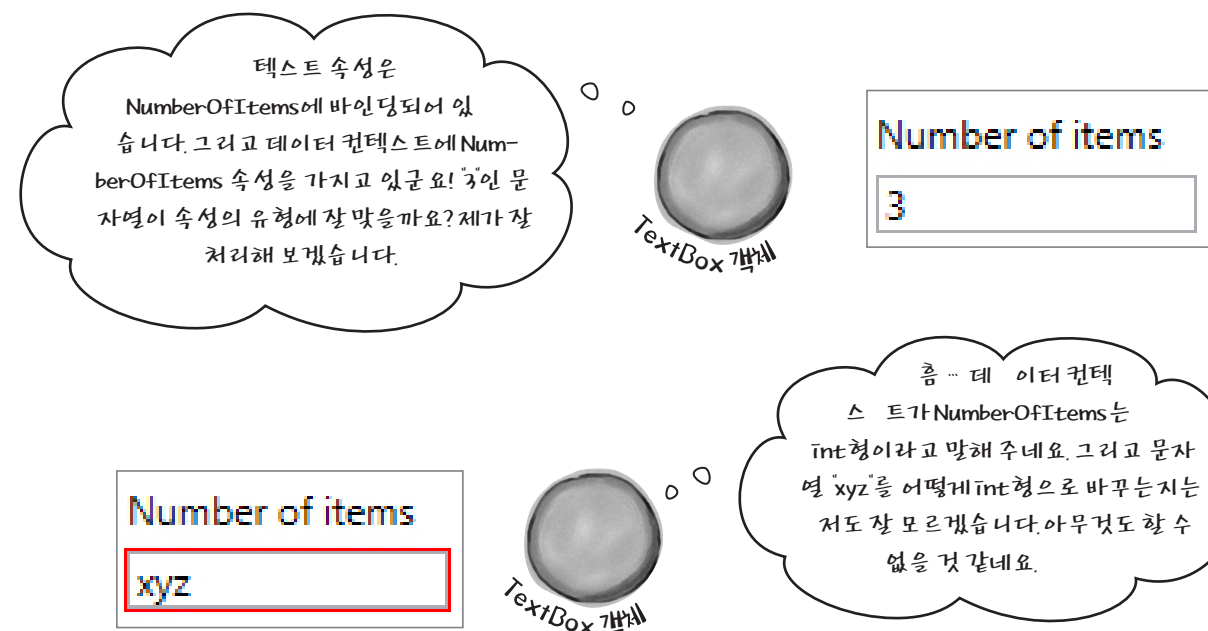
XAML과 C# 코드를 동시에 갱신할 수 있도록 이벤트 핸들러의 이름을 바꿀 수 있는 쉬운 방법이 있습니다. 부록의 “남은 것들 #8”에 가서, IDE의 리팩터링 도구에 대해서 자세한 내용을 살펴보세요.

이제 프로그램을 실행해 봅시다. TextBox의 값을 3으로 설정하고, 아래의 3가지 항목이 있는 메뉴를 생성해 보세요.



바인딩이 얼마나 유연한지 한번 살펴봅시다. TextBox에 “xyz” 혹은 데이터를 입력하지 않고 엔터키를 쳐보세요. 아무 일도 일어나지 않습니다. TextBox가 문자열 처리를 똑똑하게 하네요. 바인딩 경로가 NumberOfItems인지 알고 있습니다. 이 이름의 속성을 가진 데이터 컨텍스트를 찾아서, 문자열을 속성의 유형에 맞게 변환해 줍니다.

아래 생성되는 날짜를 자세히 살펴보세요. 메뉴와 같이 갱신되지 않습니다. 아직 뭔가 해야 할 일이 남았군요.



정적 리소스로 XAML의 객체를 선언해 봅시다

XAML로 윈도우를 만들 때, StackPanel, Grid, TextBlock, Button과 같은 객체들과 객체 그래프를 만듭니다. <TextBox> 태그를 XAML에 추가하게 되면, 윈도우 객체는 TextBox의 인스턴스에 대한 참조를 가진 TextBox 필드를 갖게 됩니다. 그리고 x:Name 속성을 이용해서 TextBox에 이름을 부여해 주죠. 코드-비하인드의 C# 코드가 TextBox에 접근하기 위해 이 이름을 사용할 수 있습니다.

XAML에 정적 리소스(Static Resource)를 추가함으로써 클래스의 인스턴스를 생성하고, 윈도우의 필드에 저장할 수 있습니다. 그리고 데이터 바인딩은 정적 리소스와 잘 맞습니다. 특히 디자이너에서 말이죠. 슬라피 조의 프로그램으로 돌아가서 MenuMaker를 정적 리소스로 바꿔봅시다.

1 코드-비하인드에서 MenuMaker 필드를 삭제합니다.

XAML에서 MenuMaker 클래스와 XAML 컨텍스트 데이터를 설정하는 C# 코드를 지웁니다.

```
MenuMaker menuMaker = new MenuMaker();
```

```
public MainWindow() {
    this.InitializeComponent();

pageLayoutStackPanel.DataContext = menuMaker;
}
```

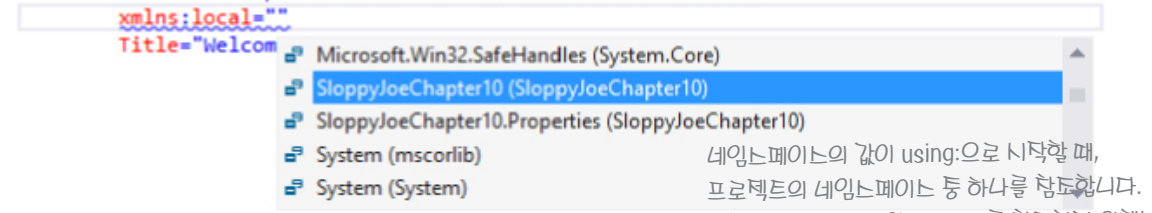
2 프로젝트의 네임스페이스를 XAML에 추가합니다.

윈도우의 XAML 코드 상단을 보세요. xmlns 속성을 가진 윈도우의 시작 태그를 볼 수 있습니다. 이 속성들은 네임스페이스로 정의되어 있죠.

```
<Window x:Class="SloppyJoeChapter10.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Welcome to Sloppy Joe's" Height="350" Width="525">
```

새로운 xmlns 속성을 추가해 봅시다.

```
<Window x:Class="SloppyJoeChapter10.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local=""
        Title="Welcom
```



인텔리센스 창에서 항목을 선택하면, 아래와 같이 추가됩니다.

```
xmlns:local="using:SloppyJoeChapter10"
```

이것은 XML의 네임스페이스 특성입니다. "xmlns:"로 되어 있고, 뒤에 식별자가 있습니다. 여기서 "local"을 사용하네요.

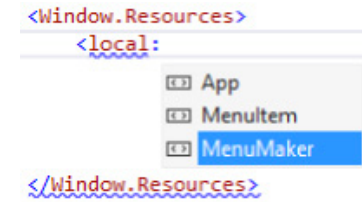
프로젝트의 네임스페이스에서 객체를 생성하기 위한 식별자입니다.

앱의 이름은 SloppyJoeChapter10입니다. 그래서 네임스페이스가 이렇게 생성되었습니다. 앱에 해당하는 네임스페이스를 찾아보세요. 그곳에 MenuMaker가 있습니다.

XAML로 윈도우에 정적 리소스를 추가할 때, 이를 FindResource() 메서드로 접근할 수 있습니다.

3 XAML에 정적 리소스를 추가하고 데이터 컨텍스트를 설정하세요.

<Window.Resources> 태그를 XAML 위쪽에 추가합니다. 그러면 </Window.Resources> 닫는 태그가 추가되죠. 그 사이에 <local:을 입력하면 인텔리센스 창이 뜹니다.



클래스 생성자의 매개변수가 없을 때만 정적 리소스를 추가할 수 있습니다. 생성자에 매개변수가 있다면, XAML 페이지에 어떤 인수를 생성자에 넘기죠?

이 창은 우리가 사용하는 네임스페이스의 모든 클래스들을 보여 줍니다. MenuMaker를 선택하고, 리소스 키의 x:Key에 menuMaker를 입력합니다.

```
<local:MenuMaker x:Key="menuMaker"/>
```

이제 윈도우는 menuMaker라 불리는 정적 MenuMaker 리소스를 갖고 있습니다.

4 StackPanel과 StackPanel의 모든 자식에 대한 DataContext를 설정합니다.

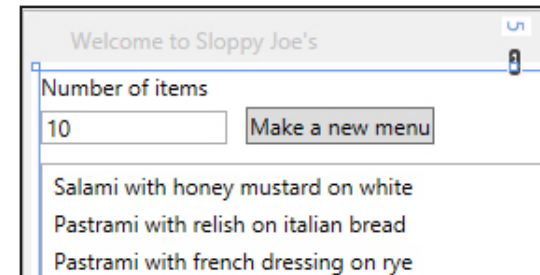
바깥쪽의 StackPanel에서 DataContext 속성을 설정합니다.

```
<StackPanel Margin="5"
            DataContext="{StaticResource ResourceKey=menuMaker}">
```

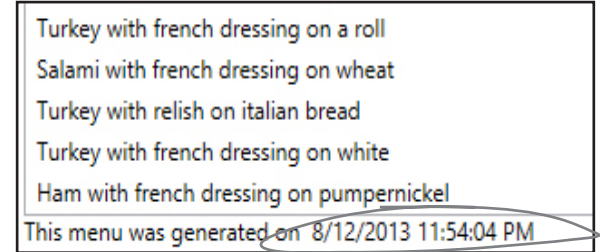
마지막으로, 버튼의 Click 이벤트 핸들러를 수정합니다. 정적 리소스를 찾아서, 메뉴를 갱신해 줍니다.

```
private void newMenu_Click(object sender, RoutedEventArgs e) {
    MenuMaker menuMaker = FindResource("menuMaker") as MenuMaker;
    menuMaker.UpdateMenu();
}
```

프로그램은 여전히 이전처럼 동작합니다. 그리고 XAML에 데이터 컨텍스트를 추가했을 때, IDE에서 무슨 일이 일어났는지 알아챘나요? 데이터 컨텍스트가 추가되는 순간, IDE는 MenuMaker의 인스턴스를 생성하고, 바인딩된 모든 컨트롤을 채우기 위해 속성을 사용합니다. 프로그램을 실행하기 전에도 이미 메뉴가 생성되어 디자이너에서 바로 볼 수 있네요. 놀랍군요!



프로그램을 실행하기 전에도 메뉴는 디자이너에서 바로 볼 수 있습니다.



뭔가 이상하군요. 항목 수를 입력했는데 날짜가 생성되지 않네요. 뭐가 잘못된 거죠?

데이터 템플릿으로 객체를 표현하기

리스트에서 항목을 표시할 때, ObservableCollection에 있는 객체에 바인딩되어 있는 ListView에 대한 ListViewItem, ListBoxItem 혹은 ComboBoxItem 컨트롤의 내용을 보여 줍니다. 그리고 슬라이더 조의 메뉴 목록의 각 ListViewItem 객체는 Menu 컬렉션에 있는 MenuItem 객체에 바인딩됩니다. ListViewItem 객체는 기본적으로 MenuMaker 객체의 ToString() 메서드를 호출합니다. 또 다른 방법으로 데이터 바인딩을 사용하는 데이터 템플릿(data template)을 이용하여, 바인딩된 객체의 속성으로부터 데이터를 보여 줄 수 있습니다.

〈ListView〉 태그를 수정하여 기본 데이터 템플릿을 추가합니다. 기본 {Binding}를 이용하여 항목의 ToString() 메서드를 호출합니다.

```

<ListView ItemsSource="{Binding Menu}" Margin="0,0,20,0">
  <ListView.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding}" />
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>

```

이것은 기본 데이터 템플릿입니다. ListViewItem 을 표시하는 데 사용됩니다.

ListView 태그를 그대로 남겨 두고, /> 대신 >로 바꾼 뒤, 아래에 달는 </ListView> 태그를 추가할 수 있습니다. 그리고 ListView.ItemTemplate 태그를 추가해서 데이터 템플릿을 사용합니다.

경로 없이 {Binding}을 추가하면 바인딩된 객체의 ToString() 메서드를 호출합니다.

데이터 템플릿을 변경해서 메뉴에 색깔을 입혀 봅시다.

```

<DataTemplate>
  <TextBlock>
    <Run Text="{Binding Meat}" Foreground="Blue"/><Run Text=" on "/>
    <Run Text="{Binding Bread}" FontWeight="Light"/><Run Text=" with "/>
    <Run Text="{Binding Condiment}" Foreground="Red" FontWeight="ExtraBold"/>
  </TextBlock>
</DataTemplate>

```

각 Run 태그를 바인딩할 수 있습니다. 각 태그의 색깔과 글씨체, 다른 속성들도 바꿀 수 있죠.

이런 추가! 데이터 템플릿은 여러분이 원하는 어떤 컨트롤들도 포함할 수 있습니다.

```

<DataTemplate>
  <StackPanel Orientation="Horizontal">
    <StackPanel>
      <TextBlock Text="{Binding Bread}" />
      <TextBlock Text="{Binding Bread}" />
      <TextBlock Text="{Binding Bread}" />
    </StackPanel>
    <Ellipse Fill="DarkSlateBlue" Height="Auto" Width="10" Margin="10,0"/>
    <Button Content="{Binding Condiment}" FontFamily="Segoe Script" />
  </StackPanel>
</DataTemplate>

```

DataTemplate 객체의 Content 속성은 하나의 객체에서만 사용할 수 있습니다. 그래서 데이터 템플릿을 다수의 컨트롤에 적용시키고 싶다면, StackPanel과 같은 컨테이너를 사용하면 됩니다.

바보 같은 질문이란 없습니다

Q: 이제 창 레이아웃을 StackPanel이나 Grid를 사용할 수 있습니다. XAML 정적 리소스나 코드-비하인드에서 필드를 사용할 수도 있죠. 컨트롤의 속성을 설정하거나 데이터 바인딩도요. 같은 일을 처리하는 방법이 왜 이렇게 많은 거죠?

Q: 바인딩 경로가 꼭 문자열 속성이어야 하나?

A: 앱을 만드는 데 C#과 XAML은 굉장히 유연한 도구입니다. 이 유연함은 다양한 많은 장치들과 디스플레이에서 매우 세부적인 창을 디자인할 수 있게 해 주죠. 이것은 창의 목적 맞게 사용될 수 있도록 여러분에게 매우 큰 도구 상자를 제공합니다. 이러한 다양한 옵션에 혼동되지 마세요. 다양한 옵션을 살펴보고, 목적에 맞게 잘 선택하면 됩니다.

A: 아니요. 모든 유형의 속성으로 바인딩할 수 있습니다. 만약 속성의 유형과 경로의 원본이 서로 변환될 수 있으면 바인딩이 동작합니다. 그렇지 않다면 바인딩이 무시되죠. 그리고 모든 컨트롤의 속성들이 텍스트로만 되어 있지는 않습니다. EnableMyObject 라 불리는 데이터 컨텍스트에서 bool 값을 얻었다고 가정해 봅시다. 아래의 IsEnabled처럼 어떤 Boolean 속성으로든 이것을 바인딩할 수 있습니다. EnableMyObject 속성값에 따라 컨트롤을 활성화하거나(enable), 비활성화(disable) 상태로 만들어줍니다.

Q: 어떻게 돌아가는지 이해가 잘 안 되네요. 〈Window.Resources〉에 태그를 추가하면 어떤 일이 일어나는 거죠?

IsEnabled="{Binding EnableMyObject}"

A: 태그를 추가할 때, Window 객체가 갱신되고, 정적 리소스를 추가합니다. 이 경우에는 MenuMaker의 인스턴스를 생성하고, Window 객체 리소스를 추가합니다. Window 객체는 Resources 라 불리는 디렉터리를 갖고 있습니다. 만약 이 태그를 추가한 뒤, 디버거를 이용해서 Window 객체를 살펴본다면, MenuMaker의 인스턴스를 포함하는 Resources를 찾을 수 있습니다. 리소스를 선언할 때, x:Key를 사용하여 리소스에 키를 할당합니다.

물론, 텍스트 속성으로 바인딩해서 그냥 True 혹은 False를 출력할 수 있습니다(이것에 대해서 생각해 봤다면, 잘 이해한 겁니다).

Q: 정적 리소스를 추가하고, XAML에서 데이터 컨텍스트를 설정했을 때, C#에서 설정했을 때와는 달리 왜 IDE는 데이터를 폼에 표시한 걸까요?

A: IDE는 창을 렌더링 할 때 필요한 객체를 생성하는 데 모든 정보들을 알고 있기 때문이죠. 여러분이 XAML 코드에 MenuMaker 리소스를 추가하자마자, IDE가 MenuMaker의 인스턴스를 생성합니다. 하지만 생성자 안에서 new문으로 이렇게 할 수 없을 겁니다. 왜냐하면 많은 다른 선언문들이 생성자에 있을 수 있고, 이 선언문들이 실행되어야 하기 때문이죠. 프로그램이 실행될 때 IDE는 단지 코드-비하인드의 C# 코드만 실행합니다. 그러나 만약 페이지에 정적 리소스를 추가한다면, IDE는 TextBlock, StackPanel 등 페이지에 있는 다른 컨트롤의 인스턴스를 생성하는 것처럼 인스턴스를 생성합니다. 이는 디자이너에 나타낼 컨트롤의 속성들을 설정하죠. 그래서 데이터 컨텍스트와 바인딩 경로를 설정했을 때, 이들 또한 메뉴의 항목에 반영되어, IDE의 디자이너에 나타난 거죠.

Q: Key를 이용하여 MenuMaker 리소스 키를 설정했습니다. 또, x:Name을 이용하여 컨트롤에 이름을 붙였습니다. 뭐가 다른 거죠? MenuMaker 객체를 찾기 위해서 왜 FindResource() 메서드를 이용한 거죠? 그리고 키 대신 이름으로 찾으면 안 되나요?

A: WPF 창에서 컨트롤을 추가할 때, XAML에 의해서 생성된 Window 객체에 해당 필드가 추가됩니다. x:Name 속성을 사용하면, 코드에 이름을 붙여 줍니다. 만약 이름을 설정하지 않으면, 컨트롤 객체는 Window 객체 그래프의 일부로써 여전히 생성됩니다. 이름을 설정한다면, XAML 객체의 해당 컨트롤의 참조 필드에 이름을 붙여 줍니다. 버튼의 이벤트 핸들러에 중단점을 놓고, 조사식 창에서 newMenu를 추가해서 확인할 수 있습니다. System.Windows.Controls를 참조합니다. 그리고 Content 속성이 있는 Button 객체는 "Make a new menu"로 설정됩니다.

리소스는 다르게 처리됩니다. Window 객체의 디렉터리에 추가되죠. FindResource() 메서드는 x:Key에서 설정한 키를 사용합니다. 똑같이 중단점을 놓고, 조사식 창에 this.Resources["menuMaker"]를 추가해 보세요. 이번에는 MenuMaker 객체의 참조를 보게 될 겁니다. 리소스 디렉터리에서 해당 참조를 키로 찾기 때문이죠.

“정적 리소스(static resource)”는 이름이 조금 잘못됐습니다. 각 인스턴스마다 정적 리소스들이 생성되기 때문이죠. 또 그렇다고 해서 static 필드는 아닙니다.



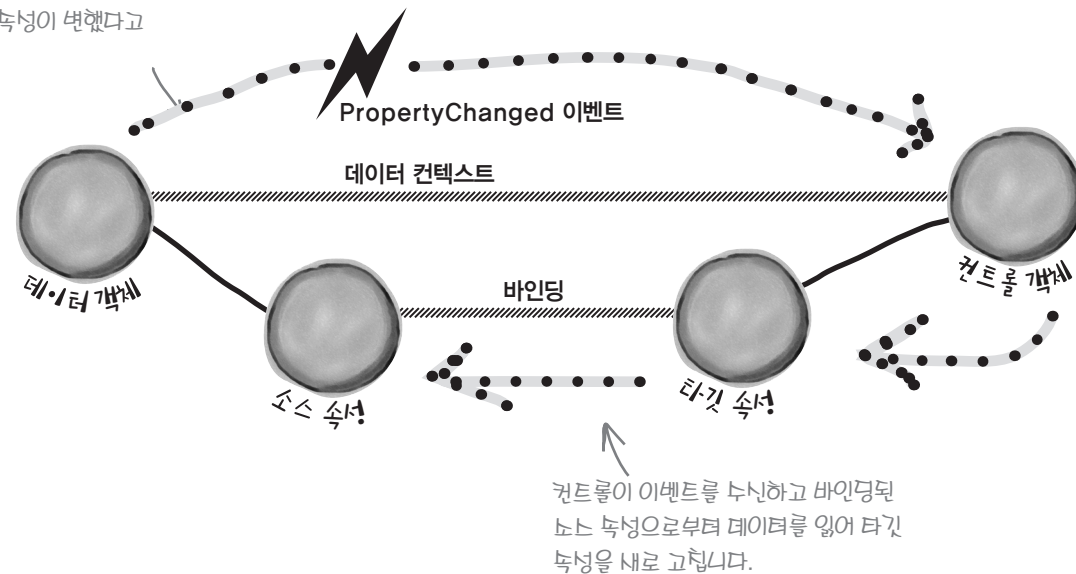
참이맨처음 불러질 때, 윈도우의 정적 리소스들이 인스턴스화됩니다. 그리고 이 정적 리소스들은 객체에 의해 언제든지 사용될 수 있습니다.

INotifyPropertyChanged는 바인딩된 객체를 갱신해 줍니다

MenuMaker 클래스가 메뉴를 수정할 때, 바인딩된 ListView가 갱신됩니다. 하지만 MenuMaker는 동시에 GeneratedDate 속성을 갱신하지 못했죠. 바인딩된 TextBlock 컨트롤이 왜 이것을 갱신하지 못했을까요? 그 이유는 ObservableCollection이 바뀔 때마다 이벤트를 발생시켜 바인딩된 컨트롤에게 데이터가 변했다고 알려주기 때문입니다. 이는 Button 컨트롤에서 클릭했을 때, Click 이벤트를 발생시키거나 Timer가 특정 시간이 경과되었을 때, Tick 이벤트를 발생시킨 것과 같습니다. 즉, ObservableCollection에서 항목이 추가되거나 삭제될 때, 이벤트가 발생합니다.

데이터 객체는 타깃 속성과 바인딩된 컨트롤에게 데이터가 변했는지 알려줄 수 있습니다. PropertyChanged라는 단일 이벤트가 포함된 INotifyPropertyChanged 인터페이스를 구현해 주지만 하면 됩니다. 속성이 변경될 때마다 이벤트를 발생시켜 바인딩된 컨트롤을 자동으로 갱신해줍니다.

데이터 객체는 PropertyChanged 이벤트를 발생시킵니다. 바인딩된 컨트롤에게 속성이 변했다고 알려줍니다.



조심하세요

컬렉션은 데이터 객체와 거의 비슷한 방식으로 작동합니다.

ObservableCollection<T> 객체는 실제로 INotifyPropertyChanged를 구현하지 않습니다. 대신, 밀접하게 연관된 INotifyCollectionChanged라는 인터페이스를 구현합니다. 이 인터페이스는 PropertyChanged 이벤트 대신 CollectionChanged 이벤트를 발생시킵니다. 컨트롤은 이 이벤트가 언제 수신되었는지 알고 있습니다. 왜냐하면 ObservableCollection은 INotifyCollectionChanged 인터페이스를 구현하기 때문이죠. ListView의 데이터 컨텍스트를 INotifyCollectionChanged 객체로 설정하여 이러한 이벤트에게 응답하게 됩니다.

GeneratedDate 속성이 바뀔 때, 알림을 받아서 MenuMaker를 수정해 봅시다

INotifyPropertyChanged는 System.ComponentModel 네임스페이스에 있어서, MenuMaker 클래스 파일의 위쪽에 using문을 추가합니다.

```
using System.ComponentModel;
```

MenuMaker 클래스에 INotifyPropertyChanged를 구현하기 위해서, IDE를 이용해 자동으로 인터페이스를 구현합니다.

```
class MenuMaker : INotifyPropertyChanged
{
    Implement interface 'INotifyPropertyChanged'
    Explicitly implement interface 'INotifyPropertyChanged'
```

7, 8장에서 본 것과 조금 다를 수 있습니다. 어떠한 메서드나 속성이 추가되지 않기 때문이죠. 대신 이벤트를 추가합니다.

```
public event PropertyChangedEventHandler PropertyChanged;
```

그리고 PropertyChanged 이벤트를 발생시킬 OnPropertyChanged() 메서드를 추가합니다.

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

← 이것은 이벤트를 발생시키는 표준 .NET 패턴입니다.

이제 바인딩된 컨트롤에게 속성이 변했는지 알려주기 위해서 속성의 이름을 인수로 한 OnPropertyChanged()을 호출합니다. 그리고 매번 메뉴를 갱신할 때마다 GeneratedDate로 바인딩된 TextBlock을 갱신하기 위해서, 마지막에 한 줄의 UpdateMenu()를 추가합니다.

```
public void UpdateMenu() {
    Menu.Clear();
    for (int i = 0; i < NumberOfItems; i++) {
        Menu.Add(CreateMenuItem());
    }
    GeneratedDate = DateTime.Now;

    OnPropertyChanged("GeneratedDate");
}
```

이제 메뉴를 생성할 때, 날짜가 바뀌는군요.



이렇게 이벤트가 발생하는 것은 이번이 처음이네요.

1장에서부터 이벤트 핸들러 메서드를 사용했었죠. 하지만 이벤트를 발생시킨 건 이번이 처음이네요. 15장에서 이벤트의 동작 방식에 대해서 배울 거예요. 지금은 그냥 인터페이스에 이벤트가 포함될 수 있다는 것만 알아두세요. OnPropertyChanged() 메서드는 다른 객체로 이벤트를 발생시킬 때, 표준 C# 패턴을 따릅니다.



조심하세요

INotifyPropertyChanged를 구현하는 것을 잊지 마세요.

컨트롤이 이 인터페이스를 구현해야만 데이터 바인딩이 작동합니다. 만약 클래스 선언부에서 : INotifyPropertyChanged를 빼먹는다면, 바인딩된 컨트롤은 갱신되지 않습니다. 데이터 객체가 PropertyChanged 이벤트를 발생시킬 때요.



고피시 게임을 WPF 응용 프로그램으로 포팅해 봅시다. 데이터 바인딩을 추가하려면 이 장의 앞부분에서처럼 XAML을 수정해야 합니다. 그리고 8장에 있는 고피시 게임의 모든 클래스와 열거형을 복사하고(혹은 저희 웹 사이트에서 내려받으세요), Player와 Game 클래스를 수정해 봅시다.

1 기존의 클래스 파일을 추가하고, 앱에 맞게 네임스페이스를 고쳐 봅시다.

8장의 고피시 게임 코드를 프로젝트에 추가해 봅시다(Values.cs, Suits.cs, Card.cs, Deck.cs, Comparer_by_Suit.cs, CardComparer_byValue.cs, Game.cs, Player.cs). 솔루션 탐색기에서 기존의 항목을 사용할 순 있지만, 새로운 프로젝트에 맞게 각 네임스페이스를 바꿔야 합니다(이 책의 여러 프로젝트에서 해온 것처럼요).

프로젝트를 빌드해 보세요. Game.cs와 Player.cs에 에러가 이렇게 보일 거예요.

```

❌ 1 The type or namespace name 'Forms' does not exist in the namespace 'System.Windows' (are you missing an assembly reference?)
❌ 2 The type or namespace name 'TextBox' could not be found (are you missing a using directive or an assembly reference?)
❌ 3 The type or namespace name 'TextBox' could not be found (are you missing a using directive or an assembly reference?)
    
```

2 원본 클래스와 객체의 모든 참조를 지우고, Game 클래스에 using 라인을 추가합니다.

여기서 원본을 사용하지 않습니다. Game.cs와 Player.cs에 Using System.Windows.Forms;를 지워주세요. 그리고 TextBox라고 적힌 건 모조리 지워주세요. INotifyPropertyChaged와 ObservableCollection<T>를 사용하기 위해서 Game 클래스를 손봐야 합니다. Game.cs의 위쪽에 아래 using 라인을 추가합니다.

```

using System.ComponentModel;
using System.Collections.ObjectModel;
    
```

3 정적 리소스로 사용될 Game 인스턴스를 추가하고, 데이터 컨텍스트를 설정합니다.

XAML을 수정하여 정적 리소스로 사용될 Game 인스턴스를 추가합니다. 이것을 앞에서 만든 고피시 페이지를 포함하는 그리드를 위한 데이터 컨텍스트로 사용합니다. 정적 리소스의 XAML 코드는 <local:Game x:Name="game" />입니다. 그리고 새로운 생성자가 필요한데, 매개변수가 없는 생성자를 가진 리소스만 포함될 수 있죠.

```

public Game() {
    PlayerName = "Ed";
    Hand = new ObservableCollection<string>();
    ResetGame();
}
    
```

XAML 위에 꼭 <Window.Resources> 태그를 추가해야 해요. 그리고 566, 567쪽에서 했던 것처럼 xmlns:local도 꼭 필요합니다!

4 데이터 바인딩을 위해서 Game 클래스에 public 속성을 추가합니다.

윈도우의 컨트롤 속성을 바인딩 할 속성이 아래에 있습니다.

```

public bool GameInProgress { get; private set; }
public bool GameNotStarted { get { return !GameInProgress; } }
public string PlayerName { get; set; }
public ObservableCollection<string> Hand { get; private set; }
public string Books { get { return DescribeBooks(); } }
public string GameProgress { get; private set; }
    
```

5 바인딩으로 TextBox, ListBox, Button을 활성화/비활성(enable/disable) 상태를 설정합니다.

게임을 시작하지 않았을 때만 "Your Name"의 TextBox와 "Start the Game"의 Button을 사용할 수 있도록 해 봅시다. 그리고 게임을 시작할 때만 "Your hand"의 ListBox와 "Ask for a card" Button을 사용할 수 있도록 합니다. Game 클래스에 GameInProgress 속성을 추가할 거예요. 어떻게 돌아가는지 살펴보고, 속성을 아래와 같은 추가하여 TextBox와 ListBox, 두 개의 Button에 바인딩해 봅시다.

이렇게 각각 두 개씩 필요하군요.

```

{
    IsEnabled="{Binding GameInProgress}"    IsEnabled="{Binding GameNotStarted}"
    IsEnabled="{Binding GameInProgress}"    IsEnabled="{Binding GameNotStarted}"
}
    
```

6 게임의 진행 상황을 Game 클래스에 알려주도록 Play 클래스를 수정합니다.

원본 버전인 Player 클래스의 생성자는 매개변수를 TextBox로 했습니다. 이를 Game 클래스의 참조로 바꿔주고, private 필드에 담아 주세요(어떻게 이 새로운 생성자가 플레이어를 추가할 때 사용되는지 아래의 StartGame() 메서드를 살펴보세요). 그리고 TextBox 참조를 사용하는 줄을 찾아서 Game 객체의 AddProgress() 메서드를 호출하는 걸로 바꿔주세요.

7 Game 클래스를 수정합니다.

bool 대신 void를 반환하도록 PlayOneRound() 메서드를 수정합니다. 그리고 게임의 진행 상황을 알리기 위해서, TextBox 대신 AddProgress() 메서드를 추가합니다. 만약 플레이어가 이겼을 때는 게임을 리셋하고, 처음으로 돌아가게 합니다. 패배했다면 Hand 컬렉션을 초기화한 후, 플레이어에게 카드를 나눠줍니다. 4개의 메서드를 추가/수정해야 합니다. 이 메서드들이 어떻게 동작하는지 살펴보세요.

```

public void StartGame() {
    ClearProgress();
    GameInProgress = true;
    OnPropertyChanged("GameInProgress");
    OnPropertyChanged("GameNotStarted");
    Random random = new Random();
    players = new List<Player>();
    players.Add(new Player(PlayerName, random, this));
    players.Add(new Player("Bob", random, this));
    players.Add(new Player("Joe", random, this));
    Deal();
    players[0].SortHand();
    Hand.Clear();
    foreach (String cardName in GetPlayerCardNames())
        Hand.Add(cardName);
    if (!GameInProgress)
        AddProgress(DescribePlayerHands());
    OnPropertyChanged("Books");
}

public void AddProgress(string progress) {
    GameProgress = progress +
        Environment.NewLine +
        GameProgress;
    OnPropertyChanged("GameProgress");
}

public void ClearProgress() {
    GameProgress = String.Empty;
    OnPropertyChanged("GameProgress");
}

public void ResetGame() {
    GameInProgress = false;
    OnPropertyChanged("GameInProgress");
    OnPropertyChanged("GameNotStarted");
    books = new Dictionary<Values, Player>();
    stock = new Deck();
    Hand.Clear();
}
    
```

그리고 속성을 갱신하기 위해서 INotifyPropertyChanged 인터페이스를 구현하고, MenuMaker 클래스와 같은 OnPropertyChanged() 메서드를 추가합니다.



```
Game game;
public MainWindow() {
    InitializeComponent();
    game = this.FindResource("game") as Game;
}
private void startButton_Click(object sender, RoutedEventArgs e) {
    game.StartGame();
}
private void askForACard_Click(object sender, RoutedEventArgs e) {
    if (cards.SelectedIndex >= 0)
        game.PlayOneRound(cards.SelectedIndex);
}
private void cards_MouseDoubleClick(object sender, MouseButtonEventArgs e) {
    if (cards.SelectedIndex >= 0)
        game.PlayOneRound(cards.SelectedIndex);
}
```

여기에 모든 코드-비하인드가 있군요.

Player 클래스가 변경된 부분입니다.

```
class Player {
    private string name;
    public string Name { get { return name; } }
    private Random random;
    private Deck cards;
    private Game game;
    public Player(String name, Random random, Game game) {
        this.name = name;
        this.random = random;
        this.game = game;
        this.cards = new Deck(new Card[] { });
        game.AddProgress(name + " has just joined the game");
    }
    public Deck DoYouHaveAny(Values value)
    {
        Deck cardsIHave = cards.PullOutValues(value);
        game.AddProgress(Name + " has " + cardsIHave.Count + " " + Card.Plural(value));
        return cardsIHave;
    }
    public void AskForACard(List<Player> players, int myIndex, Deck stock, Values value) {
        game.AddProgress(Name + " asks if anyone has a " + value);
        int totalCardsGiven = 0;
        for (int i = 0; i < players.Count; i++) {
            if (i != myIndex) {
                Player player = players[i];
                Deck CardsGiven = player.DoYouHaveAny(value);
                totalCardsGiven += CardsGiven.Count;
                while (CardsGiven.Count > 0)
                    cards.Add(CardsGiven.Deal());
            }
        }
        if (totalCardsGiven == 0) {
            game.AddProgress(Name + " must draw from the stock.");
            cards.Add(stock.Deal());
        }
    }
}
```

// 나머지 기존 Player 클래스와 같습니다

XAML에서 변경된 부분입니다.

```
<Grid Margin="10" DataContext="{StaticResource ResourceKey=game}">
    <TextBlock Text="Your Name" />
    <StackPanel Orientation="Horizontal" Grid.Row="1">
        <TextBox x:Name="playerName" FontSize="12" Width="150"
            Text="{Binding PlayerName, Mode=TwoWay}"
            IsEnabled="{Binding GameNotStarted}" />
        <Button x:Name="startButton" Margin="5,0" IsEnabled="{Binding GameNotStarted}"
            Content="Start the game!" Click="startButton_Click" />
    </StackPanel>
    <TextBlock Text="Game progress" Grid.Row="2" Margin="0,10,0,0" />
    <ScrollViewer Grid.Row="3" FontSize="12" Background="White" Foreground="Black"
        Content="{Binding GameProgress}" />
    <TextBlock Text="Books" Margin="0,10,0,0" Grid.Row="4" />
    <ScrollViewer FontSize="12" Background="White" Foreground="Black"
        Grid.Row="5" Grid.RowSpan="2"
        Content="{Binding Books}" />
    <TextBlock Text="Your hand" Grid.Row="0" Grid.Column="2" />
    <ListBox x:Name="cards" Background="White" FontSize="12"
        Height="Auto" Margin="0,0,0,10"
        Grid.Row="1" Grid.RowSpan="5" Grid.Column="2"
        ItemsSource="{Binding Hand}" IsEnabled="{Binding GameInProgress}"
        MouseDoubleClick="cards_MouseDoubleClick" />
    <Button x:Name="askForACard" Content="Ask for a card"
        HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
        Grid.Row="6" Grid.Column="2"
        Click="askForACard_Click" IsEnabled="{Binding GameInProgress}" />
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="5*" />
        <ColumnDefinition Width="40" />
        <ColumnDefinition Width="2*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" MinHeight="150" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
</Grid>
```

그리드의 데이터 컨텍스트는 Game 클래스입니다. 모든 바인딩은 이 클래스의 속성이기 때문이죠.

TextBox는 PlayerName에 양방향 바인딩 되어 있군요.

여기에 시작 버튼의 Click 이벤트

핸들러가 있군요.

게임 진행과 북의 ScrollViewer는

GameProgress와 Books 속성에

바인딩 되었군요.

IsEnabled 속성은 부울 속성으로 컨트롤을 활성화 하거나 비 활성화 합니다. 이 속성을 부울 속성으로 바인딩 합니다. 그래서 속성에 따라 컨트롤을 활성/비활성(on/off) 상태로 만들어 줍니다.



연습문제 정답

이 속성들은 XAML의 데이터 바인딩에 사용되는데요.

이 메서드들은 게임 진행 데이터 바인딩을 동작하게 합니다. 새로운 게임 진행의 라인은 앞부분에 추가됩니다. 이전엔 있던 라인은 ScrollViewer 아래에 있습니다.

문제에서 제공한 StartGame() 메서드입니다. 이 메서드는 게임 진행을 초기화하고, 플레이어를 생성합니다. 카드를 돌리고, 상황과 복을 업데이트합니다.

Game 클래스가 변경된 부분입니다. 문제에서 설명한 코드가 포함되어 있습니다.

```

using System.ComponentModel;
using System.Collections.ObjectModel;

class Game : INotifyPropertyChanged {
    private List<Player> players;
    private Dictionary<Values, Player> books;
    private Deck stock;

    public bool GameInProgress { get; private set; }
    public bool GameNotStarted { get { return !GameInProgress; } }
    public string PlayerName { get; set; }
    public ObservableCollection<string> Hand { get; private set; }
    public string Books { get { return DescribeBooks(); } }
    public string GameProgress { get; private set; }

    public Game() {
        PlayerName = "Ed";
        Hand = new ObservableCollection<string>();
        ResetGame();
    }

    public void AddProgress(string progress) {
        GameProgress = progress + Environment.NewLine + GameProgress;
        OnPropertyChanged("GameProgress");
    }

    public void ClearProgress() {
        GameProgress = String.Empty;
        OnPropertyChanged("GameProgress");
    }

    public void StartGame() {
        ClearProgress();
        GameInProgress = true;
        OnPropertyChanged("GameInProgress");
        OnPropertyChanged("GameNotStarted");

        Random random = new Random();
        players = new List<Player>();
        players.Add(new Player(PlayerName, random, this));
        players.Add(new Player("Bob", random, this));
        players.Add(new Player("Joe", random, this));

        Deal();
        players[0].SortHand();
        Hand.Clear();
        foreach (String cardName in GetPlayerCardNames())
            Hand.Add(cardName);
        if (!GameInProgress)
            AddProgress(DescribePlayerHands());
        OnPropertyChanged("Books");
    }
}

```

INotifyPropertyChanged와 ObservableCollection을 위한 라인입니다.

새로운 Game 클래스의 생성자군요. 게임이 리셋될 때, 단 하나의 컬렉션을 생성하고, 초기화 합니다. 만약 새 컬렉션 객체가 생성되며, 이전 객체에 대한 참조를 잃고, 업데이트가 중단됩니다.

지금까지 이 책에서 만든 모든 프로그램은 XAML을 사용한 WPF 응용 프로그램과 같이 적용하거나 다시 만들 수 있습니다. 만드는 방법은 다양하죠. 특히 XAML을 사용한다면요! 연습문제에서 이러한 많은 코드의 문제를 주는 이유이기도 하죠.

폼이 진행 상황을 업데이트하기 위해서, Boolean 값을 반환했던 부분이네요. 지금은 그냥 AddProgress() 메서드를 호출하고, 데이터 바인딩을 통해 진행 상황을 업데이트해 줍니다.

```

public void PlayOneRound(int selectedPlayerCard) {
    Values cardToAskFor = players[0].Peek(selectedPlayerCard).Value;
    for (int i = 0; i < players.Count; i++) {
        if (i == 0)
            players[0].AskForACard(players, 0, stock, cardToAskFor);
        else
            players[i].AskForACard(players, i, stock);
        if (PullOutBooks(players[i])) {
            AddProgress(players[i].Name + " drew a new hand");
            int card = 1;
            while (card <= 5 && stock.Count > 0) {
                players[i].TakeCard(stock.Deal());
                card++;
            }
        }
        OnPropertyChanged("Books");
        players[0].SortHand();
        if (stock.Count == 0) {
            AddProgress("The stock is out of cards. Game over!");
            AddProgress("The winner is... " + GetWinnerName());
            ResetGame();
            return;
        }
    }
    Hand.Clear();
    foreach (String cardName in GetPlayerCardNames())
        Hand.Add(cardName);
    if (!GameInProgress)
        AddProgress(DescribePlayerHands());
}

public void ResetGame() {
    GameInProgress = false;
    OnPropertyChanged("GameInProgress");
    OnPropertyChanged("GameNotStarted");
    books = new Dictionary<Values, Player>();
    stock = new Deck();
    Hand.Clear();
}

public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName) {
    PropertyChangedEventHandler propertyChangedEvent = PropertyChanged;
    if (propertyChangedEvent != null) {
        propertyChangedEvent(this, new PropertyChangedEventArgs(propertyName));
    }
}

// 나머지 기존 Game 클래스와 같습니다.

```

복이 바뀌면, 폼은 이를 알아차려서 ScrollViewer를 갱신합니다.

PlayOneRound() 메서드가 누락된 부분이군요. 게임이 끝났을 때, 진행 상황을 업데이트해 줍니다. 그렇지 않은 경우, 플레이어의 카드와 복을 업데이트해 두는군요.

문제에서 본 ResetGame() 메서드네요. books, stock, hand를 초기화해 줍니다.

이전에 본 표준 PropertyChanged 이벤트 패턴입니다.

Chapter 11



조심하세요

네임스페이스에 클래스가 존재하지 않는다는 XAML 오류를 만났나요? 모든 C# 코드가 컴파일 되었는지 확인하고, 모든 컨트롤의 이벤트 핸들러가 코드-비하인드에 선언되었는지 확인해 보세요.

심지어 MyWpfApplication 네임스페이스에 MyDataClass라 불리는 클래스를 확실히 선언했다고 하더라도 정적 리소스를 선언하지 않을 때, 이와 같은 오류를 볼 수 있습니다.

X 1 The name "MyDataClass" does not exist in the namespace "clr-namespace:MyWpfApplication".

코드-비하인드 혹은 XAML 컨트롤의 이벤트 핸들러가 없을 경우 자주 발생하는 오류입니다. 약간의 오해의 소지가 있을 수 있습니다. 왜냐하면 다른 곳에서 코드 오류가 발생했는데, 정적 리소스가 선언된 태그에 오류가 발생했다고 알려주기 때문이죠.

여러분이 이 오류를 재현할 수 있습니다. MyWpfApplication이라는 새 WPF 프로젝트를 생성하고, MyDataClass라는 데이터 클래스를 추가한 뒤, <Window.Resources>에 정적 리소스로 이 클래스를 추가합니다. 그리고 창에 버튼을 추가해 주세요. 그리고 나서 버튼의 이벤트 핸들러를 XAML에 추가하기 위해서, Button 태그에 Click="Button_Click"을 추가합니다. 절대로 코드-비하인드에 Button_Click() 메서드를 추가하지 마세요. 코드를 다시 빌드했을 때, 위와 같은 오류를 볼 수 있습니다. 코드-비하인드에 Button_Click() 메서드를 추가하면, 오류가 사라집니다.

↑
 솔루션 탐색기에서 프로젝트를 마우스 오른쪽 클릭하여, "프로젝트 언로드"(Unload Project)를 선택하고, 다시 "프로젝트 다시 로드"(Reload Project)를 선택했을 때, 오류가 조금 더 명확해지는 경우가 있습니다. 여러분에게 더 많은 도움이 되는 다른 오류 메시지가 표시될 수도 있습니다.

책에서 윈도우 스토어 앱에 대해서 많이 다루지만, 부록의 WPF에서도 핵심 내용을 배울 수 있습니다.

윈도우 스토어는 비동기 프로그래밍을 고려해서 만들었습니다. WPF에서도 할 수 있지만, 비동기 프로그래밍에 대한 도구를 많이 제공하지 않습니다.

책 580, 581쪽을 읽어 주세요(9장에서 본 익숙한 파일 클래스가 다 어디로 갔죠?) 글썄요, WPF 앱에서는 문제될 게 없습니다. 여러분이 이미 사용하고 있는 파일 클래스와 직렬화를 계속 사용할 수 있습니다. 하지만 WPF 응용 프로그램은 윈도우 스토어에 대한 .NET 프레임워크와 함께 제공되는 새로운 비동기 파일 및 다이얼로그 클래스를 사용할 수 없습니다.

이 부록에서는 async와 await 키워드, 데이터 컨트랙트 직렬화를 사용해서 두 개의 WPF 프로젝트를 진행합니다. 11장에서는 아래의 방법을 추천합니다.

- ★ 이 부록은 582, 583쪽을 대체합니다.
- ★ 584-589쪽은 윈도우 스토어 앱입니다. 넘겨주세요.
- ★ 책 590, 591쪽에서 데이터 컨트랙트 직렬화에 대한 내용을 읽어 주세요.
- ★ 592-594쪽은 윈도우 스토어 앱입니다. 넘겨 주세요.
- ★ 책 595쪽을 읽어 주세요. 그리고 이 부록에서 책 596-600쪽 "직접 해 보세요!"를 대체하는 프로젝트를 진행합니다.
- ★ 이 책의 나머지 부분은 브라이언의 변명 관리에 대한 윈도우 스토어 앱을 다룹니다. 이 프로젝트의 목표는 윈도우 스토어 앱의 Windows.Storage 네임스페이스에 있는 파일 도구를 배웁니다. 이 클래스들은 윈도우 스토어 앱에 한정되어 있기 때문에, WPF에서 이 프로젝트를 대체할 수 없습니다.



반응적인 await

원폼 프로그램에서 MessageBox.Show()를 호출했을 때 무슨 일이 일어났죠? 모든 것이 중단되고, 프로그램은 대화상자가 사라질 때까지 잠시 멈춰져 있었습니다. 이 프로그램은 별로 반응적이지 못하네요. WPF 앱은 항상 반응적이어야 합니다. 심지어 사용자가 피드백을 기다리는 동안에도 말이죠. 하지만, 대화상자를 기다리거나 모든 파일을 읽고 쓰는 일은 시간이 오래 걸립니다. 어떤 메시지가 수행될 때, 다른 나머지 프로그램들이 이 메시지의 수행이 끝날 때까지 기다리는 것을 **블로킹(Blocking)**이라고 합니다. 이것이 반응적이지 못한 프로그램의 가장 큰 이유이기도 하죠. 윈도우 스토어 앱은 블록 상태에 있는 동안 반응적이지 못한 것을 방지하기 위해서 **await 연산자(operator)**와 **async 제한자(modifier)**를 사용합니다. WPF로 하나의 작업을 정의해서 어떻게 블록 상태가 되는지 그리고 이를 어떻게 비동기적으로 처리하는지 예제를 통해 살펴 봅시다.

비동기적으로 호출될 수 있다는 것을 나타내기 위해서 async 제한자로 메시지를 선언합니다.

```
private async Task LongTaskAsync()
{
    await Task.Delay(5000);
}
```

Task 클래스는 System.Threading.Tasks 네임스페이스에 있습니다. 그리고 Delay() 메서드는 지정한 밀리 초 단위로 메서드 수행을 블록 상태로 만듭니다. 이 메서드는 2장에서 본 Thread.Sleep() 메서드와 매우 비슷합니다. 하지만 await 연산자를 통해 비동기적으로 호출될 수 있도록, async 제한자와 함께 선언되어야 합니다.

await 연산자는 이 코드에서 수행하고 있는 메시지를 잠시 멈춥니다. 그리고 LongTaskAsync() 메서드 수행이 끝날 때까지 기다립니다. 이 메서드는 사용자가 하나의 명령을 내릴 때까지 블록 상태가 됩니다. 그 사이 나머지 프로그램은 **다른 이벤트와 반응을 유지합니다.** LongTaskAsync() 메서드가 반환되자마자 이를 호출한 메시드는 블록된 메시지를 깨워줍니다(그 사이 수행된 다른 모든 이벤트가 끝날 때까지 기다릴 수도 있습니다).

만약 메시지가 await 연산자를 사용한다면, async 제한자를 반드시 선언해야 합니다.

```
private async void countButton_Click(object sender, RoutedEventArgs e) {
    // ... 어떤 코드 ...
    await LongTaskAsync();
    // ... 어떤 코드 ...
}
```

메서드가 async로 선언되었을 때, 이 메시지를 호출하는 몇 가지 옵션이 있습니다. 평소와 같은 방법으로 이 메시지를 호출한다면, await문이 호출되자마자 앱의 블록 상태를 막은 뒤, 그 결과값이 반환됩니다.

새로운 WPF 응용 프로그램을 생성하고 다음과 같이 메인 윈도우 XAML을 추가해 주세요. 어떻게 코드가 동작하는지 볼 수 있습니다.

```
<Window x:Class="WpfAndAsync.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="WPF and async" Height="150" Width="200" ResizeMode="CanResizeWithGrip">
    <Grid>
        <StackPanel>
            <CheckBox x:Name="useAwaitAsync" IsChecked="True"
                Content="Use await/async" Margin="5"/>
            <Button x:Name="countButton" Content="Start counting"
                HorizontalAlignment="Left" Click="countButton_Click" Margin="5"/>
            <TextBlock x:Name="progress" HorizontalAlignment="Left" Margin="5" />
        </StackPanel>
    </Grid>
</Window>
```

코드-비하인드입니다.

```
using System.Threading;
using System.Windows.Threading;

public partial class MainWindow : Window {
    DispatcherTimer timer = new DispatcherTimer();

    public MainWindow() {
        InitializeComponent();

        timer.Tick += timer_Tick;
        timer.Interval = TimeSpan.FromSeconds(.1);
    }

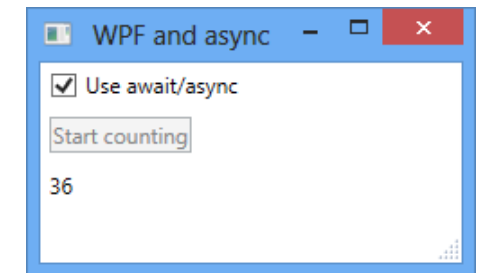
    int i = 0;
    void timer_Tick(object sender, EventArgs e) {
        progress.Text = (i++).ToString();
    }

    private async void countButton_Click(object sender, RoutedEventArgs e) {
        countButton.IsEnabled = false;
        timer.Start();
        if (useAwaitAsync.IsChecked == true)
            await LongTaskAsync();
        else
            LongTask();
        countButton.IsEnabled = true;
    }

    private void LongTask() {
        Thread.Sleep(5000);
        timer.Stop();
    }

    private async Task LongTaskAsync() {
        await Task.Delay(5000);
        timer.Stop();
    }
}
```

프로젝트 이름을 WpfAndAsync로 지었습니다. 만약 프로젝트를 다른 이름으로 설정했다면, 여러분은 다음 부분을 프로젝트의 이름에 맞게 고쳐야 합니다. x:Class="WpfAndAsync.MainWindow"



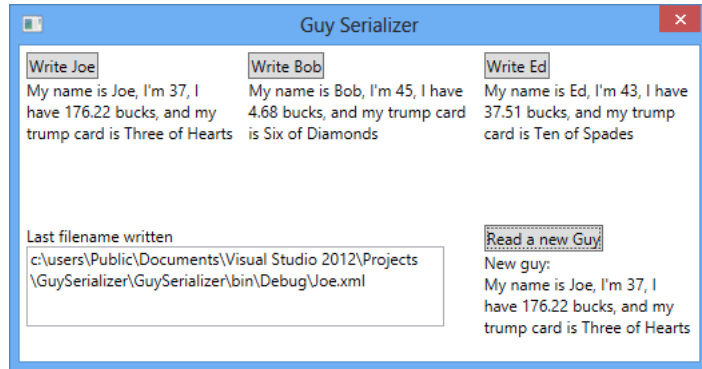
버튼의 이벤트 핸들러는 체크박스의 IsChecked 속성을 사용합니다. 박스가 체크되었다면, 이벤트 핸들러는 비동기 메서드인 await LongTaskAsync()를 호출합니다. 이 메서드는 await 키워드와 함께 호출되어서, 이벤트 핸들러를 잠시 멈추고, 프로그램의 나머지 부분을 계속 실행할 수 있게 해줍니다. 창의 속성을 바꾸거나, 출력 창에 텍스트를 보여 주는 다른 버튼을 추가해 보세요. 타이머가 틱을 하는 동안 이 버튼을 사용합니다. 체크박스가 체크되어 있지 않으면, IsChecked는 false가 됩니다. 그리고 이벤트 핸들러는 LongTask()를 호출해서 블록 상태에 빠지게 되어, 전체 프로그램이 무반응 상태가 됩니다. 추가한 다른 버튼들도 마찬가지로 이벤트에 아무런 반응이 없습니다.

체크박스가 체크된 상태에서 버튼을 클릭하세요. 숫자가 증가하면서, 폼이 반응적이게 됩니다. 버튼이 비활성화되면서, 폼을 움직이거나 크기를 조정할 수 있죠. 만약 체크박스가 체크되지 않은 상태에서 버튼을 누르면, 폼은 무반응 상태에 빠지게 됩니다.

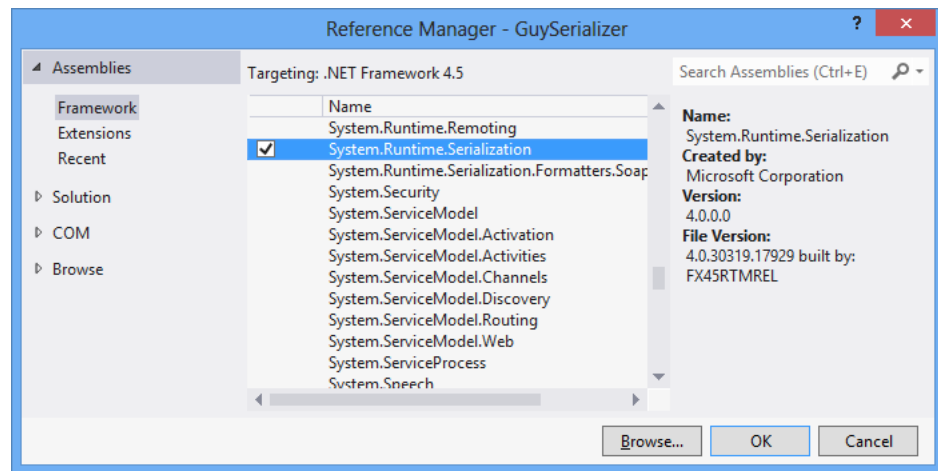
Guy 객체를 파일로 보내주세요

데이터 컨트랙트 직렬화 실습 프로젝트를 해 봅시다. 새 WPF 응용 프로그램을 생성하고, 책 595쪽과 같이 데이터 컨트랙트가 있는 두 클래스를 추가합니다(두 클래스 모두 using System.Runtime.Serialization이 필요합니다). Card 클래스를 위한 Suites와 Values 열거형을 추가해 주세요. 그리고 아래와 같은 창을 만들면 됩니다.

직접 해
봅시다!



- 1 코딩을 시작하기 전에 솔루션 탐색기 > 참조 (마우스 오른쪽 클릭) > 참조 추가 메뉴를 선택합니다. 프레임워크를 선택한 후 System.Runtime.Serialization을 찾아서 체크하세요. 그리고 확인을 클릭합니다.



이것은 WPF 응용 프로그램이 System.Runtime.Serialization 네임스페이스를 사용하겠다는 것을 의미합니다. 과정 2에서 XAML을 추가할 때, <local:GuyManager>의 오류를 지우기 위해서 빈 GuyManager 클래스를 추가할 수 있습니다. 과정 3에서 GuyManager 클래스의 내용을 채웁니다.

- 2 윈도우의 XAML 코드입니다.

```
<Window x:Class="GuySerializer.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="clr-namespace:GuySerializer"
  Title="Guy Serializer" Height="275" Width="525" ResizeMode="NoResize">
  <Window.Resources>
    <local:GuyManager x:Key="guyManager"/>
  </Window.Resources>
  <Grid DataContext="{StaticResource guyManager}" Margin="5">
    <Grid.ColumnDefinitions>
      <ColumnDefinition/>
      <ColumnDefinition/>
      <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="4*" />
      <RowDefinition Height="3*" />
    </Grid.RowDefinitions>
    <StackPanel>
      <Button x:Name="WriteJoe" Content="Write Joe"
        HorizontalAlignment="Left" Click="WriteJoe_Click"/>
      <TextBlock Text="{Binding Joe}" Margin="0,0,10,20" TextWrapping="Wrap"/>
    </StackPanel>
    <StackPanel Grid.Column="1">
      <Button x:Name="WriteBob" Content="Write Bob"
        HorizontalAlignment="Left" Click="WriteBob_Click"/>
      <TextBlock Text="{Binding Bob}" Margin="0,0,0,20" TextWrapping="Wrap"/>
    </StackPanel>
    <StackPanel Grid.Column="2" Margin="10,0,0,0">
      <Button x:Name="WriteEd" Content="Write Ed"
        HorizontalAlignment="Left" Click="WriteEd_Click"/>
      <TextBlock Text="{Binding Ed}" Margin="0,0,0,20" TextWrapping="Wrap"/>
    </StackPanel>
    <StackPanel Grid.Row="1" Grid.ColumnSpan="2" Margin="0,0,20,0">
      <TextBlock>Last filename written</TextBlock>
      <TextBox Text="{Binding GuyFile, Mode=TwoWay}"
        TextWrapping="Wrap" Height="60" Margin="0,0,0,20"/>
    </StackPanel>
    <StackPanel Grid.Row="1" Grid.Column="2" Margin="10,0,0,0">
      <Button x:Name="ReadNewGuy" Content="Read a new Guy"
        HorizontalAlignment="Left" Click="ReadNewGuy_Click" />
      <StackPanel>
        <TextBlock Text="New guy:"/>
        <TextBlock TextWrapping="Wrap" Text="{Binding NewGuy}"/>
      </StackPanel>
    </StackPanel>
  </Grid>
</Window>
```

이 프로젝트의 이름은 "GuySerializer"입니다. 프로젝트에 다른 네임스페이스를 사용한다면, 이들을 여러분의 네임스페이스에 맞게 바꿔주세요.

그리드의 데이터 컨트랙트는 GuyManager의 정적 리소스입니다.

윈도우는 두 개의 행과 세 개의 열이 있습니다.

위쪽 행에 있는 각각의 열은 TextBlock과 Button으로 이루어진 StackPanel이 있습니다.

이 TextBlock은 GuyManager의 Ed 속성에 바인딩되어 있습니다.

맨 아래쪽 행의 첫 번째 셀은 두 열을 차지하고 있네요. 몇몇의 컨트롤이 속성에 바인딩되었군요. 왜 TextBox에 Path(경로)가 설정되었을까요?

아직 안 끝났어요. 페이지를 넘겨주세요!

3 GuyManager 클래스를 추가합니다.

```
using System.ComponentModel;
using System.IO;
using System.Runtime.Serialization;
class GuyManager : INotifyPropertyChanged
{
    private Guy joe = new Guy("Joe", 37, 176.22M);
    public Guy Joe
    {
        get { return joe; }
    }
    private Guy bob = new Guy("Bob", 45, 4.68M);
    public Guy Bob
    {
        get { return bob; }
    }
    private Guy ed = new Guy("Ed", 43, 37.51M);
    public Guy Ed
    {
        get { return ed; }
    }
    public Guy NewGuy { get; set; }
    public string GuyFile { get; set; }

    public void ReadGuy()
    {
        if (String.IsNullOrEmpty(GuyFile))
            return;

        using (Stream inputStream = File.OpenRead(GuyFile))
        {
            DataContractSerializer serializer = new DataContractSerializer(typeof(Guy));
            NewGuy = serializer.ReadObject(inputStream) as Guy;
        }
        OnPropertyChanged("NewGuy");
    }
}
```

이 프로그램에서는 get 접근자만으로 읽기 전용의 속성으로 바인딩된 TextBox를 사용합니다. private set 접근자와 public get 접근자를 가진 속성으로 바인딩을 한다면, 오류가 발생합니다. 이 코드와 같이 백킹 필드를 사용하면 아무런 문제가 없습니다.

3개의 읽기 전용 Guy 속성이 있습니다. private 백킹 필드군요. XAML에서 TextBlock은 이들을 바인딩하고 있습니다.

네 번째 TextBlock은 이 Guy 속성에 바인딩되어 있습니다. 이 속성은 ReadGuy() 메서드에 의해 설정됩니다.

ReadGuy() 메서드는 스트림을 열고, 읽기 위해서 System.IO 메서드를 사용하는 것과 비슷합니다. XML 파일의 데이터를 직렬화하기 위해서, BinaryFormatter를 사용하는 대신 DataContractSerializer를 사용합니다.

```
public void WriteGuy(Guy guyToWrite)
{
    GuyFile = Path.GetFullPath(guyToWrite.Name + ".xml");
    if (File.Exists(GuyFile))
        File.Delete(GuyFile);
    using (Stream outputStream = File.OpenWrite(GuyFile))
    {
        DataContractSerializer serializer = new DataContractSerializer(typeof(Guy));
        serializer.WriteObject(outputStream, guyToWrite);
    }
    OnPropertyChanged("GuyFile");
}

public event PropertyChangedEventHandler PropertyChanged;

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

파일이 이미 존재한다면, 지워진 후, 파일 스트림을 이용해 파일을 다시 만듭니다. 그리고 데이터 컨트랙트 직렬화를 사용합니다.

파일을 쓸 전체 경로를 얻기 위해서, Path 클래스(System.IO 네임스페이스)의 GetFullPath() 메서드를 사용합니다.

이런 장에서 본 코드와 똑같네요. INotifyPropertyChanged를 구현하고 PropertyChanged 이벤트를 발생시킵니다.

4 MainWindow.xaml.cs의 코드-비하인드입니다.

```
public partial class MainWindow : Window
{
    GuyManager guyManager;

    public MainWindow() {
        InitializeComponent();

        guyManager = FindResource("guyManager") as GuyManager;
    }

    private void WriteJoe_Click(object sender, RoutedEventArgs e) {
        guyManager.WriteGuy(guyManager.Joe);
    }
    private void WriteBob_Click(object sender, RoutedEventArgs e) {
        guyManager.WriteGuy(guyManager.Bob);
    }
    private void WriteEd_Click(object sender, RoutedEventArgs e) {
        guyManager.WriteGuy(guyManager.Ed);
    }
    private void ReadNewGuy_Click(object sender, RoutedEventArgs e) {
        guyManager.ReadGuy();
    }
}
```

Guy Serializer의 시범운영

Guy Serializer로 데이터 컨트랙트 직렬화를 시험해 보세요.

- ★ 각 Guy 객체를 프로젝트 폴더 > bin > Debug 폴더에 파일로 씁니다. 그리고 작성된 사람의 정보를 읽기 위해서 ReadGuy 버튼을 클릭합니다. 파일을 읽기 위해서 TextBox에 있는 경로를 사용하죠. 다른 사람의 정보를 읽기 위해서 경로를 바꿔봅시다. 그리고 존재하지 않는 파일을 읽어 봅시다. 무슨 일이 일어나나요?
- ★ 전에 만든 텍스트 편집기를 열어 봅시다. 파일을 열고 저장하는 파일 피커의 옵션을 XML 파일에 추가했기 때문에 사람 (Guy) 파일을 편집할 수 있습니다. 한 사람의 파일을 열어 보고, 수정 및 저장을 해 봅시다. 그리고 다시 편집한 파일을 Guy Serializer에서 읽어 봅시다. XML이 유효하지 않다면 어떤 일이 일어날까요? 카드 무늬나 숫자가 변했다면, 유효한 enum 값과 일치할까요?
- ★ [DataMember(Name="...")]의 이름을 추가하거나 제거해 보세요. XML에 무슨일이 일어나나요? 컨트랙트를 수정했을 때, 이전에 저장된 XML 파일을 불러오나요? 프로그램이 올바르게 동작하도록 XML 파일을 고쳐 봅시다.
- ★ Card 데이터 컨트랙트의 네임스페이스를 바꿔보세요. XML에 무슨 일이 일어나나요?

바보 같은 질문이란 없습니다

Q: 가끔 XAML이나 코드를 수정할 때, IDE 디자이너는 다시 빌드하라는 메시지를 줍니다. 왜 그럴까요?

A: IDE의 XAML 디자이너는 정말로 똑똑합니다. XAML 코드를 수정할 때, 수정된 페이지를 실시간으로 볼 수 있게 해줍니다. XAML이 정적 리소스를 사용할 때, Window 클래스에 대한 객체 참조를 추가하는 것은 이미 알고 있습니다. 이 객체가 디자이너에서 보이기 위해서는 초기화될 필요가 있습니다. 그리고 정적 자원에 사용되는 클래스를 수정할 경우, 클래스를 다시 빌드할 때 비로소 디자이너가 업데이트됩니다. IDE가 여러분의 프로젝트를 다시 빌드 하라고 요청하는 것은 당연한 일입니다. 정적 리소스를 인스턴스화할 필요가 있는 메모리에서 컴파일된 코드가 실제로 없다면 말이죠.

IDE에서 이렇게 동작하는 과정을 한번 살펴봅시다. Guy Serializer를 열고, Guy.ToString() 메서드에서 반환값을 다른 단어로 수정합니다. 디자이너는 아직 수정되지 않은 결과값을 보여 줍니다. 이제 메뉴에서 솔루션 다시 빌드를 선택합니다. 코드가 다시 빌드되자마자 디자이너 스스로 업데이트합니다. 다른 코드도 바꿔 보세요. 잠깐, 다시 빌드는 아직 하지 마세요. 대신 Guy 객체에 바인딩된 또 다른 TextBlock을 추가해 보세요. IDE가 다시 빌드될 때까지, 객체의 이전 버전을 사용하고 있을 거예요.

Q: 네임스페이스가 아직도 헷갈리네요. 프로그램의 네임스페이스와 XML 파일의 네임스페이스는 뭐가 다른 거죠?

A: 네임스페이스가 왜 필요한지 다시 한번 생각해 봅시다. C#, XML 파일, 윈도우 파일시스템, 웹 페이지 등 모두 다른 네이밍 시스템을 사용합니다(보통 연관되어 있습니다). 각 클래스, XML 문서, 파일 혹은 웹 페이지에 대한 자신의 유일한 이름이 있죠. 그래서 왜 이러한 네임스페이스가 중요한 걸까요? 9장에서 브라이언이 변명 폴더를 알아내기 위해서 KnownFolders 클래스를 만들었습니다. 이제 .NET 프레임워크가 이미 KnownFolders 클래스가 있는지 살펴보세요. .NET의 KnownFolders 클래스는 Windows.Storage 네임스페이스에 있어서 걱정할 필요가 없습니다. 같은 클래스 이름이라도 나란히 행복하게 있군요. 정말 **명확(disambiguation)**하군요.

데이터 컨트랙트 또한 차이를 보여야 합니다. 이 책에서 몇 개의 다른 Guy 클래스를 봤습니다. 만약 2개의 다른 컨트랙트를 서로 다른 Guy로 직렬화하고 싶다면 어떻게 해야 할까요? 이 둘을 명확히 구분하기 위해서 서로 다른 네임스페이스에 두면 됩니다. 그리고 여러분은 클래스와 컨트랙트를 혼동할 필요가 없습니다. 네임스페이스는 클래스에 대한 것들과 분리되는 게 맞는 거죠.

한 가지 더 있습니다. WPF 응용 프로그램에서 원품과 같은 OpenFileDialog와 SaveFileDialog 클래스를 사용할 수 있습니다. 조금 더 자세한 정보와 코드 예제는 아래에 있습니다.
<http://msdn.microsoft.com/ko-kr/library/aa969773.aspx>

Chapter 12

9장에서 브라이언의 변명 관리 프로그램을 만들었습니다. 그런데 몇몇의 버그가 있네요. 이번 장에서 이들을 고쳐 봅시다.



WPF의 예외 처리는 원품과 윈도우 스토어 앱과 같은 방식으로 동작합니다.

12장의 부록은 XAML 코드가 없습니다. 왜냐하면 WPF 응용 프로그램, 원품 프로그램, 윈도우 스토어 앱, 심지어 콘솔 응용 프로그램에서든지 상관없이 헤드 퍼스트 C#에서는 같은 예외 처리에 대해서 학습하는 내용이 같기 때문이죠.

12장에서 참고해야 할 사항입니다.

- ★ 책에서 619쪽까지 읽어 주세요. “연필을 깎으며” 문제도 풀어 주세요.
- ★ 이 부록은 620, 621쪽을 대체합니다.
- ★ 책 622, 623쪽을 읽어 주세요.
- ★ 부록 624-634쪽을 읽어 주세요. 책 635쪽을 건너뜁니다.
- ★ 그리고 책 12장의 나머지 부분을 읽어 주세요.
- ★ 13장은 책을 읽으면 됩니다.

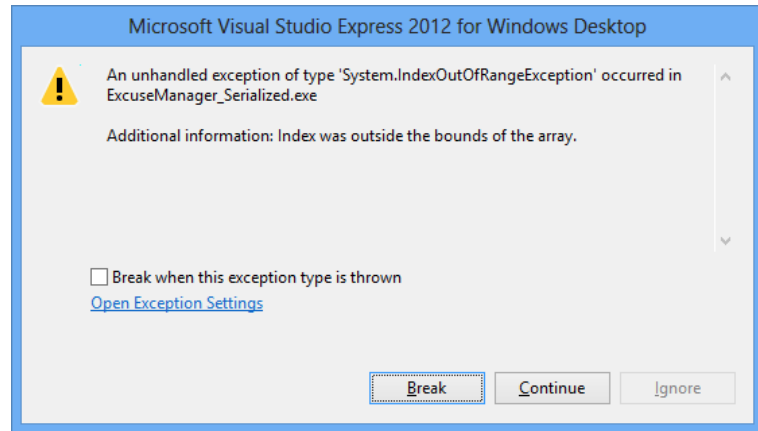
이번 장을 마치고, 책 13장으로 바로 가서 읽으면 됩니다. 13장은 윈도우 8 스토어 앱에 의존하지 않습니다.

브라이언의 코드도 예상치 못한 방식으로 동작했군요

브라이언은 변명 관리 프로그램을 만들 당시에 사용자가 빈 디렉토리에서 Random Excuse 버튼을 클릭하려고 한다는 점은 전혀 예상치 못했습니다.

이 예제는 9장에서 만들었던 변명 관리 프로그램입니다. 부록과 코드가 맞지 않다면, 아래의 사이트에서 코드를 내려받으세요.
<http://www.hanbit.co.kr/exam/2165>

- 1 노트북에 설치한 브라이언의 변명 관리 프로그램은 디렉토리가 비어 있는 상태에서 Random 버튼을 클릭했을 때 문제가 발생했습니다. 먼저 코드를 살펴보고 어떤 점이 잘못됐는지 찾아봅시다. IDE 외부에서 프로그램 실행 시 띄워진 처리되지 않은 예외 창이 여기에 나와 있습니다.



직접 해 봅시다!

- 2 여기서부터 시작해 볼까요? 내용을 보니 어떤 값이 어떤 범위에 들어가지 못한다고 하는군요. 아래의 표시된 코드 줄에 중단점을 설정하고, 디버거를 실행해 봅시다.

```
public async void OpenRandomExcuseAsync()
{
    string[] fileNames = Directory.GetFiles(folder, "*.excuse");
    OpenFile(fileNames[random.Next(fileNames.Length)]);
}
```

- 3 문제를 추적하기 위해서 조사식 창을 이용해 봅시다. 이름에 files.Length를 추가하세요. 0을 반환하는 것 같군요. 그리고 random.Next(fileNames.Length)를 추가해 보면, 역시 0을 반환합니다. 이름에 fileNames[random.Next(fileNames.Length)]를 추가합니다. 조사식 창의 값 열에서 과정 1에서 본 오류(배열의 인덱스 범위가 넘어 감: Out of bounds array index)를 볼 수 있습니다.

Name	Value
fileNames.Length	0
random.Next(fileNames.Length)	0
fileNames[random.Next(fileNames.Length)]	Out of bounds array index

조사식 창에서 메시지를 호출하고, 반환된 값으로 배열의 인덱스(인덱서)를 사용할 수 있습니다. 여기서 예외가 발생한다면, 조사식 창에서 그 예외를 이렇게 볼 수 있죠.

- 4 무슨 일이 일어났죠? 빈 폴더를 선택했을 때, Directory.GetFiles()에서 빈 배열을 반환합니다. 그래서 fileNames.Length는 0이고, Random.Next()에 0을 전달하면 항상 0을 반환합니다. 빈 배열에 0번째 요소를 접근해 보세요. 그러면 인덱스가 배열의 범위를 벗어났다(Index was outside the bounds of the array)는 메시지와 함께 System.IndexOutOfRangeException 예외가 발생합니다.

문제 파악을 했으니, 한번 고쳐 봅시다. 임의의 변명(Random excuse)을 불러오기 전에, 선택한 폴더에서 변명 파일이 있는지 확인하면 됩니다.

```
private void randomExcuse_Click(object sender, EventArgs e)
{
    if (Directory.GetFiles(selectedFolder).Length == 0)
        MessageBox.Show("There are no excuse files in the selected folder.");
    else if (CheckChanged())
    {
        currentExcuse = new Excuse(random, selectedFolder);
        UpdateForm(false);
    }
}
```

Excuse 객체를 생성하기 전에 폴더에 변명 파일이 있는지 검사해서 예외가 발생하는 것을 방지하고, 도움이 되는 메시지를 보여 두고 있습니다.

위 코드에 대해서 어떻게 생각하세요? 이 코드를 폼에 넣는 게 좋을까요? 아니면 Excuse 클래스 내부에 캡슐화를 하는 것이 더 좋을까요?



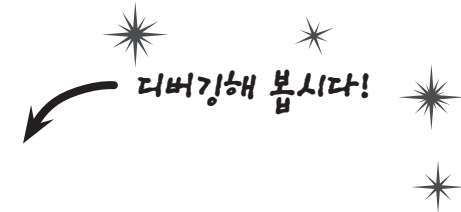
오, 알겠어요. 예외가 항상 나쁜 것만은 아니군요. 예외는 버그를 식별하기도 하지만, 대부분은 제가 예상한 것과 다르게 일어나는 상황이 무엇인지 알려주는군요.

맞습니다. 예외는 예상치 못한 동작을 하는 코드가 어디에 있는지 찾아낼 수 있는 매우 유용한 도구입니다.

많은 프로그래머들이 예외를 처음 접하게 되면 당황하지만, 실제로는 매우 유용하며 좋은 점이 많습니다. 예외는 여러분의 코드가 언제 예상치 못한 상황에 대응하게 될 것인지를 알 수 있게 도움이 되는 단서를 제공해 줍니다. 또한 여러분의 프로그램이 처리해야만 하는 새로운 시나리오에 대한 정보를 알려주며, 이를 처리할 수 있는 기회를 제공합니다.

디버거를 사용해서 변명 관리 프로그램의 문제를 살펴봅시다

디버거를 사용해서 변명 관리 프로그램에서 마주쳤던 문제를 좀 더 자세히 살펴 보죠. 지난 몇 장에서도 디버거를 사용했습니다. 이번에는 변명 관리 프로그램의 문제를 자세히 들여다보기 위해서 디버깅을 단계별로 진행해 봅시다.



1 Random 버튼의 이벤트 핸들러에 중단점을 추가합니다.

어디에서 문제가 발생하는지는 알고 있습니다. 바로 빈 폴더를 선택한 후에 Random Excuse 버튼을 클릭할 때 예외가 발생합니다. 따라서 이 버튼의 코드-비하인드 창을 열고 메서드 첫 번째 줄을 클릭한 후, 디버그(Debug) 메뉴 > 중단점 설정/해제(Toggle Breakpoint)를 선택하거나 F9키를 누른 후, 프로그램을 실행하세요. 빈 폴더를 선택하고 Random 버튼을 클릭해서 프로그램이 중단점에서 멈추도록 합니다.

```
private void randomExcuse_Click(object sender, EventArgs e)
{
    if (Directory.GetFiles(selectedFolder).Length == 0)
        MessageBox.Show("There are no excuse files in the selected folder.");
    else if (CheckChanged())
    {
        currentExcuse = new Excuse(random, selectedFolder);
        UpdateForm(false);
    }
}
```

2 Excuse 생성자에서 한 단계씩 코드를 실행합니다.

문제를 재현해야 하지만, 문제를 해결하는 코드를 이미 추가했습니다. 다음 과정을 따라 해서 문제를 재현해 봅시다. currentExcuse = new Excuse(random, selectedFolder); 오른쪽 클릭한 후, 다음 문 설정(Ctrl-Shift-F10)을 선택한 다음, 한 단계씩 코드 실행(F11)을 선택해서 생성자로 들어갑니다.

```
public Excuse(Random random, string folder)
{
    string[] fileNames = Directory.GetFiles(folder, "*.excuse");
    OpenFile(fileNames[random.Next(fileNames.Length)]);
}
```

예외를 피하기 위해서 추가한 해결 방안의 코드를 건너뛴 수 있도록 디버거를 사용했습니다. 그러면 Excuse 생성자가 다시 예외를 던질 수 있습니다.

3 프로그램이 예외를 던질 때까지, 한 단계씩 코드를 실행하세요.

조사식 창은 아주 편리한 기능을 제공합니다. 예외를 재현하기 위해서 조사식 창을 사용해 봅시다. 예외를 발생시키기 위해서 프로시저 단위 실행(F10)을 두 번 선택합니다. 그리고 나서 fileNames.Length를 선택한 후 오른쪽 클릭을 한 후, Add Watch (조사식 추가)를 선택해서 조사식을 추가합니다. 그리고 random.Next(fileNames.Length)와 fileNames[random.Next(fileNames.Length)]도 마찬가지로 조사식 창에 추가합니다.

Name	Value
fileNames.Length	0
random.Next(fileNames.Length)	0
fileNames[random.Next(fileNames.Length)]	Out of bounds array index

조사식 창은 유용한 기능이 많습니다. 조사식 창에 표시되는 변수와 필드의 값을 바꿀 수 있습니다. 심지어 메서드를 실행하거나 새로운 객체를 생성할 수 있습니다. 값 옆에 표시되는 이 아이콘은 메서드를 다시 실행해서, 값을 다시 계산할 수 있습니다.

4 Exception 객체를 조사식 창에 추가합니다.

디버깅은 프로그램에서 범죄 현장 조사를 수행하는 것과 비슷합니다. 문제의 원인이 무엇인지를 알아야 비로소 그 원인에 대해서 조금 더 세부적으로 살펴볼 수 있습니다. 그래서 단서를 따라 범인을 추적하는 CSI 키트, 즉 디버거가 필요하죠. 그리고 또 하나의 다른 팁이 있습니다. 조사식 창에서 \$exception을 추가해 보세요. Exception 객체가 발생한 내용을 여러분에게 보여 줍니다.

Name	Value	Type
\$exception	{"Index was outside the bounds of the array."}	System.Exception (System.In
[System.IndexOutOfRangeException]	{"Index was outside the bounds of the array."}	System.IndexOutOfRangeException
Data	{System.Collections.ListDictionaryInternal}	System.Collections.IDictiona
HelpLink	null	string
HResult	-2146233080	int
InnerException	null	System.Exception
Message	"Index was outside the bounds of the array"	string
Source	"ExcuseManager_Serialized"	string
StackTrace	" at ExcuseManager_Serialized.Excuse..cto"	string

예외가 발생했을 때, 예외가 발생한 곳으로 돌아가서 디버거로 예외를 재현해 보세요. Exception 객체를 이용해서 문제의 원인을 파악할 수 있습니다.

바보 같은 질문이란 없습니다

Q : 중단점을 어디에 설정해야 되죠?

A : 정말 좋은 질문입니다. 사실 이 질문에 대한 정답은 없습니다. 코드에서 예외가 발생할 때 예외를 발생시킨 문장부터 살펴보는 것이 좋습니다. 하지만 보통은 실제로 문제가 발생한 장소는 예외 발생 훨씬 이전이며, 발생한 예외는 단지 그 부산물일 뿐입니다. 예를 들어, 0으로 나누는 에러를 발생시킨 문장은 실제로 사용된 적이 없는 10개의 문장 이전에 생성된 값으로 나눗셈을 하게 됩니다. 결국 상황에 따라 다르므로 중단점을 어디에 삽입해야 하는지에 대한 정답은 없지만, 여러분의 코드가 어떤 식으로 수행되는지 알고 있다면 적당한 지점이 어디인지 알 수 있을 것입니다.

Q : 조사식 창에서 메서드를 실행할 수 있나요?

A : 예, 프로그램에서 이상 없이 동작한 문장은 조사식 창에서도 문제없이 동작합니다. 프로그램을 실행하고, 중단한 후 다음 내용을 조사식 창에 추가해 보세요.

```
System.Threading.Thread.Sleep(2000)
```

이 메서드는 프로그램을 2초 동안 멈추게 합니다. 어떤 일이 일어날까요? 이 메서드가 실행되는 동안 IDE가 2초 동안 잠시 멈추고, 기다리는 커서를 표시합니다. Sleep()은 반환 값이 없으므로, 조사식 창은 “식을 계산했지만 값이 없습니다”라는 메시지를 표시하고, 반환 값이 없음을 알려줄 것입니다. 뿐만 아니라 코드를 작성하는 데 도움을 주는 작은 팝업 창이 나타나는데, 현재 메모리에 있는 해당 객체에 사용 가능한 속성과 메서드의 목록을 보여 주므로 아주 유용합니다.

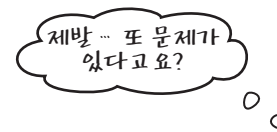
Q : 잠깐만요! 조사식 창에서 뭔가 실행할 때, 실행하고 있는 프로그램의 실행 방식을 바꿀 수 없다는 건가요?

A : 예! 영구적인 영향을 주지 않습니다. 하지만, 결과에는 영향을 끼칠 수 있죠. 디버거 내부의 필드에 마우스를 올리면, 프로그램의 동작을 변경할 수 있습니다. 속성의 get 접근자에 실행할 메서드가 있다면, 그 속성에 마우스를 올릴 경우 이 메서드가 실행됩니다. 그리고 메서드가 어떤 값을 설정한다면, 프로그램을 다시 실행할 경우 그 값이 남아 있습니다. 이러한 경우, 디버거 내부에서 예측할 수 없는 결과를 일으킬 수 있습니다. 예측할 수 없고 무작위적인 결과를 하이젠버그(heisenbug)라 부릅니다(상자 속에 갇힌 고양이와 하이젠베르크란 물리학자 이름에서 따온 거예요).

IDE에서 프로그램을 실행할 때 처리되지 않은 예외는 마치 중단점을 설정한 것처럼 프로그램을 중지시킵니다.

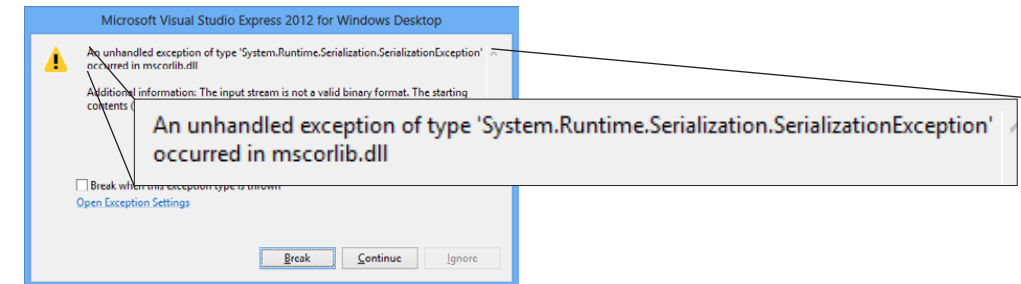
어휴, 코드에 여전히 문제가 있네요

브라이언은 가벼운 마음으로 변명 관리 프로그램을 사용하다가 변명 관리 프로그램으로 만들어지지 않은 XML 파일로 가득 차 있는 폴더를 잘못 선택했습니다. 이렇게 했을 경우 어떤 일이 일어나는지 살펴봅시다.

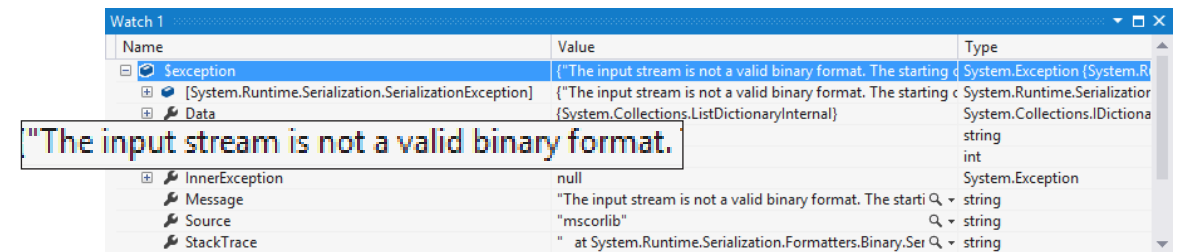


1 브라이언이 고민하고 있는 문제를 재현할 수 있습니다. 변명 파일로 직렬화되지 않은 아무 파일을 선택해서 “.excuse” 확장자를 붙여 주세요.

2 IDE에서 변명 관리 프로그램을 실행한 후, 변명을 열어보세요. 예외가 발생합니다. 메시지를 보고, 중단(Break) 버튼을 눌러 무엇이 문제인지 살펴봅시다.



3 지역 창을 열어 조사식 창에 \$exception을 입력한 후, + 버튼을 눌러 확장합니다. 무엇이 잘못되었는지 알아내기 위해서, 이들의 멤버를 자세히 살펴 봅시다.



왜 프로그램은 예외를 발생시킬까요?

변명(.excuse) 형식이 아닌 XML 파일을 선택했을 때, 프로그램이 멈추는 게 맞는 걸까요?

이 문제를 해결할 수 있는 방법이 있나요?

잠깐만요. 물론 잘못된 파일을 읽으려고 하니 프로그램이 충돌했겠죠. 항상 사용자들은 모든 것을 망쳐놓는다니까요. 이런 상황에서 무엇을 해야 하죠?



실제로 여러분이 할 수 있는 것이 있습니다.

사용자들이 항상 문제가 되는 것은 사실입니다. 세상살이가 다 그런 거죠. 그렇다고 여러분이 할 수 있는 것이 아무것도 없지는 않습니다. 잘못된 데이터를 입력하거나 다른 예상치 못한 상황들을 처리하는 프로그램을 부르는 이름이 있는데, 바로 **튼튼한(robust)** 프로그램이라고 합니다. C#은 여러분의 프로그램을 좀 더 튼튼하게 만들 수 있게 도움을 주는 실제로 강력한 예외 처리 도구를 제공하고 있습니다. 비록 사용자들이 무엇을 하는지 제어할 수 없지만, 사용자들이 방해할 때 프로그램이 충돌하지 않게 할 수 있습니다.

강력한, 튼튼한(robust), 형용사 정반대의 상황을 극복하거나 버틸 수 있는 능력.
타코마 다리 붕괴 사고 후에 도시 건설 팀은 이를 대체할 좀 더 **튼튼한** 다리 설계를 검토하고 있다.

파일에 직렬화된 객체가 정확하지 않을 경우 BinaryFormatter 클래스 또한 SerializationException 을 발생시킵니다.DataContractSerializer보다 더 신경 쓸게 많죠.



조심하세요

잘못된 직렬화 파일이 있다면, 시리얼라이저는 예외를 발생시킵니다.
변경 관리 프로그램에서 SerializationException을 발생시키는 것은 쉽습니다. 그냥 직렬화된 Excuse 객체를 가지고 있지 않은 파일을 열면 되죠. 파일로부터 객체를 역직렬화하려고 시도할 때 DataContractSerializer는 당연히 읽으려고 하는 클래스와 일치하는 직렬화된 객체를 가지고 있는 파일을 읽고 있다고 생각합니다. 만약 파일에 다른 내용이 있다면 ReadObject() 메서드는 SerializationException을 발생시킵니다.

try와 catch로 예외를 처리합시다

C#에서는 “이 코드를 한 번 실행해 보고(try) 예외가 발생하면, 이 예외를 처리하는(catch), 다른 코드를 수행한다”라고 말할 수 있습니다. 여기서 시도하는 부분이 바로 try 블록이며, 예외를 처리하는 부분이 바로 catch 블록입니다. catch 블록에서는 프로그램을 중단하는 대신, 친절한 에러 메시지를 보여 줄 수 있습니다.

```
Excuse.cs 파일 위쪽에 아래의 using문이 필요합니다.
using System.Runtime.Serialization;
using System.Windows.Forms;
```

```
public async Task ReadExcuseAsync () {
    try
    {
        this.ExcusePath = excusePath;
        BinaryFormatter formatter = new BinaryFormatter ();
        Excuse tempExcuse;
        using (Stream input = File.OpenRead(excusePath))
        {
            tempExcuse = (Excuse)formatter.Deserialize(input);
        }
        Description = tempExcuse.Description;
        Results = tempExcuse.Results;
        LastUsed = tempExcuse.LastUsed;
    }
    catch (SerializationException)
    {
        MessageBox.Show("Unable to read " + excusePath);
        LastUsed = DateTime.Now;
    }
}
```

여기가 try 블록입니다. try 를 사용해서 예외 처리를 시작합니다.

try 블록 내부에 예외를 발생시킬 만한 코드를 넣습니다. 만약 예외가 발생하지 않는다면 다른 때와 마찬가지로 정상적으로 실행되며, catch 블록에 있는 문장들은 무시됩니다. 하지만 예외를 발생시키면, try 블록에 있는 나머지 문장들은 실행되지 않습니다.

메서드 전체가 이 try 블록으로 되어 있어서, 코드를 쉽게 알아볼 수 있을 거예요.

catch 블록은 바로 다음에 예외를 처리하는 블록이 나온다는 것을 의미합니다.

예외가 발생할 때 프로그램은 즉시 catch 블록을 수행합니다.



브레인 파워

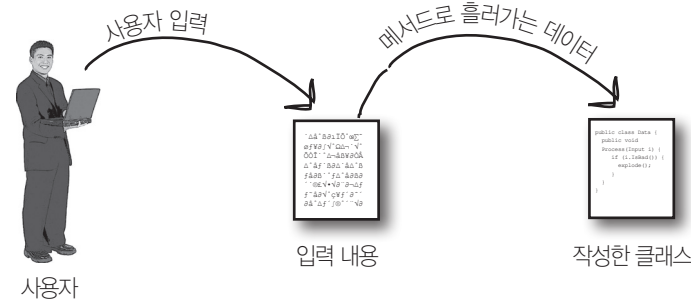
예외를 발생시켜 여러분의 코드가 자동으로 catch 블록으로 넘어간다면 예외가 발생하기 전에 작업했던 객체나 데이터에는 무슨 일이 일어날까요?

위의 코드는 간단한 예외 처리의 예입니다. 프로그램을 중지하고 예외 메시지를 보여준 후 나머지 부분을 계속 수행하는군요.

위험한 메서드 호출

사용자는 예측 불가능합니다. 그들은 모든 종류의 왜곡된 데이터를 프로그램에 입력해서 여러분이 절대로 예상하지 못한 방식으로 프로그램을 사용합니다. 여기까지는 그나마 다행입니다. 왜냐하면 잘 짜인 예외 처리를 사용하면 예상치 못한 입력사항을 처리할 수 있기 때문입니다.

1 사용자가 예상치 못한 내용을 입력했다고 칩시다.



2 이 메서드는 런타임 시에 동작하지 않을 것 같은 원가 위험한 것을 합니다.

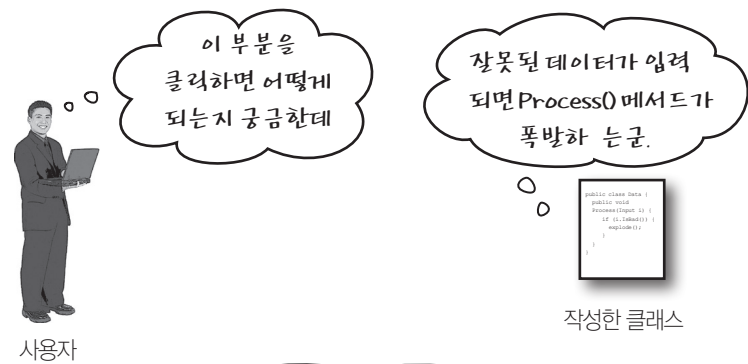
```

public class Data {
    public void Process(Input i) {
        if (i.IsBad()) {
            explode();
        }
    }
}

```

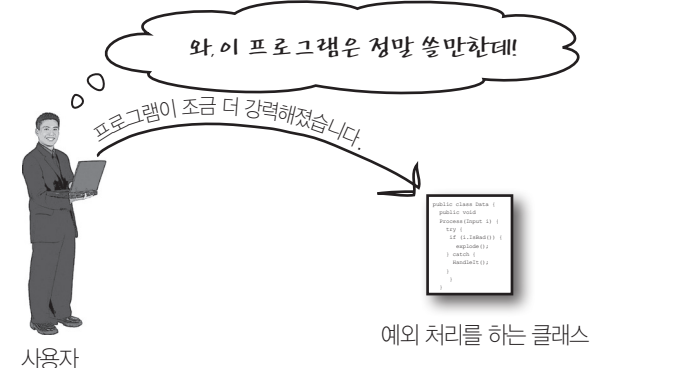
“런타임”은 “프로그램이 실행되는 동안”을 나타냅니다. 일부 사람들은 예외를 “런타임 오류”라고 부르곤 하죠.

3 여러분이 호출하는 메서드가 위험하다는 사실을 알아야 합니다.



예외를 발생시키지 않는 덜 위험한 방법을 찾을 수만 있다면, 이보다 더 좋은 방법은 없습니다! 하지만 어느 정도 위험은 피할 수 없으므로 이렇게 처리해야 합니다.

4 위험한 상황으로 인한 오작동을 처리할 수 있는 코드를 작성합니다. 만약의 사태에 대비해야 하니까요.



바보 같은 질문이란 없습니다

Q : 그렇다면 try와 catch는 언제 사용하나요?

A : 위험성이 있거나 예외가 발생할 소지가 있는 코드를 작성할 때 사용하세요. 문제는 과연 어떤 코드가 위험하고, 어떤 코드가 더 안전한가 하는 것입니다.

여러분은 이미 사용자가 입력한 내용을 기반으로 처리하는 코드가 위험해진다는 것을 봤습니다. 사용자들은 잘못된 파일이나 숫자 대신 문자를, 날짜 대신 이름을 입력하거나, 여러분이 상상할 수 있는 모든 곳을 클릭해합니다. 좋은 프로그램일수록 모든 입력사항에 대해 조심히, 예측 가능한 방법으로 처리합니다. 사용자에게 처리한 결과를 보여 주지는 않겠지만, 문제를 발견하고 해결책을 제시할 수는 있어야 합니다.

Q : 어떻게 하면 미리 알지 못하는 문제에 관한 해결책을 프로그램이 제시할 수 있죠?

A : 그런 일을 바로 catch 블록이 합니다. catch 블록은 try 블록에서 예외가 발생할 때만 수행되며, 여러분은 사용자에게 뭔가 잘못된 것이 있고 이를 올바르게 고쳐야 한다는 사실을 확인시켜줄 수 있는 기회를 가지게 된 거죠.

만약 변명 관리 프로그램에 잘못된 데이터가 입력될 때 충돌하기만 한다면 이 프로그램은 별로 쓸만하지 않겠죠. 또한 입력사항을 읽어 폼에 쓰레기 데이터를 보여준다면 이 또한 유용하지 않고 오히려 더 나쁜 프로그램이라고 얘기할 것입니다. 하지만 파일을 읽을 수 없다는 사실을 사용자에게 알려

주는 에러 메시지를 보여준다면 사용자는 무엇이 잘못되었고 그 문제를 고칠 수 있는 지에 대한 단서를 얻게 될 것입니다.

Q : 디버거는 예외를 해결하는 데만 사용되나요?

A : 아닙니다. 디버거는 실제로 작성한 코드를 검사하는 데 사용할 수 있는 유용한 도구의 하나일 뿐입니다. 코드를 단계별로 실행해서 특정 변수나 필드의 값을 검사하는 데 유용한 도구죠. 여러분이 복잡한 메서드를 작성할 때 그것이 제대로 작동되고 있는지 확인할 때 사용할 수 있습니다. 하지만 “디버거”란 이름에서 알 수 있듯이, 이것의 주용도는 버그를 추적해서 제거하는 것입니다. 가끔씩 그러한 버그들이 예외가 되죠. 앞으로 여러분은 꽤 오랜 시간 동안 예상하지 못한 결과를 산출하는 코드처럼 다양한 종류의 문제점을 찾아내는 데 디버거를 사용할 것 같은군요.

Q : 조사식 창에 대해 아직도 잘 모르겠어요. 이것은 무엇을 위한 것이죠?

A : 프로그램을 디버깅할 때 여러분은 임의의 필드나 변수들 값이 어떻게 변하고 있는지 알고 싶어할 것입니다. 바로 이때 조사식 창이 등장합니다. 일부 변수들을 조사식에 추가하면, 조사식 창은 코드가 단계별로 실행될 때마다 해당 변수의 값을 갱신합니다. 조사식은 모든 문장에서 정확히 무슨 일이 일어나고 있는지 감시할 수 있게 하며, 이는 문제점을 추적할 때 정말 유용하게 사용할 수 있습니다.

또한 조사식 창에 여러분이 원하는 문장을 입력할 수 있죠. 심지어는 메서드를 호출해 주고, 그 내용을 계산해서 결과를 보여 줍니다. 만약 문장 내에서 어떤 필드나 변수를 갱신한다면 조사식 창에서도 적용되어 프로그램이 실행되는 동안 값이 변경됩니다. 이런 점은 예외나 다른 버그들을 재현하는데 있어 아주 유용한 도구로 사용할 수 있죠.

조사식 창에서 만든 변경사항은 메모리에 있는 데이터에 영향을 주지 않으며, 프로그램이 실행되는 동안에만 지속됩니다. 프로그램을 재 시작하면 변경했던 모든 값들은 원래상태로 남아 있게 될 거예요.

catch 블록은 try 블록의 코드에서 예외가 발생할 때만 수행됩니다. 이를 이용해서 여러분은 사용자에게 문제점을 수정하기 위한 정보를 제공할 수 있습니다.

디버거를 사용해서 try/catch 흐름을 따라가 보죠

예외를 처리하는 중요한 부분이 바로 try 블록에 있는 문장에서 예외가 발생할 때인데, 이때 블록에 있는 나머지 코드들은 중단됩니다. 일명 단락(short-circuited)된다고 표현하죠. 그리고 프로그램은 즉각 catch 블록의 첫 번째 라인으로 이동합니다. 하지만 항상 이렇게 동작하는 것은 아닙니다.



- 1 몇 페이지 전에 작성한 변명 관리 프로그램의 OpenFile() 메서드로 갑니다. 그리고 나서 try 블록에서 시작되는 [에 중단점을 설정합니다.
- 2 디버깅을 시작해서 유효하지 않은 변명 파일을 열어 봅시다(확장자가 .xml 이어야 합니다). 디버거가 중단점에서 멈출 때, 프로시저 단위 실행(혹은 F10키)을 다섯 번 클릭합니다. 그러면 Excuse 객체를 역직렬화하는 ReadObject() 메서드로 가게 됩니다. 디버거는 아래와 같이 보일 겁니다.

```
private void OpenFile(string excusePath)
{
    try
    {
        this.ExcusePath = excusePath;
        BinaryFormatter formatter = new BinaryFormatter();
        Excuse tempExcuse;
        using (Stream input = File.OpenRead(excusePath))
        {
            tempExcuse = (Excuse)formatter.Deserialize(input);
        }
        Description = tempExcuse.Description;
        Results = tempExcuse.Results;
        LastUsed = tempExcuse.LastUsed;
    }
    catch (SerializationException)
    {
        MessageBox.Show("Unable to read " + excusePath);
        LastUsed = DateTime.Now;
    }
}
```

try 블록이 시작하는 등괄호에 중단점을 설정하세요.

↳ 트립으로부터 Excuse 객체를 읽기 위해서, 노란색 막대의 "다음 문장"까지 프로시저 단위 실행을 합니다.

- 3 프로시저 단위 실행을 눌러 다음 문장을 수행합니다. 디버거가 Deserialize() 문을 실행하자마자 예외가 발생합니다. 그리고 메서드의 예외가 발생한 곳부터 try 블록이 끝나는 지점까지 흐름이 일단락되고(short-circuits), 바로 catch 블록으로 이동합니다.

디버거는 노란색 블록으로 "다음 문장"을 표시하고, 블록의 나머지 부분은 회색으로 표시하는데, 이는 곧 실행될 부분을 의미합니다.

```
private void OpenFile(string excusePath)
{
    try
    {
        this.ExcusePath = excusePath;
        BinaryFormatter formatter = new BinaryFormatter();
        Excuse tempExcuse;
        using (Stream input = File.OpenRead(excusePath))
        {
            tempExcuse = (Excuse)formatter.Deserialize(input);
        }
        Description = tempExcuse.Description;
        Results = tempExcuse.Results;
        LastUsed = tempExcuse.LastUsed;
    }
    catch (SerializationException)
    {
        MessageBox.Show("Unable to read " + excusePath);
        LastUsed = DateTime.Now;
    }
}
```

- 4 계속(Continue) 버튼(F5)을 클릭해서 프로그램을 다시 실행하면 노란색의 "다음 문장"이 표시된 부분에서 시작합니다. 이 경우엔 catch 블록이 되겠죠. 아무 일이 일어나지 않은 것처럼 대화상자를 표시합니다. 이렇게 예외처리를 하는군요.

여기 유용한 팁이 있군요. C# 프로그래머 면접 시 생성자 내부에서 발생한 예외를 어떻게 처리할 것인지 묻는 경우가 많죠.



조심하세요

예외가 발생할 것 같은 코드는 생성자 밖에서 작성하세요.

void형이 아니더라도 지금까지 생성자는 값을 반환하지 않는다는 것을 봤습니다. 생성자는 실제로 아무것도 반환하지 않기 때문이죠. 생성자의 목적은 객체를 초기화하는 것이며, 이로 인해 생성자 내부에서 예외를 처리할 경우 문제가 발생할 수 있습니다. 생성자 내부에서 예외가 발생할 때 클래스를 초기화하려는 문장에서 객체의 인스턴스를 생성하지 못합니다.

뒷정리를 잘합시다

언제나 실행되는 코드를 작성하려면 finally문을 사용하세요

프로그램이 예외를 발생시킬 때 몇 가지 일이 일어나는데, 발생한 예외가 **처리되지 않는다면** 프로그램은 실행을 중지하고 충돌하게 되죠. 예외가 **처리된다면** 여러분의 코드는 catch 블록으로 이동합니다. 하지만 try 블록에 있는 나머지 코드들은 어떻게 될까요? 여러분이 스트림을 닫거나 중요한 리소스들을 해제해야 한다면 어떨까요? 예외가 발생하거나 프로그램의 상태가 엉망이 되더라도 이러한 코드들은 수행될 필요가 있습니다. 이런 경우 try와 catch 블록 다음에 오는 finally 블록이 유용합니다. 예외가 발생하건 말건 **finally 블록은 항상 수행됩니다.**

```
private void OpenFile(string excusePath) {
    try {
        this.ExcusePath = excusePath;
        BinaryFormatter formatter = new BinaryFormatter();
        Excuse tempExcuse;
        using (Stream input = File.OpenRead(excusePath))
        {
            tempExcuse = (Excuse)formatter.Deserialize(input);
        }
        Description = tempExcuse.Description;
        Results = tempExcuse.Results;
        LastUsed = tempExcuse.LastUsed;
    }
    catch (SerializationException) {
        MessageBox.Show("Unable to read " + excusePath);
        LastUsed = DateTime.Now;
    }
    finally
    {
        // 여기에 있는 모든 코드는 뭐든지 실행됩니다.
    }
}
```

NewExcuse()
메서드가 호출되며,
이 Excuse 객체를
다시 설정합니다. 만약
PropertyChanged
이벤트가 실행되지
않는다면, 페이지는
CurrentExcuse 녹성을
입지 않습니다. finally
블록은 예외가 발생하든
하지 않든 상관없이
PropertyChanged
이벤트가 실행되도록
해둡니다.

항상 SerializationException 같은 특정한 예외를 잡아내세요. 일반적으로 catch문 다음에 특정 종류의 예외를 명시해서 이 예외를 잡아냅니다. C#에서도 "catch (Exception)" 형태로 코드를 작성하는데 예외 유형을 생략하고 catch만 표기해도 됩니다. 이런 경우에는, 발생한 예외의 유형에 상관없이 **모든 예외를 잡아내죠.** 하지만 **이렇게 사용하는 것은 좋은 습관이 아닙니다.** 가능한 한 항상 특정 예외 타입을 명시하도록 하세요

알림: 부록 12장을 마치고, 책 13장으로 바로 가면 됩니다. 13장은 윈도우 8 스토어 앱에 의존하지 않습니다.

Chapter 14

14장에서는 많은 LINQ 쿼리가 있습니다. 책에는 이러한 쿼리들을 모두 윈도우 스토어 앱에서 사용합니다. 대신 이 부록에서는 WPF 응용 프로그램을 만들 겁니다.



LINQ는 모든 종류의 C# 프로그램에서 동작합니다.

이 책 14장의 주요 부분을 읽을 때, 다른 장들과는 다르게 구성되었다는 것을 볼 수 있습니다. 복잡한 LINQ 쿼리를 설명하기 위해서, 작은 콘솔 앱을 통해서 예제를 살펴봅시다. 이번 장의 전반에 걸쳐서 모든 쿼리를 단일 사용자 인터페이스로 결합하는 윈도우 스토어 앱 예제를 볼 수 있습니다. 이 부록의 다음 몇 페이지에 걸쳐서 이와 같은 쿼리를 실행하는 WPF 응용 프로그램을 만드는 법을 배워 봅니다. 14장의 부록을 아래와 같이 활용하면 좋습니다.

- ★ 책 701쪽까지 읽어 주세요.
- ★ 709쪽까지 윈도우 스토어 앱을 만드는 과정이 있지만, 읽어 주세요. 특히 **익명 유형(anonymous type)**이 나오는 부분을 넘어가면 안 됩니다. 이것은 Comic, ComicQuery, ComicQueryManager 클래스가 어떻게 동작하는지 이해하는 데 도움이 됩니다.
- ★ 710, 711쪽은 LINQ 쿼리의 기능에 대해 설명합니다. 712, 713쪽은 윈도우 스토어 앱에 관한 것이기 때문에 넘어가도 됩니다.
- ★ 714-724쪽을 읽어 주세요. 723쪽에 있는 연습문제는 하지 마세요.
- ★ 나머지 부분은 윈도우 스토어 앱에 관련된 겁니다. 넘어가 주세요. 이 부분을 부록의 724-727쪽에서 대체합니다.

WPF 만화 쿼리 응용 프로그램을 만들어 봅시다

책 14장을 읽으면서, LINQ 쿼리를 실행하는 윈도우 스토어 앱을 만드는 과정을 살펴보았습니다. 관심사 분리의 원칙을 따랐기 때문에, 데이터를 관리하고 쿼리를 실행하는 코드와 사용자 인터페이스를 생성하는 코드를 따로 분리했습니다. 이것은 비주얼 스튜디오의 분할 앱 템플릿으로 또 다른 앱을 구축할 때, **동일한 데이터 및 쿼리를 관리하는 클래스들을 쉽게 재사용**할 수 있게 해줍니다. 이와 같은 관심사 분리를 활용해서, 같은 데이터와 쿼리 클래스를 이용하여 WPF 응용 프로그램을 만들어 봅시다.



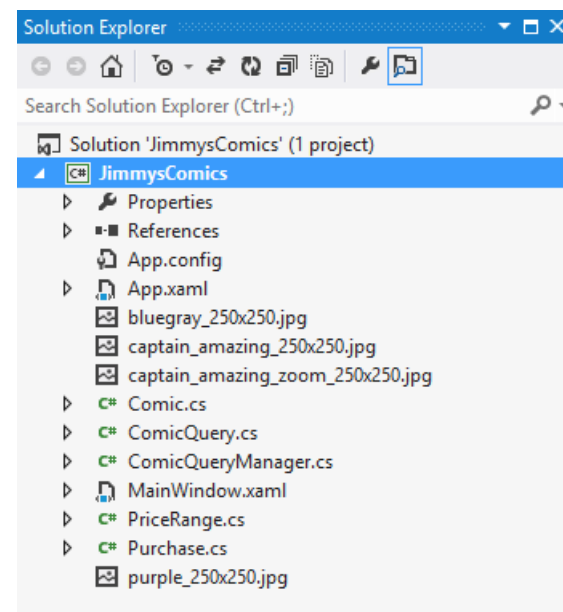
1 새로운 WPF 응용 프로그램을 생성하고, 만화책 앱에서 기존 클래스들과 이미지들을 추가합니다.

이 프로젝트를 시작하기 전에, 14장의 JimmysComics 앱의 코드를 내려받아야 합니다. 여기(<http://www.hanbit.co.kr/exam/2165>)에 소스 코드의 링크가 있습니다.

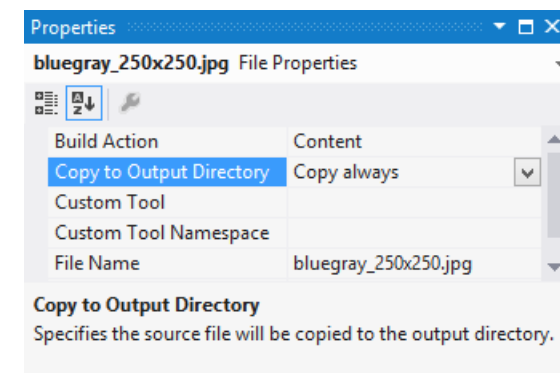
소스 코드를 내려받았다면, JimmysComics라는 WPF 응용 프로그램을 생성합니다. 그리고 나서 솔루션 탐색기의 프로젝트 이름에 마우스 오른쪽쪽을 클릭한 후, 기존 항목 추가를 선택해서 책에서 만든 윈도우 스토어 앱의 항목을 추가합니다(책의 웹 사이트에서 내려받을 수 있습니다).

- Purchase.cs • Comic.cs
- ComicQuery.cs • ComicQueryManager.cs
- PriceRange.cs,
- Assets 폴더의 bluegray_250x250.jpg, bluegray_250x250.jpg, captain_amazing_250x250.jpg, captain_amazing_zoom_250x250.jpg 파일들을 WPF 응용 프로그램의 루트 레벨에 추가해 주세요. XAML과 C# 파일에 나란히 놓아 주세요.

위 사항이 반영된 솔루션 탐색기 창입니다.



솔루션 탐색기의 각 이미지 파일을 클릭합니다. 그리고 **속성 창**의 빌드 작업(Build Action)에서 내용(Content)을, 출력 디렉터리로 복사(Copy to Output Directory)에서 항상 복사(Copy always)를 선택합니다. 이 작업을 각 .jpg 파일을 선택해서 해 주세요.



프로젝트의 이름을 다르게 했다면, 여러분이 추가한 C# 파일의 네임스페이스를 여러분의 프로젝트 이름에 맞게 변경해 두세요.

2 ComicQueryManager.cs에서 두 가지를 수정해 봅시다.

ComicQueryManager.cs에서 두 가지의 작은 변화가 있습니다. WPF 응용 프로그램은 Windows.UI 네임스페이스를 사용하지 않습니다. 이 네임스페이스는 윈도우 스토어 앱의 .NET 프레임워크에만 해당하기 때문이죠. 그래서 using문의 "Windows.UI"를 "System.Windows"로 바꿔줍니다.

```
using System.Collections.ObjectModel;
using System.Windows.Media.Imaging;
```

WPF 응용 프로그램에서 이미지를 불러오는 방식은 윈도우 스토어 앱과 조금 다릅니다. ComicQueryManager의 CreateImageFromAssets() 메서드를 아래와 같이 바꿔줍니다.

```
private static BitmapImage CreateImageFromAssets(string imageFilename)
{
    try
    {
        Uri uri = new Uri(imageFilename, UriKind.RelativeOrAbsolute);
        return new BitmapImage(uri);
    }
    catch (System.IO.IOException)
    {
        return new BitmapImage();
    }
}
```

프로젝트의 최상위 레벨 폴더에 .jpg 파일을 복사했습니다. 새 CreateImageFromAssets() 메서드는 이 파일들을 불러옵니다.

3 메인 윈도우의 코드-비하인드를 추가합니다.

MainWindow.xaml.cs에 대한 코드-비하인드입니다.

```
public partial class MainWindow : Window
{
    ComicQueryManager comicQueryManager;
    public MainWindow()
    {
        InitializeComponent();
        comicQueryManager = FindResource("comicQueryManager") as ComicQueryManager;
        comicQueryManager.UpdateQueryResults(comicQueryManager.AvailableQueries[0]);
    }
    private void ListView_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        if (e.AddedItems.Count >= 1 && e.AddedItems[0] is ComicQuery)
        {
            comicQueryManager.CurrentQueryResults.Clear();
            comicQueryManager.UpdateQueryResults(e.AddedItems[0] as ComicQuery);
        }
    }
}
```

ListView 컨트롤은 항목이 선택/해제될 때마다 SelectionChanged 이벤트를 발생시킵니다. 선택된 항목은 e.AddedItems 컬렉션에서 찾을 수 있습니다.

4 메인 윈도우의 XAML을 추가합니다.

메인 윈도우의 XAML 코드입니다. 만약 다른 프로젝트의 이름을 사용했다면 JimmysComics를 여러분이 만든 프로젝트의 네임 스페이스로 바꿔주세요.

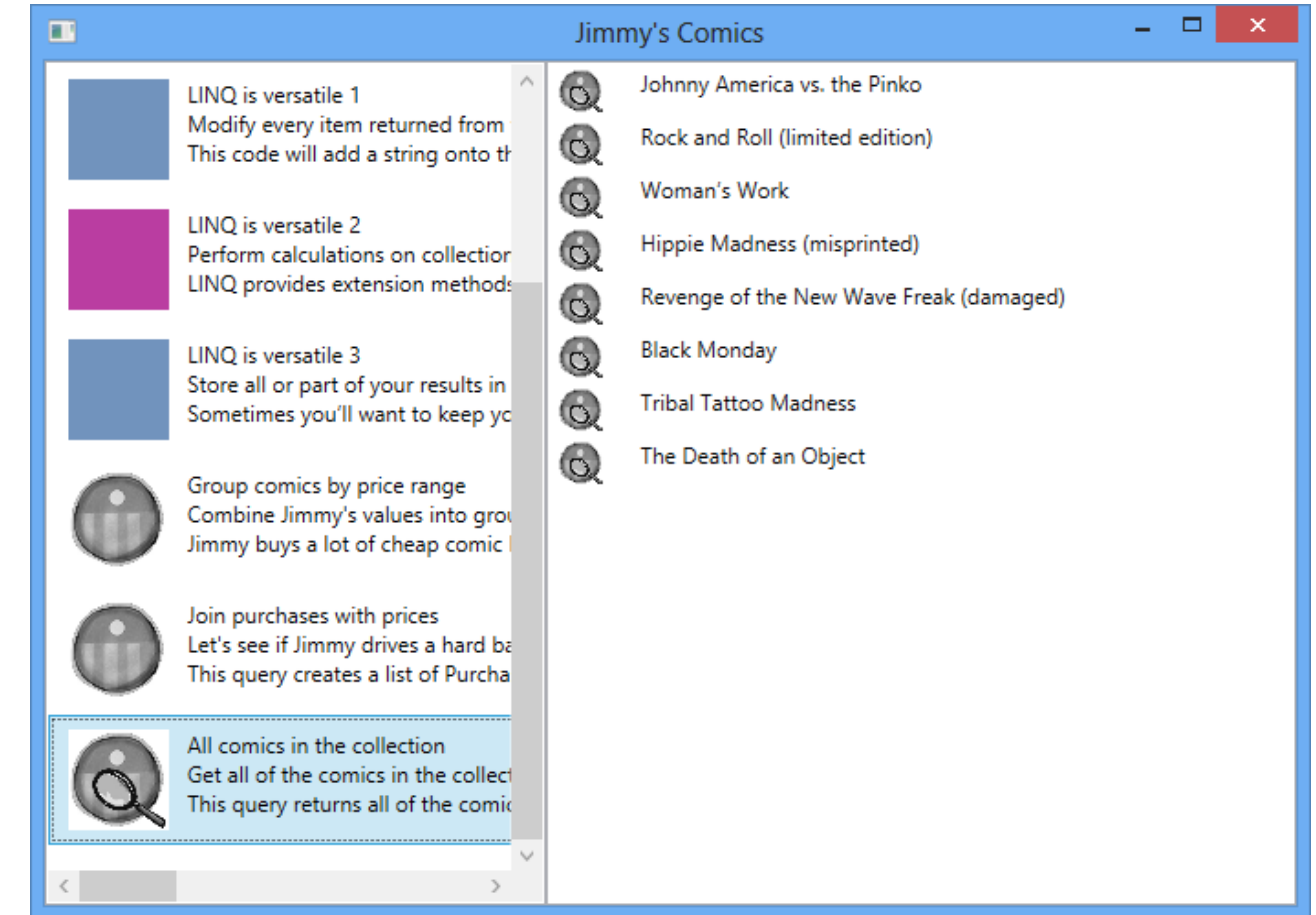
```
<Window x:Class="JimmysComics.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="clr-namespace:JimmysComics"
  Title="Jimmy's Comics" Height="350" Width="525">
  <Window.Resources>
    <local:ComicQueryManager x:Key="comicQueryManager"/>
  </Window.Resources>
  <Grid DataContext="{StaticResource ResourceKey=comicQueryManager}"
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="2*" />
      <ColumnDefinition Width="3*" />
    </Grid.ColumnDefinitions>
    <ListView SelectionMode="Single" ItemsSource="{Binding AvailableQueries}"
      SelectionChanged="ListView_SelectionChanged">
      <ListView.ItemTemplate>
        <DataTemplate>
          <Grid Height="55" Margin="6">
            <Grid.ColumnDefinitions>
              <ColumnDefinition Width="Auto" />
              <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <Border Width="55" Height="55">
              <Image Source="{Binding Image}" Stretch="UniformToFill" />
            </Border>
            <StackPanel Grid.Column="1" VerticalAlignment="Top" Margin="10,0,0,0">
              <TextBlock Text="{Binding Title}" TextWrapping="NoWrap" />
              <TextBlock Text="{Binding Subtitle}" TextWrapping="NoWrap" />
              <TextBlock Text="{Binding Description}" TextWrapping="NoWrap" />
            </StackPanel>
          </Grid>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
    <ListView Grid.Column="1" SelectionMode="Single"
      ItemsSource="{Binding CurrentQueryResults}">
      <ListView.ItemTemplate>
        <DataTemplate>
          <StackPanel Orientation="Horizontal">
            <Image Source="{Binding Image}" Margin="0,0,20,0"
              Stretch="UniformToFill" Width="25" Height="25"
              VerticalAlignment="Top" HorizontalAlignment="Right" />
            <StackPanel>
              <TextBlock Text="{Binding Title}" />
            </StackPanel>
          </StackPanel>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </Grid>
</Window>
```

ListView의 SelectionMode는 한번에 하나의 쿼리만 선택할 수 있는 Single로 설정됩니다.

왼쪽에 있는 ListView는 각 쿼리에 대한 정보를 표시하는 항목 템플릿 (ItemTemplate) 이 있습니다.

오른쪽에 있는 ListView는 쿼리 결과의 개별 항목을 보여 주는 항목 템플릿 (ItemTemplate) 이 있습니다.

앱을 실행하면 쿼리는 왼쪽에 표시됩니다. 그리고 왼쪽에서 선택한 항목의 쿼리 결과는 오른쪽에 표시됩니다.



만화책 정보를 반환하는 쿼리는 추가적인 정보(가격, 개요, 표지 이미지)가 있습니다. 여러분은 각 만화책이 추가적인 정보를 표시하는 쿼리를 작성할 수 있나요? 책 733, 734쪽의 XAML 코드를 보면 도움이 됩니다.

부록에서 두 페이지 모드를 유지하기 위해서 일부러 페이지를 비웠습니다. 연습문제와 연습문제 정답이 서로 마주하는 면에 나오면 안 되기 때문이죠. 그리고 두 페이지 모드의 짝수 쪽에는 오른쪽 위에, 홀수 쪽에는 왼쪽 위에 작은 캡션이 있습니다.

Chapter 15

이번 장의 책에서
윈도우 스토어 앱에 대한 몇몇 페이지가 있
습니다. 아무튼 이 페이지들을 읽어야 해요!



이벤트는 모든 앱에 유용합니다. 그리고 XAML을 이해하는 것도 중요합니다.

이 책의 전반에 걸쳐서 사용되는 이벤트는 단순하고 직관적일 수 있습니다. 여러분의 예상과는 달리 더 깊은 이벤트의 세계가 있습니다. 이번 장을 통해서 이벤트를 자세히 이해해 봅시다.

이번 장에서 참고할 부분입니다

- ★ 책 755쪽까지 읽어 주세요.
- ★ 이 부록은 책의 연습문제 756, 757쪽과 정답 758, 759쪽을 대체합니다.
- ★ 책 760-763쪽을 읽어 주세요.
- ★ 책 764-767쪽은 윈도우 스토어 앱에 대한 내용이지만, 이 페이지를 읽는 것을 추천합니다. 이것은 윈도우 스토어 앱뿐만 아니라, 윈도우 8의 기초적인 특징을 파악할 수 있습니다.
- ★ 부록은 책 768-773쪽을 대체합니다.
- ★ 책 나머지 부분을 읽어 주세요. 그리고 784쪽의 윗부분과 786, 787쪽은 넘어가주세요.



지금까지 배운 내용을 실습할 때가 되었습니다. Ball과 Pitcher 클래스를 완성하고, Fan 클래스를 추가해서 야구 게임 시뮬레이터의 기본 버전이 제대로 동작하는지 확인하세요.

1 Pitcher 클래스를 완성합니다.

아래에 Pitcher 클래스에 대한 코드가 나와 있습니다. CatchBall()과 CoverFirstBase() 메서드를 추가하세요. 이 두 메서드는 포수가 공을 잡거나 1루로 공을 송구한 경우 내용을 출력해야 합니다. 그리고 PitcherSays라 불리는 public ObservableCollection<string> 컬렉션에 해당 문자열을 추가합니다.

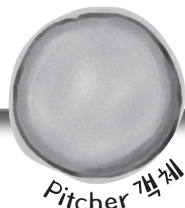
```

class Pitcher {
    public Pitcher(Ball ball) {
        ball.BallInPlay += new EventHandler(ball_BallInPlay);
    }

    void ball_BallInPlay(object sender, EventArgs e) {
        if (e is BallEventArgs){
            BallEventArgs ballEventArgs = e as BallEventArgs;
            if ((ballEventArgs.Distance < 95) && (ballEventArgs.Trajectory < 60))
                CatchBall();
            else
                CoverFirstBase();
        }
    }
}

```

PitcherSays의 ObservableCollection에 문자열을 추가하려면 두 개의 메서드가 필요합니다.



2 Fan 클래스를 작성합니다.

Fan 클래스를 생성합니다. 이 클래스 또한 자신의 생성자에서 BallInPlay 이벤트를 구독해야 합니다. 이 객체의 이벤트 핸들러는 거리가 400피트보다 크고 궤도가 30도보다 큰지(홈런일 경우) 검사해서 이 조건에 맞으면 글러브로 공을 잡으려고 시도합니다. 조건에 맞지 않을 경우 팬은 울상을 짓는군요. 팬의 비명과 고함을 ObservableCollection<string> FanSays에 추가하기만 하면 됩니다.

정확히 어떤 내용을 출력해야 하는지는 다음 페이지에 있는 결과를 참조하세요.



3 아주 간단한 시뮬레이터를 만듭니다.

아직 시뮬레이터를 만들지 않았다면, 새로운 WPF 응용 프로그램을 생성합니다. 그리고 BaseballSimulator 클래스를 추가합니다. 그리고 나서 윈도우에 정적 리소스를 추가합니다.

```

using System.Collections.ObjectModel;

class BaseballSimulator {
    private Ball ball = new Ball();
    private Pitcher pitcher;
    private Fan fan;

    public ObservableCollection<string> FanSays { get { return fan.FanSays; } }
    public ObservableCollection<string> PitcherSays { get { return pitcher.PitcherSays; } }
    public int Trajectory { get; set; }
    public int Distance { get; set; }
    public BaseballSimulator() {
        pitcher = new Pitcher(ball);
        fan = new Fan(ball);
    }

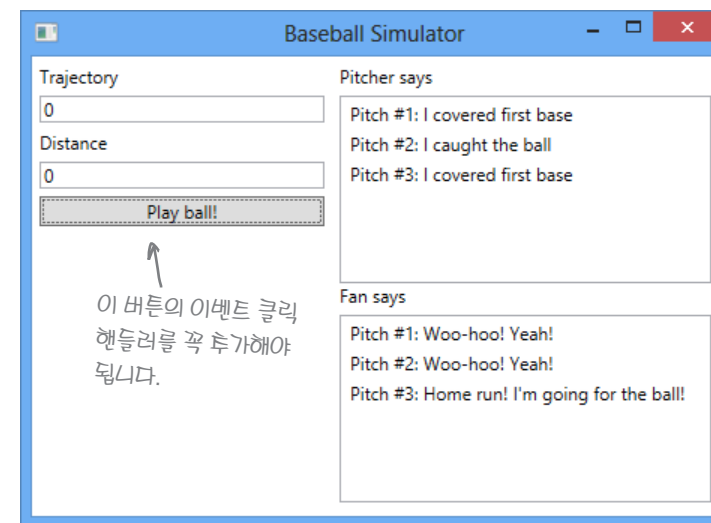
    public void PlayBall() {
        BallEventArgs ballEventArgs = new BallEventArgs(Trajectory, Distance);
        ball.OnBallInPlay(ballEventArgs);
    }
}

```

4 메인 윈도우를 만듭니다.

오른쪽에 있는 화면을 보고 똑같은 XAML을 만들 수 있나요? 두 개의 TextBox 컨트롤은 정적 리소스인 BaseballSimulator의 Trajectory(각도)와 Distance(거리) 속성에 바인딩되어 있습니다. 그리고 Pitcher와 Fan Says는 두 ObservableCollection에 바인딩된 ListView 컨트롤입니다.

세 개의 공이 성공적으로 날아갔다고 가정하고 여러분이 만든 시뮬레이터가 아래와 같은 결과를 출력하는지 확인하세요. 아래와 같은 결과를 얻으려면 어떤 값을 사용해야 하는지도 하단에 적어보세요.



이 버튼의 이벤트 클릭 핸들러를 꼭 추가해야 합니다.

공 1:	공 2:	공 3:
각도:	각도:	각도:
거리:	거리:	거리:



Ball과 BallEventArgs 클래스와 새 Fan 클래스에 추가할 내용입니다.

```

class Ball {
    public event EventHandler BallInPlay;
    public void OnBallInPlay(BallEventArgs e) {
        EventHandler ballInPlay = BallInPlay;
        if (ballInPlay != null)
            ballInPlay(this, e);
    }
}

class BallEventArgs : EventArgs {
    public int Trajectory { get; private set; }
    public int Distance { get; private set; }
    public BallEventArgs(int trajectory, int distance) {
        this.Trajectory = trajectory;
        this.Distance = distance;
    }
}

using System.Collections.ObjectModel;
class Fan {
    public ObservableCollection<string> FanSays = new ObservableCollection<string>();
    private int pitchNumber = 0;

    public Fan(Ball ball) {
        ball.BallInPlay += new EventHandler(ball_BallInPlay);
    }

    void ball_BallInPlay(object sender, EventArgs e) {
        pitchNumber++;
        if (e is BallEventArgs) {
            BallEventArgs ballEventArgs = e as BallEventArgs;
            if (ballEventArgs.Distance > 400 && ballEventArgs.Trajectory > 30)
                FanSays.Add("Pitch #" + pitchNumber
                    + ": Home run! I'm going for the ball!");
            else
                FanSays.Add("Pitch #" + pitchNumber + ": Woo-hoo! Yeah!");
        }
    }
}

```

이벤트 핸들러들은 오직 전달된 데이터를 읽기만 하므로 읽기 전용 자동 속성들은 이벤트 인수에서 실제로 잘 동작합니다.

OnBallInPlay() 메서드가 BallInPlay 이벤트만 발생시키고 있는데, null이 아닌지 검사해야 합니다. 만약 그렇게 하지 않으면 예외가 발생합니다.

Fan 객체의 생성자는 자신의 이벤트 핸들러를 BallInPlay 이벤트와 연결하고 있습니다.

Fan 객체의 BallInPlay 이벤트 핸들러는 공의 높이와 날아간 거리를 검사하는군요.

윈도우에 대한 코드-비하인드입니다.

```

public partial class MainWindow : Window {
    BaseballSimulator baseballSimulator;

    public MainWindow() {
        InitializeComponent();
        baseballSimulator = FindResource("baseballSimulator") as BaseballSimulator;
    }

    private void Button_Click(object sender, RoutedEventArgs e) {
        baseballSimulator.PlayBall();
    }
}

```

여기에 윈도우에 대한 XAML 코드가 있군요. 여기에 <local:BaseballSimulator x:Name="baseballSimulator"/>가 필요합니다.

```

<Window.Resources>
    <local:BaseballSimulator x:Key="baseballSimulator"/>
</Window.Resources>

<Grid Margin="5" DataContext="{StaticResource ResourceKey=baseballSimulator}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="200" />
    </Grid.ColumnDefinitions>
    <StackPanel Margin="0,0,10,0">
        <TextBlock Text="Trajectory" Margin="0,0,0,5"/>
        <TextBox Text="{Binding Trajectory, Mode=TwoWay}" Margin="0,0,0,5"/>
        <TextBlock Text="Distance" Margin="0,0,0,5"/>
        <TextBox Text="{Binding Distance, Mode=TwoWay}" Margin="0,0,0,5"/>
        <Button Content="Play ball!" Click="Button_Click"/>
    </StackPanel>
    <StackPanel Grid.Column="1">
        <TextBlock Text="Pitcher says" Margin="0,0,0,5"/>
        <ListView ItemsSource="{Binding PitcherSays}" Height="125"/>
        <TextBlock Text="Fan says" Margin="0,0,0,5"/>
        <ListView ItemsSource="{Binding FanSays}" Height="125"/>
    </StackPanel>
</Grid>

```

<Window> 태그에 xmlns:local 속성을 꼭 추가하세요.

그리고 여기에 Pitcher 클래스가 있군요(맨 위에 using System.Collections.ObjectModel;을 선언해야 합니다).

```

class Pitcher {
    public ObservableCollection<string> PitcherSays = new ObservableCollection<string>();
    private int pitchNumber = 0;

    public Pitcher(Ball ball) {
        ball.BallInPlay += ball_BallInPlay;
    }

    void ball_BallInPlay(object sender, EventArgs e) {
        pitchNumber++;
        if (e is BallEventArgs) {
            BallEventArgs ballEventArgs = e as BallEventArgs;
            if ((ballEventArgs.Distance < 95) && (ballEventArgs.Trajectory < 60))
                CatchBall();
            else
                CoverFirstBase();
        }
    }

    private void CatchBall() {
        PitcherSays.Add("Pitch #" + pitchNumber + ": I caught the ball");
    }

    private void CoverFirstBase() {
        PitcherSays.Add("Pitch #" + pitchNumber + ": I covered first base");
    }
}

```

조금 전에 Pitcher의 BallInPlay 이벤트 핸들러를 봤었죠. 낮게 오는 공을 검사합니다.

결과를 얻기 위해 사용한 값들이 여기 나와 있군요. 여러분의 값이 여기 있는 것과 약간 다를 수도 있습니다.

공 1:	공 2:	공 3:
각도: 75	각도: 48	각도: 40
거리: 105	거리: 80	거리: 435

XAML 컨트롤은 라우트된 이벤트를 사용합니다

책 766쪽으로 돌아가서 여러분이 override를 입력했을 때 뜨는 인텔리센스 창의 팝업을 살펴봅시다. 그렇죠, 윈도우 스토어 앱에 대한 겁니다. 하지만 원리는 WPF에서도 똑같이 적용됩니다. DoubleTapped 이벤트의 두 번째 인수는 DoubleTappedRouteEventArgs 유형을 갖고 있고, GotFocus 이벤트는 RoutedEventArgs 유형을 갖고 있습니다. 두 이벤트 인수 유형의 이름이 다른 것들과 조금 구별됩니다. 그 이유는 DoubleTapped와 GotFocus는 **라우트된 이벤트 (routed event)**기 때문이죠. 이들은 다른 두 가지를 제외하고는 보통의 이벤트와 같습니다. 컨트롤 객체가 라우트된 이벤트에 응답할 때, 먼저 평소와 같이 이벤트 핸들러 메서드를 호출합니다. 그리고 다른 작업을 수행합니다. 만약 이벤트가 처리되지 않은 경우, **라우트된 이벤트를 컨트롤의 컨테이너에 보내줍니다.** 이 컨테이너는 이벤트를 발생시키죠. 그리고 나서 이벤트가 처리되지 않는다면, 라우트된 이벤트를 상위 컨테이너에게 또 보내줍니다. 이벤트가 처리되거나 루트 (root) 혹은 최상위 컨테이너로 도달할 때까지, 이벤트는 **버블링(bubbling)**을 유지합니다. 아래에 전형적인 라우트된 이벤트 핸들러 메서드의 시그네처가 있습니다.

```
private void EventHandler(object sender, RoutedEventArgs e)
```

RoutedEventArgs 객체는 Handled 속성이 있습니다. 이벤트 핸들러는 이벤트를 처리하는 것을 알리기 위해서 이 속성을 사용하죠. 이 속성을 true로 설정하면 **이벤트가 버블링되지 않습니다.**

라우트된 이벤트와 표준 이벤트에서, sender 매개변수는 항상 이벤트 핸들러를 호출하는 객체의 참조를 포함하고 있습니다. 그래서 만약 이벤트가 컨트롤에서 Grid와 같은 컨테이너로 버블링된다면, Grid는 이벤트 핸들러를 호출할 때 sender는 Grid 컨트롤의 참조가 됩니다. 하지만 여러분이 어떤 컨트롤에서 본 이벤트를 발생시키는지 찾아야 한다면 무엇을 해야 할까요? RoutedEventArgs 객체는 컨트롤이 처음 발생한 이벤트의 참조를 포함하는 **OriginalSource**라 불리는 속성이 있습니다. 만약 OriginalSource와 sender가 같은 객체를 가리킨다면 이벤트 핸들러를 호출한 컨트롤은 이벤트가 발생되고 버블링이 시작된 같은 컨트롤이라는 것을 의미합니다.

마우스와 포인터에서 어느 한 요소를 "볼 수" 있는 것은 IsHitTestVisible이 결정합니다

일반적으로 페이지에 있는 모든 요소는 특정 영역의 범위를 벗어나지 않는 한, 포인터나 마우스에 의해서 "선택"될 수 있습니다. 요소들이 보여야 하고(Visibility 속성으로 바꿀 수 있죠), Background 혹은 Fill 속성이 null이 아니어야 합니다(투명도인 Transparent는 괜찮습니다). 그리고 활성화되어 있고(IsEnabled 속성), height와 width가 0보다 커야 합니다. 만약 이 모든 것들을 만족한다면 IsHitTestVisible 속성은 **True를 반환**하고, 포인터나 마우스 이벤트에 응답하게 됩니다.

이 속성은 특히 이벤트를 마우스에 "보이지 않는"(invisible) 상태로 만들 때 유용합니다. 만약 IsHitTestVisible이 False로 설정되어 있다면, 어떤 포인터의 탭 혹은 마우스 클릭은 **컨트롤을 통해서 바로 전달**됩니다. 만약 보이지 않는 컨트롤 아래에 다른 컨트롤이 있다면, 그 컨트롤이 대신 이벤트를 얻게 되는 거죠.

라우트된 이벤트 개요에 대해서 자세히 살펴 보려면...
<http://msdn.microsoft.com/ko-kr/library/windows/apps/Hh758286.aspx>

다른 컨트롤을 포함하는 컨트롤의 구조를 객체 트리라 부릅니다. 라우트된 이벤트는 자식으로부터 부모까지 (상위 루트에 갈 때까지) 트리를 버블링합니다.

라우트된 이벤트를 살펴봅시다

다음 그림은 라우트된 이벤트를 실험하는 데 사용하는 WPF 응용 프로그램입니다. Border를 포함하는 StackPanel이 있습니다. 이것은 Grid와 그 안에 사각형의 Rectangle과 원의 Ellipse가 있죠. 그림을 한 번 살펴보세요. Rectangle이 Ellipse 위에 있는 걸 어떻게 볼 수 있죠? 만약 두 개의 컨트롤이 같은 셀에 놓여 있다면, 각 두 컨트롤의 위에 또 다른 컨트롤을 놓을 수 있습니다. 그리고 두 컨트롤 모두 같은 부모를 가지죠. Grid의 부모인 Border, Border의 부모인 StackPanel을 가집니다. Rectangle 혹은 Ellipse에서 라우트된 이벤트는 부모에서 **객체 트리의 루트까지** 버블링됩니다.

체크 상태를 전환할 수 있는 CheckBox 컨트롤입니다. Content 속성은 컨트롤에 대한 레이블을 설정합니다. IsChecked 속성은 Nullable<bool>입니다. 왜냐하면 체크가 된 상태와 안 된 상태 그리고 또 다른 불확정 상태를 가질 수 있기 때문이죠.

```

<Grid Margin="5">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <StackPanel x:Name="panel" MouseDown="StackPanel_MouseDown">
    <Border BorderThickness="10" BorderBrush="Blue" Width="155" x:Name="border"
      Margin="20" MouseDown="Border_MouseDown">
      <Grid x:Name="grid" MouseDown="Grid_MouseDown">
        <Ellipse Fill="Red" Width="100" Height="100"
          MouseDown="Ellipse_MouseDown"/>
        <Rectangle Fill="Gray" Width="50" Height="50"
          MouseDown="Rectangle_MouseDown" x:Name="grayRectangle"/>
      </Grid>
    </Border>
    <ListBox BorderThickness="1" Width="250" Height="140" x:Name="output" Margin="0,0,20,0"/>
  </StackPanel>
  <StackPanel Grid.Column="1">
    <CheckBox Content="Border sets handled" x:Name="borderSetsHandled"/>
    <CheckBox Content="Grid sets handled" x:Name="gridSetsHandled" />
    <CheckBox Content="Ellipse sets handled" x:Name="ellipseSetsHandled"/>
    <CheckBox Content="Rectangle sets handled" x:Name="rectangleSetsHandled"/>
    <Button Content="Update Rectangle IsHitTestVisible"
      Click="UpdateHitTestButton" Margin="0,20,20,0"/>
    <CheckBox IsChecked="True" Content="New IsHitTestVisible value"
      x:Name="newHitTestVisibleValue" />
  </StackPanel>
</Grid>
  
```

라우트된 이벤트는 객체 트리를 버블링합니다.

IsChecked의 기본값은 False입니다. 이 CheckBox는 True로 설정되어 있습니다. 기본적으로 컨트롤의 IsHitTestVisible 속성은 항상 참이기 때문이죠.

ListBox에 출력 결과를 표시하기 위해서 ObservableCollection이 필요합니다.

outputItems 필드를 만들고, 윈도우 생성자에 ListBox.ItemsSource 속성을 집어넣습니다. 그리고 ObservableCollection<T>를 위해서 using System.Collections.ObjectModel; 문장을 추가하는 것도 잊지 마세요.

```
public partial class MainWindow : Window {
    ObservableCollection<string> outputItems = new ObservableCollection<string>();

    public MainWindow() {
        this.InitializeComponent();

        output.ItemsSource = outputItems;
    }
}
```

여기에 코드-비하인드가 있습니다. 각 컨트롤의 MouseDown 이벤트 핸들러는 출력 결과를 초기화합니다. 그리고 나서 출력 결과에 문자열을 추가합니다. 만약 "handled"를 체크 상태로 전환한다면, 이벤트를 처리하기 위해서 e.Handled를 사용합니다.

```
private void Ellipse_MouseDown(object sender, MouseButtonEventArgs e) {
    if (sender == e.OriginalSource) outputItems.Clear();
    outputItems.Add("The ellipse was pressed");
    if (ellipseSetsHandled.IsChecked == true) e.Handled = true;
}

private void Rectangle_MouseDown(object sender, MouseButtonEventArgs e) {
    if (sender == e.OriginalSource) outputItems.Clear();
    outputItems.Add("The rectangle was pressed");
    if (rectangleSetsHandled.IsChecked == true) e.Handled = true;
}

private void Grid_MouseDown(object sender, MouseButtonEventArgs e) {
    if (sender == e.OriginalSource) outputItems.Clear();
    outputItems.Add("The grid was pressed");
    if (gridSetsHandled.IsChecked == true) e.Handled = true;
}

private void Border_MouseDown(object sender, MouseButtonEventArgs e) {
    if (sender == e.OriginalSource) outputItems.Clear();
    outputItems.Add("The border was pressed");
    if (borderSetsHandled.IsChecked == true) e.Handled = true;
}

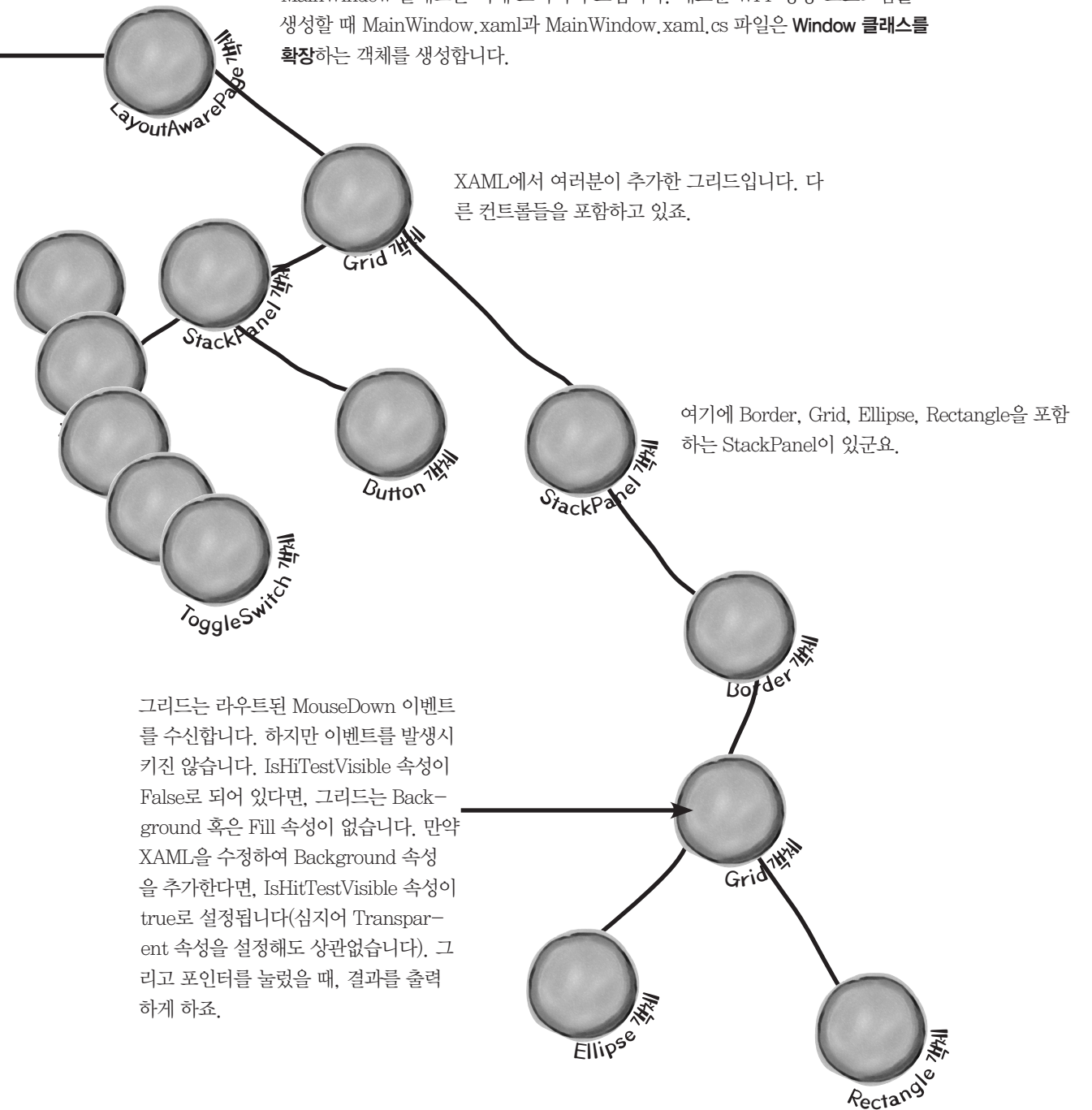
private void StackPanel_MouseDown(object sender, MouseButtonEventArgs e) {
    if (sender == e.OriginalSource) outputItems.Clear();
    outputItems.Add("The panel was pressed");
}

private void UpdateHitTestButton(object sender, RoutedEventArgs e) {
    grayRectangle.IsHitTestVisible = (bool)newHitTestVisibleValue.IsChecked;
}
```

버튼의 Click 이벤트는 토글을 전환하는 IsChecked 속성을 이용해서 Rectangle 컨트롤의 IsHitTestVisible을 true/false로 설정합니다.

메인 윈도우의 객체 그래프입니다.

MainWindow 클래스는 객체 트리의 루트입니다. 새로운 WPF 응용 프로그램을 생성할 때 MainWindow.xaml과 MainWindow.xaml.cs 파일은 Window 클래스를 확장하는 객체를 생성합니다.



XAML에서 여러분이 추가한 그리드입니다. 다른 컨트롤들을 포함하고 있죠.

여기에 Border, Grid, Ellipse, Rectangle을 포함하는 StackPanel이 있군요.

그리드는 라우트된 MouseDown 이벤트를 수신합니다. 하지만 이벤트를 발생시키진 않습니다. IsHitTestVisible 속성이 False로 되어 있다면, 그리드는 Background 혹은 Fill 속성이 없습니다. 만약 XAML을 수정하여 Background 속성을 추가한다면, IsHitTestVisible 속성이 true로 설정됩니다(심지어 Transparent 속성을 설정해도 상관없습니다). 그리고 포인터를 눌렀을 때, 결과를 출력하게 하죠.

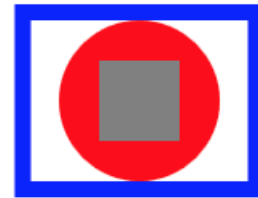
머릿속에 버블링이 바로 전달됩니다

앱을 실행하고 회색 사각형을 탭 하거나 눌러봅시다.

오른쪽 화면과 같은 출력 결과를 볼 수 있습니다. →

Rectangle 컨트롤의 PointerPressed 이벤트 핸들러인 Rectangle_MouseDown()의 첫 번째 줄에 중단점을 놓고, 디버깅하면 무슨 일어나는지 알 수 있습니다.

```
private void Rectangle_PointerPressed(object sender, PointerRoutedEventArgs e)
{
    if (sender == e.OriginalSource) outputItems.Clear();
    outputItems.Add("The rectangle was pressed");
    if (rectangleSetsHandled.IsOn) e.Handled = true;
}
```



The rectangle was pressed
The grid was pressed
The border was pressed
The panel was pressed

회색 사각형을 다시 한번 클릭해 보세요. 이번에는 중단점이 실행될 겁니다. 프로시저 단위 실행(F10)을 이용해서 코드 한 줄씩 실행해 보세요. 먼저 if문에서 ListBox에 바인딩된 ObservableCollection의 outputItems를 초기화합니다. sender와 e.OriginalSource가 같은 Rectangle 컨트롤을 참조하기 때문에 이 코드가 실행되죠. 그래서 이벤트가 발생한 컨트롤(클릭 혹은 탭 할 경우)에 대한 이벤트 핸들러 메서드의 sender == e.OriginalSource는 true입니다.

메서드의 끝에 도달할 때, 프로시저 단위 실행을 유지하세요. 이벤트는 객체 트리를 통해 버블링합니다. 먼저 Rectangle의 이벤트 핸들러에 버블링되고 난 후, Grid와 Border, Panel의 이벤트 핸들러 순으로 버블링합니다. 그리고 마침내 Window의 일부 이벤트 핸들러 메서드를 실행합니다(여러분의 코드를 벗어났고, 라우트된 이벤트의 일부가 아니기 때문에 항상 실행됩니다). 그리고 컨트롤 중에서 이벤트의 원본 출처가 없기 때문에, 이벤트 핸들러의 senders는 e.OriginalSource와 일치하지 않습니다. 그래서 이들 중 아무것도 출력 결과를 초기화하지 않습니다.

IsHitTestVisible를 체크하지 않은 상태로 두고, "Update" 버튼을 눌러서 회색 사각형을 클릭하거나 탭 해 봅시다.



← 출력 결과가 이렇게 나와야 합니다.

The ellipse was pressed
The grid was pressed
The border was pressed
The panel was pressed

잠깐만요! Rectangle을 눌렀는데 Ellipse 컨트롤의 MouseDown 이벤트 핸들러가 발생했습니다. 무슨 일이 일어난 걸까요?

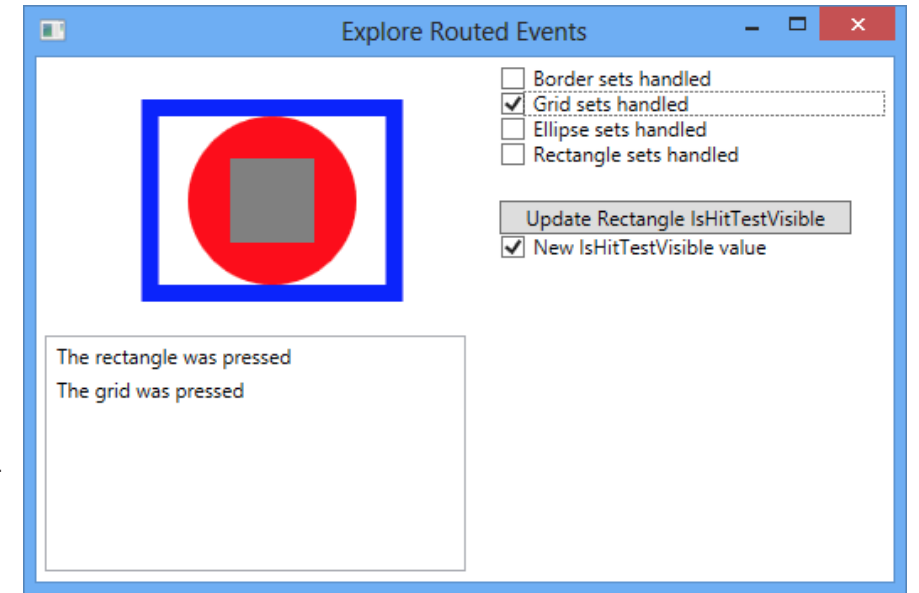
버튼을 눌렀을 때, Click 이벤트 핸들러는 Rectangle 컨트롤의 IsHitTestVisible 속성을 false로 갱신합니다. 이것은 포인터를 누르거나 클릭 혹은 다른 포인터 이벤트에서 컨트롤을 보이지 않게 합니다(invisible). 그래서 Rectangle을 클릭했을 때, 이 이벤트는 Rectangle 아래에 있는 Ellipse가 페

이지 아래의 최상위 컨트롤에 전달됩니다. 페이지의 Ellipse는 IsHitTestVisible 속성이 true로 설정되고, Color 혹은 Transparent가 설정된 Background 속성을 가집니다. 이 경우에는 Ellipse 컨트롤을 발견하고, 컨트롤의 MouseDown 이벤트를 발생시킵니다.

"Grid sets handled"를 체크하고, 회색 사각형을 클릭하거나 탭 해 봅시다.

출력 결과가 이렇게 나와야 합니다. →

왜 리스트박스에 출력 결과가 두 줄밖에 나오지 않을까요? 프로시저 단위 실행을 통해 무슨 일이 일어나는지 살펴봅시다. 이번에는 여러분이 gridSetsHandled를 체크했기 때문에, gridSetHandled, IsChecked가 true로 설정돼 있습니다. 그래서 Grid의 이벤트 핸들러의 마지막 줄에서 e.Handled가 true로 설정됩니다. 라우트된 이벤트 핸들러가 버블링을 시작할 때, 이벤트는 버블링을 중단합니다. Grid의 이벤트 핸들러의 수행을 끝내자마자, 앱은 이벤트가 처리된 걸 보고서는 Border 혹은 Panel의 이벤트 핸들러 메서드를 호출하지 않습니다. 대신 코드 외부에 추가한 Window의 이벤트 핸들러로 건너뛩니다.



앱을 이용해서 라우트된 이벤트를 실험해 봅시다.

아래에 해야 할 일들이 있습니다.

- ★ 회색 사각형과 빨간색 타원을 클릭하고 이벤트가 어떻게 버블링되는지 출력 결과를 살펴봅시다.
- ★ 이벤트 핸들러가 e.Handled를 true로 설정하도록, 위에서부터 각각의 토글을 전환해 봅시다. 그리고 버블링이 중단된 결과를 살펴봅시다.
- ★ 모든 이벤트 핸들러 메서드에 중단점을 설정해서 디버깅해 봅시다.
- ★ Ellipse의 이벤트 핸들러 메서드에 중단점을 놓고, Rectangle의 IsHitTestVisible 속성을 체크/해제해 봅시다. 아래의 토글로 전환하고, "Update" 버튼을 누릅니다. 프로시저 단위 실행을 통해서 언제 IsHitTestVisible이 false로 되는지 살펴봅시다.
- ★ 프로그램을 멈추고, Background 속성을 Grid에 추가해서 포인터가 도달하는 것을 볼 수 있게 해 봅시다.

라우트된 이벤트는 먼저 컨트롤에서 발생된 이벤트의 이벤트 핸들러를 발생시킵니다. 그리고 컨트롤의 계층구조에 따라 최상위 컨트롤에 도달하거나 이벤트 핸들러가 e.Handled를 true로 설정한 곳에 도달할 때까지 버블링합니다.

부록에서 두 페이지 모드를 유지하기 위해서 일부러 페이지를 비웠습니다. 연습문제와 연습문제 정답이 서로 마주하는 면에 나오면 안되기 때문이죠. 그리고 두 페이지 모드의 짝수 쪽에는 오른쪽 위에, 홀수 쪽에는 왼쪽 위에 작은 캠페인이 있습니다.

Chapter 16

MVVM(Model-View-ViewModel) 패턴을 사용하여 응용 프로그램을 만들 때 앞으로 여러분이 코드를 더 쉽게 관리할 수 있습니다.

숙련된 개발자는 디자인 패턴을 사용합니다.

이번 장에서는 효과적인 WPF 응용 프로그램을 만들기 위해서 MVVM(Model-View-ViewModel) 디자인 패턴에 대해 알아보겠습니다. 디자인 패턴이 무엇인지 그리고 애니메이션을 위해서 XAML 컨트롤을 어떻게 사용하는지 배워 봅시다.

16장에서 참고할 사항입니다.

- ★ 책 793쪽까지 읽어 주세요.
- ★ 이 부록은 책 794-801쪽을 대체합니다.
- ★ 책 802-808쪽을 읽어 주세요.
- ★ 책 806쪽에서 스톱워치 프로젝트를 시작하세요. 계속 읽다가 809, 812, 814-817, 825-831쪽이 나오면 이 부분은 부록을 참고해 주세요.
- ★ 책 832쪽을 읽습니다.
- ★ 이 부록은 책 833-851쪽을 대체합니다.
- ★ 850쪽에 실습#3에 대한 정보가 있습니다.



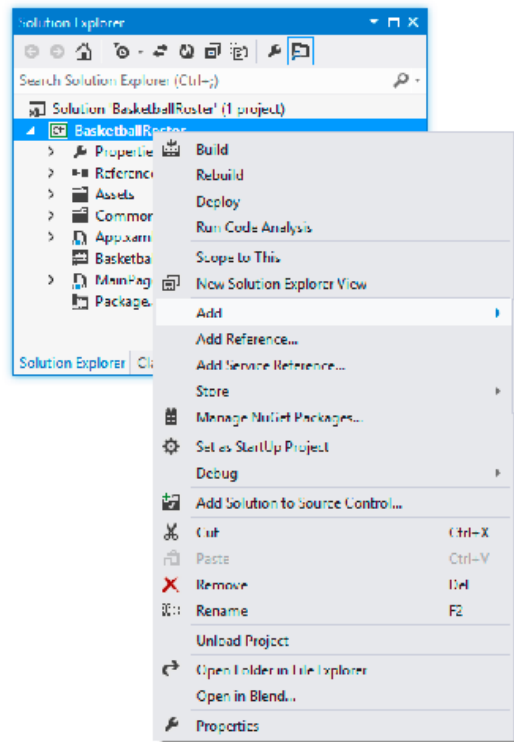
MVVM 패턴으로 농구선발명단 앱을 만들어 봅시다

새 윈도우 스토어 앱을 생성하고, 이름은 농구선발명단인 BasketballRoster로 합니다(이 코드에서 BasketballRoster 네임스페이스를 사용하기 때문이죠. 여러분의 코드와 다음에 나오는 몇 페이지의 코드가 같아야 합니다).

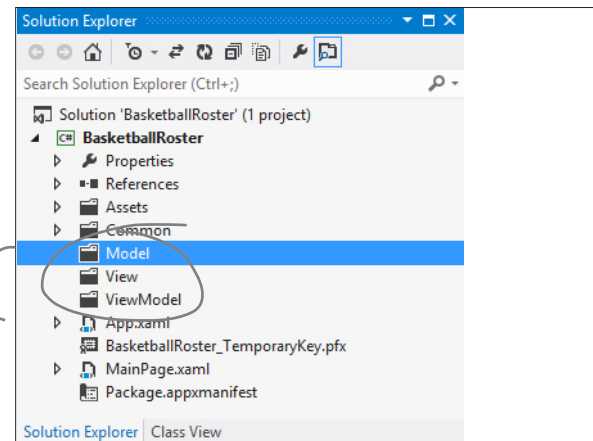
직접 해
봅시다!

1 프로젝트에 모델, 뷰, 뷰모델 폴더를 생성합니다.

프로젝트의 솔루션 탐색기에서 마우스 오른쪽 버튼을 클릭하고, 추가 > 새 폴더를 선택합니다.



솔루션 탐색기에서 프로젝트에 새 폴더를 추가할 때, IDE는 폴더 이름을 기반으로 한 새로운 네임스페이스를 생성합니다. 이것은 추가 > 클래스 메뉴 옵션에서 생성한 클래스들의 네임스페이스가 됩니다. 만약 Model 폴더에 클래스를 추가한다면, IDE는 클래스 파일의 꼭대기에 있는 네임스페이스 라인에서 BasketballRoster.Model을 추가합니다.



Model, View, ViewModel 폴더를 추가합니다. 솔루션 탐색기는 오른쪽 사진과 같이 보입니다.

이 폴더에 앱의 클래스와 컨트롤, 윈도우를 담을 거예요.

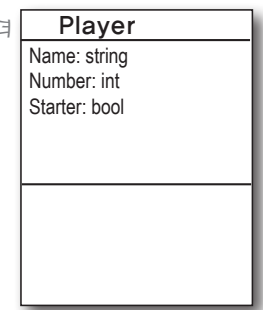


2 Player 클래스를 추가해서 모델을 만들어 봅시다.

모델 폴더를 오른쪽 클릭해서 Player 클래스를 추가합니다. 폴더에 클래스를 추가하면, IDE는 끝으로 폴더 이름의 네임스페이스를 업데이트합니다. 여기에 Player 클래스가 있습니다.

```
namespace BasketballRoster.Model {
    class Player {
        public string Name { get; private set; }
        public int Number { get; private set; }
        public bool Starter { get; private set; }

        public Player(string name, int number, bool starter) {
            Name = name;
            Number = number;
            Starter = starter;
        }
    }
}
```



폴더에 클래스 파일을 추가할 때, IDE는 폴더 이름으로 된 네임스페이스를 추가합니다.

왜 클래스들은 서로 다른 것들에만 관심을 보일까요? 비유해 보기도 한테 말이죠.

이 두 클래스는 선수가 각 명부에 있는 데이터를 파악하는 데만 관심이 있기 때문에 클래스가 간단합니다. 모델에 있는 클래스는 데이터를 보여 주는 데 관심이 없죠. 그저 데이터를 관리할 뿐입니다.

3 마지막으로 Roster 클래스를 Model에 추가합니다.

다음으로 Model 폴더에 Roster 클래스를 추가합니다. 아래에 코드가 있군요.

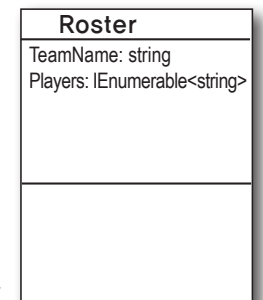
```
namespace BasketballRoster.Model {
    class Roster {
        public string TeamName { get; private set; }

        private readonly List<Player> _players = new List<Player>();
        public IEnumerable<Player> Players {
            get { return new List<Player>(_players); }
        }

        public Roster(string teamName, IEnumerable<Player> players) {
            TeamName = teamName;
            _players.AddRange(players);
        }
    }
}
```

는 private 필드를 의미합니다.

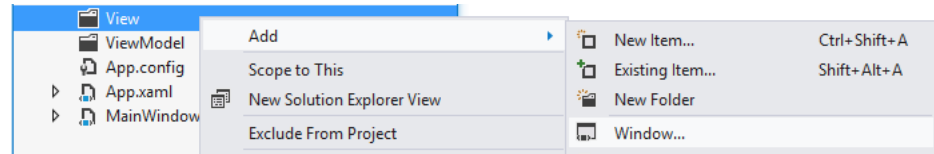
players 필드의 이름에 언더스코어()를 추가했습니다. 언더스코어를 private 필드 앞에 추가하는 것은 일반적인 네이밍 컨벤션입니다. 이번 장에서 이러한 규칙이 사용되는 것을 볼 수 있습니다.



뷰는 다음 페이지에

4 View 폴더에 메인 윈도우를 추가합니다.

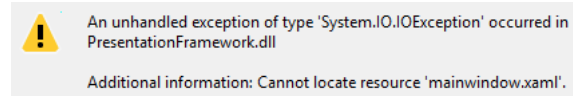
View 폴더에 오른쪽 클릭을 한 후, 새 LeagueWindow.xaml을 추가합니다.



View 폴더에 XAML 윈도우인 LeagueWindow.xaml이 있습니다. 이것은 부록 전반에서 본 MainWindow.xaml 윈도우와 같은 동작을 하죠, 그리고 여전히 XAML로 정의된 그래프의 Window 객체입니다. 다른 점이 있다면 MainWindow를 대신 LeagueWindow로 부르는 겁니다.

5 메인 윈도우를 지우고, 이것을 새 윈도우로 교체합니다.

솔루션 탐색기에서 MainWindow.xaml 파일을 선택하고, 마우스 오른쪽 클릭 > 삭제를 선택합니다. 한번 프로젝트를 빌드해서 실행해 보세요. 프로그램이 시작될 때, 아래와 같은 예외를 얻게 됩니다.



글쎄요, MainWindow.xaml을 지웠으니 예외가 나오는건 이해가 되네요. WPF 응용 프로그램이 시작될 때, App.xaml 파일의 <Application> 태그에 있는 StartupUri 속성에서 지정한 윈도우를 보여 줍니다.

```
<Application x:Class="BasketballRoster.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  StartupUri="MainWindow.xaml">
  <Application.Resources>
  </Application.Resources>
</Application>
```

App.xaml 파일을 열고 StartupUri 속성을 수정합니다. 입력하는 도중 인텔리센스 창에서 여러분이 추가한 윈도우를 볼 수 있습니다.

```
<Application x:Class="BasketballRoster.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  StartupUri="View/LeagueWindow.xaml">
```

위와 같이 수정하고 난 뒤, 프로그램을 다시 빌드해서 실행해 보세요. 이제 프로그램이 실행되고, 새롭게 추가된 창이 보일 겁니다.

사용자 정의 컨트롤

잠시 농구선수명단 프로그램을 살펴보세요. 각 팀은 같은 세트의 컨트롤로 구성되어 있습니다. TextBox, ListView, 또 다른 TextBox, ListView... 모든 컨트롤은 Border 내부의 StackPanel 컨트롤 안에 있습니다. 같은 컨트롤들의 두 세트를 정말로 페이지에 추가해야 할까요? 만약 세 번째 팀과 네 번째 팀을 추가해야 한다면, 중복적으로 일어나는 일에 대해서 생각해 봐야 합니다. 그리고 여기에 사용자 정의 컨트롤(User Control)이 있습니다. 사용자 정의 컨트롤은 자신만의 컨트롤을 만들기 위해서 사용되는 클래스입니다. 보통 페이지를 만들 때처럼 사용자 정의 컨트롤을 만들기 위해서 XAML과 코드-비하인드를 이용하죠. BasketballRoster 프로젝트에 사용자 정의 컨트롤을 추가해 봅시다.

1 View 폴더에 새로운 사용자 정의 컨트롤을 추가합니다.

사용자 정의 컨트롤(WPF)을 선택하고, 이름을 RosterControl.xaml로 합니다.

2 새로운 사용자 정의 컨트롤의 코드-비하인드를 살펴봅시다.

RosterControl.xaml.cs를 열면, 여러분의 새 컨트롤은 UserControl 베이스 클래스를 확장합니다. 사용자 정의 컨트롤로 정의된 모든 비하인드-코드는 아래의 코드와 같습니다.

```
namespace BasketballRoster.View
{
  /// <summary>
  /// Interaction logic for RosterControl.xaml
  /// </summary>
  public partial class RosterControl : UserControl
  {
    public RosterControl()
    {
      InitializeComponent();
    }
  }
}
```

3 새로운 사용자 정의 컨트롤의 XAML을 살펴봅시다.

IDE는 빈 <Grid>의 사용자 정의 컨트롤을 추가합니다.

페이지를 넘기기 전에 책 790쪽에서 윈도우 스토어 앱의 스크린 샷 보고, 새 RosterControl.xaml에서 무엇을 해야 할지 한번 생각해 보세요.

- ★ 파란색 <Border> 내부에 있는 컨트롤을 <StackPanel>에 담아야 합니다. Border 컨트롤에서 둥근 모서리를 만들려면 어떤 속성을 사용해야 할까요?
- ★ 선수에 대한 데이터를 표시하는 두 개의 ListView 컨트롤이 있습니다. 또한 한 DataTemplate이 포함된 <UserControl.Resources> 섹션이 필요합니다. 이 템플릿의 이름을 PlayerItemTemplate으로 합니다.
- ★ ListView 항목을 선발, 벤치 플레이어를 의미하는 Starters와 Bench 속성에 바인딩합니다. 그리고 위쪽에 있는 TextBox 컨트롤을 팀 이름의 TeamName 속성과 바인딩해줍니다.
- ★ Border 컨트롤은 <Grid> 안에 있습니다. 그리고 <Grid>에는 한 라인의 Height="Auto" 속성이 있는데, 이것은 하단 부분의 ListView 컨트롤을 페이지에 맞게 채워 줍니다.

UserControl은 여러분이 지정한 컨트롤을 캡슐화하는 방법을 제공하는 베이스 클래스입니다. 그리고 컨트롤의 동작을 정의하는 논리를 만들 수 있게 해주죠.

“남시하는 법 배우기”
점점 이 책의 끝을 향해 가고 있습니다. 우리는 여러분이 현실 세계에서 직면하게 될 문제들을 고민하고 해결하는 습관을 유도하고 있습니다. 좋은 프로그래머는 직면할 문제에 대해서 많은 추측을 합니다. 이 책에서도 UserControl이 어떻게 동작하는지에 관한 충분한 정보를 제공하고 있습니다. 아직 UserControl의 바인딩 설정을 하지 않아서, 디자이너에는 데이터가 보이지 않습니다. 여러분이 코드를 보기 위해 페이지를 넘기기전에 여러분이 할 수 있는 만큼 XAML 코드를 작성해 보세요.

4 RosterControl의 XAML 코드를 완성해 봅시다.

View 폴더에 추가한 RosterControl의 사용자 정의 컨트롤 코드가 아래에 있습니다. 데이터 컨텍스트 없이 속성에 어떻게 바인딩하는지 눈치 채셨나요? 윈도우의 두 컨트롤은 서로 다른 데이터를 보여 주기 때문에 페이지는 각 컨트롤에게 다른 데이터 컨텍스트를 설정해야 합니다.

```
<UserControl x:Class="BasketballRoster.View.RosterControl"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  mc:Ignorable="d"
  d:DesignHeight="450" d:DesignWidth="300">
  <UserControl.Resources>
    <DataTemplate x:Key="PlayerItemTemplate">
      <TextBlock>
        <Run Text="{Binding Name, Mode=OneWay}"/>
        <Run Text=" #"/>
        <Run Text="{Binding Number, Mode=OneWay}"/>
      </TextBlock>
    </DataTemplate>
  </UserControl.Resources>
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Border BorderThickness="2" BorderBrush="Blue" CornerRadius="6" Background="Black">
      <StackPanel Margin="20">
        <TextBlock Foreground="White" FontFamily="Segoe" FontSize="20px"
          FontWeight="Bold" Text="{Binding TeamName}" />
        <TextBlock Foreground="White" FontFamily="Segoe" FontSize="16px"
          Text="Starting Players" Margin="0,5,0,0"/>
        <ListView Background="Black" Foreground="White" Margin="0,5,0,0"
          ItemTemplate="{StaticResource PlayerItemTemplate}"
          ItemsSource="{Binding Starters}" />
        <TextBlock Foreground="White" FontFamily="Segoe" FontSize="16px"
          Text="Bench Players" Margin="0,5,0,0"/>
        <ListView Background="Black" Foreground="White" ItemsSource="{Binding Bench}"
          ItemTemplate="{StaticResource PlayerItemTemplate}" Margin="0,5,0,0"/>
      </StackPanel>
    </Border>
  </Grid>
</UserControl>
```

여러분은 이미 컨트롤은 Height 및 Width 속성에 따라 크기가 변경되는 것을 알고 있죠. 이 속성들을 수정할 때, 숫자를 조정해서 디자이너 창에 표시되는 컨트롤의 크기를 변경해도 됩니다.

ListView 항목에 대한 데이터 템플릿을 정적 리소스에 놓습니다. 그리고 나서 <ListView.ItemTemplate> 섹션 대신, ListView 태그의 ItemTemplate 속성에 정적 리소스를 설정합니다. ItemTemplate="{StaticResource PlayerItemTemplate}"

CornerRadius 속성을 사용하면 Border에 둥근 모서리 효과를 둘 수 있습니다.

두 ListView 컨트롤은 정적 리소스로 정의된 같은 템플릿을 사용하고 있네요.



연습문제

BasketballRoster 앱에서 모델의 데이터와 뷰의 바인딩을 살펴봅시다. 그리고 이 둘을 연결하는 "배관" 역할을 하는 뷰모델을 만들어 봅시다.



1 LeaguePage.xaml에 컨트롤을 추가합니다.

먼저 새 네임스페이스를 인식하도록 아래의 xmlns 속성을 페이지에 추가합니다.

```
xmlns:view="clr-namespace:BasketballRoster.View"
xmlns:viewmodel="clr-namespace:BasketballRoster.ViewModel"
```

그리고 나서 LeagueViewModel 인스턴스를 정적 리소스에 추가합니다.

```
<Window.Resources>
  <viewmodel:LeagueViewModel x:Key="LeagueViewModel"/>
</Window.Resources>
```

이제 두 RosterControl이 있는 StackPanel을 추가합니다.

```
<StackPanel Orientation="Horizontal" Margin="5"
  VerticalAlignment="Center" HorizontalAlignment="Center"
  DataContext="{StaticResource ResourceKey=LeagueViewModel}" >
  <view:RosterControl Width="200" DataContext="{Binding JimmysTeam}" Margin="0,0,20,0" />
  <view:RosterControl Width="200" DataContext="{Binding BriansTeam}" />
</StackPanel>
```

2 뷰모델 클래스를 추가합니다.

정확한 위치에 폴더에 클래스와 윈도우를 생성해야 합니다. 그렇지 않으면, 네임스페이스는 솔루션 탐색기에 있는 코드와 일치하지 않습니다.

ViewModel 폴더에 아래의 세 클래스를 생성해 주세요.

PlayerViewModel
Name: string
Number: int

RosterViewModel
TeamName: string
Starters: ObservableCollection <PlayerViewModel>
Bench: ObservableCollection <PlayerViewModel>
constructor: RosterViewModel(Model.Roster)
private UpdateRosters()

LeagueViewModel
JimmysTeam: RosterViewModel
BriansTeam: RosterViewModel
private GetBomberPlayers(): Model.Roster
private GetAmazinPlayers(): Model.Roster

3 뷰모델 클래스를 동작하게 합니다.

- ★ PlayerViewModel 클래스는 두 속성이 있는 간단한 데이터 객체입니다.
- ★ LeagueViewModel 클래스는 페이지에 더미 데이터를 생성하기 위한 두 private 메서드가 있습니다. 이것은 RosterViewModel 생성자에 전달받을 각 팀에 대한 Model.Roster 객체를 생성합니다.
- ★ RosterViewModel 클래스의 생성자의 매개변수는 Model.Roster 객체입니다. 이 생성자는 TeamName 속성을 설정한 다음, private UpdateRosters() 메서드를 호출합니다. 이 메서드는 선발과 벤치 플레이어를 추출하기 위해서 LINQ 쿼리를 사용하고, Starters와 Bench 속성을 업데이트합니다. 모델의 네임스페이스에서 객체를 사용하려면 클래스의 상단에 "using Model;"을 추가합니다.

LINQ 쿼리에 대한 힌트를 얻으려면 책 792쪽을 살펴보세요.

IDE가 XAML 디자이너에 에러 메시지를 띄운다면, ViewModel 네임스페이스에 LeagueViewModel이 존재하지 않을 겁니다. 하지만 여러분이 정확하게 클래스를 추가했다면, BasketballRoster 프로젝트에 마우스 오른쪽 버튼을 클릭한 후, 프로젝트 언로드(Unload Project)를 선택합니다. 다시 오른쪽 버튼을 클릭한 후 프로젝트 다시 로드(Reload Project)를 선택해서 프로젝트를 다시 불러옵니다. 물론 C# 코드 파일에 어떤 오류도 없어야 합니다.



LeagueViewModel 클래스는 RosterControl을 데이터 컨텍스트로 사용할 수 있는 RosterViewModel을 노출합니다. 그리고 이 필드는 RosterViewModel을 사용할 수 있도록 모델 객체를 생성합니다.

BasketballRoster 앱의 뷰모델에는 세 클래스가 있습니다(LeagueViewModel, PlayerViewModel, RosterViewModel 클래스). 이들은 모두 ViewModel 폴더에 있습니다.

```
namespace BasketballRoster.ViewModel {
    using Model;
    using System.Collections.ObjectModel;
```

만약 "using Model;"이 없다면, 여러분은 Roster를 입력하는 대신 Model.Roster를 입력해야 합니다.

```
class LeagueViewModel {
    public RosterViewModel BriansTeam { get; private set; }
    public RosterViewModel JimmysTeam { get; private set; }

    public LeagueViewModel() {
        Roster briansRoster = new Roster("The Bombers", GetBomberPlayers());
        BriansTeam = new RosterViewModel(briansRoster);

        Roster jimmysRoster = new Roster("The Amazins", GetAmazinPlayers());
        JimmysTeam = new RosterViewModel(jimmysRoster);
    }
```

이 private 메서드는 Player 객체의 새로운 리스트를 생성하는 데미 데이터를 만들어냅니다.

```
private IEnumerable<Player> GetBomberPlayers() {
    List<Player> bomberPlayers = new List<Player>() {
        new Player("Brian", 31, true),
        new Player("Lloyd", 23, true),
        new Player("Kathleen", 6, true),
        new Player("Mike", 0, true),
        new Player("Joe", 42, true),
        new Player("Herb", 32, false),
        new Player("Fingers", 8, false),
    };
    return bomberPlayers;
}
```

데미 데이터는 일반적으로 뷰모델에 있습니다. 왜냐하면 MVVM 응용 프로그램의 상태는 ViewModel 객체 안에 캡슐화된 Model 클래스의 인스턴스로 관리하기 때문이죠.

데이터를 저장하기 위해 뷰의 클래스들을 이용할 수 있습니다. 이 메서드는 PlayerViewModel 객체가 아닌 Player 객체를 반환하기 때문이죠.

```
private IEnumerable<Player> GetAmazinPlayers() {
    List<Player> amazinPlayers = new List<Player>() {
        new Player("Jimmy", 42, true),
        new Player("Henry", 11, true),
        new Player("Bob", 4, true),
        new Player("Lucinda", 18, true),
        new Player("Kim", 16, true),
        new Player("Bertha", 23, false),
        new Player("Ed", 21, false),
    };
    return amazinPlayers;
}
```

```
namespace BasketballRoster.ViewModel {
    class PlayerViewModel {
        public string Name { get; private set; }
        public int Number { get; private set; }

        public PlayerViewModel(string name, int number) {
            Name = name;
            Number = number;
        }
    }
}
```

PlayerViewModel 클래스는 간단한 데이터 객체입니다. 데이터 템플릿을 바인딩하는 속성이 있습니다.

```
namespace BasketballRoster.ViewModel {
    using Model;
    using System.Collections.ObjectModel;
    using System.ComponentModel;
```

보통 MVVM 앱에서는 ViewModel의 클래스들만 INotifyPropertyChanged 인터페이스를 구현합니다. 왜냐하면 이들만이 XAML 컨트롤에 바인딩된 객체이기 때문이죠.

```
class RosterViewModel : INotifyPropertyChanged {
    public ObservableCollection<PlayerViewModel> Starters { get; private set; }
    public ObservableCollection<PlayerViewModel> Bench { get; private set; }
```

ViewModel 내부에 있는 캡슐화된 Roster 객체에서 앱의 상태를 저장합니다. 클래스의 나머지 부분은 View를 바인딩할 수 있는 속성으로 Model 데이터를 바꿉니다.

```
private string _teamName;
public string TeamName {
    get { return _teamName; }
    set {
        _teamName = value;
    }
}
```

TeamName 속성이 변할 때마다, RosterViewModel은 PropertyChanged 이벤트를 발생시켜서, 바인딩된 객체들을 갱신합니다.

```
public RosterViewModel(Roster roster) {
    _roster = roster;

    Starters = new ObservableCollection<PlayerViewModel>();
    Bench = new ObservableCollection<PlayerViewModel>();

    TeamName = _roster.TeamName;

    UpdateRosters();
}
```

```
private void UpdateRosters() {
    var startingPlayers =
        from player in _roster.Players
        where player.Starter
        select player;

    foreach (Player player in startingPlayers)
        Starters.Add(new PlayerViewModel(player.Name, player.Number));
```

LINQ 쿼리는 모든 선발 플레이어들을 찾아서 ObservableCollection 속성의 Starters에 추가합니다.

```
var benchPlayers =
    from player in _roster.Players
    where player.Starter == false
    select player;
```

여기에 벤치 플레이어들을 찾는 비슷한 LINQ 쿼리가 있네요.

```
foreach (Player player in benchPlayers)
    Bench.Add(new PlayerViewModel(player.Name, player.Number));
}
```

일반적인 MVVM 앱은 ViewModel 클래스에서만 INotifyPropertyChanged를 구현합니다. 왜냐하면 ViewModel은 XAML 컨트롤을 바인딩하는 객체들만 포함하기 때문이죠. 그러나 연습문제에서는 생성자에서 바인딩된 속성을 갱신하지 않기 때문에, INotifyPropertyChanged를 구현하지 않았습니다. 브라이언과 지미가 그들의 팀 이름을 변경하기 위해서 프로젝트를 수정하고 싶다면, TeamName의 set 접근자에서 PropertyChanged 이벤트를 발생시켜야 합니다.



스톱워치 뷰 만들기

스톱워치 컨트롤을 위한 XAML 코드가 있습니다. **View 폴더에 BasicStopwatch.xaml**이라는 WPF 사용자 정의 컨트롤을 추가하고, 아래의 코드를 입력하세요. 사용자 정의 컨트롤에는 경과 시간과 랩 타임을 표시하기 위한 TextBlock 컨트롤과 시작, 중지, 리셋, 랩 타임 버튼이 있습니다.



책 810, 811쪽을 참고할 때, 고쳐야 할 부분이 하나 있습니다. 810쪽에서 아래의 using문을 찾아주세요.

```
using Windows.UI.Xaml;
```

그리고 아래와 같이 바꿔주세요.

```
using System.Windows.Threading;
```

Windows.UI.Xaml 네임스페이스는 윈도우 스토어에 대한 .NET 프레임워크의 일부입니다. 이걸 WPF 응용 프로그램에서는 사용할 수 없죠. 대신 System.Windows.Threading 네임스페이스가 필요합니다.

이 부분을 제외하고 코드가 똑같습니다. 그리고 여기에서 Model-View-ViewModel의 좋은 예를 보여 주고 있습니다. 하나의 using 문을 제외하고, ViewModel과 Model에 대해서 똑같은 C# 코드를 사용하기 때문에, 윈도우 스토어 앱의 스톱워치 앱 클래스들을 WPF에서 쉽게 재사용할 수 있습니다.

```
<UserControl x:Class="Stopwatch.View.BasicStopwatch"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  mc:Ignorable="d"
  d:DesignHeight="300" d:DesignWidth="300"
  xmlns:viewmodel="clr-namespace:Stopwatch.ViewModel">

  <UserControl.Resources>
    <viewmodel:StopwatchViewModel x:Key="viewModel"/>
  </UserControl.Resources>

  <Grid DataContext="{StaticResource ResourceKey=viewModel}">
    <StackPanel>
      <TextBlock>
        <Run>Elapsed time: </Run>
        <Run Text="{Binding Hours, Mode=OneWay}" />
        <Run>:</Run>
        <Run Text="{Binding Minutes, Mode=OneWay}" />
        <Run>:</Run>
        <Run Text="{Binding Seconds, Mode=OneWay}" />
      </TextBlock>
      <TextBlock>
        <Run>Lap time: </Run>
        <Run Text="{Binding LapHours, Mode=OneWay}" />
        <Run>:</Run>
        <Run Text="{Binding LapMinutes, Mode=OneWay}" />
        <Run>:</Run>
        <Run Text="{Binding LapSeconds, Mode=OneWay}" />
      </TextBlock>
      <StackPanel Orientation="Horizontal">
        <Button Click="StartButton_Click" Margin="0,0,5,0">Start</Button>
        <Button Click="StopButton_Click" Margin="0,0,5,0">Stop</Button>
        <Button Click="ResetButton_Click" Margin="0,0,5,0">Reset</Button>
        <Button Click="LapButton_Click">Lap</Button>
      </StackPanel>
    </StackPanel>
  </Grid>
</UserControl>
```

← 네임스페이스를 추가하기 위해서 이 xaml 속성이 필요합니다. 프로젝트의 이름이 Stopwatch 이기 때문에, 뷰모델의 네임스페이스는 Stopwatch.ViewModel입니다.

이 사용자 정의 컨트롤은 정적 리소스로써 ViewModel의 인스턴스를 저장합니다. 그리고 인스턴스를 데이터 컨텍스트로 사용하죠. 데이터 컨텍스트를 설정하기 위해서 컨테이너가 필요 없습니다. 인스턴스에서 자신의 상태를 계속 저장하고 있습니다.

이 TextBlock 컨트롤은 경과 시간을 반환해 주는 ViewModel의 속성에 바인딩되어 있습니다.

이 TextBlock 컨트롤은 랩 타임을 노출하는 속성에 바인딩되어 있습니다.

← ViewModel은 이 값들을 최상으로 유지하기 위해서 PropertyChanged 이벤트를 발생시켜야 합니다.

컴파일을 위해서 ViewModel 네임스페이스의 StopwatchViewModel 클래스와 컨트롤에 Click 이벤트 핸들러를 추가해야 합니다.

힌트: 지속적으로 Model을 확인하고, 속성을 업데이트 하기 위해서 DispatcherTimer를 사용합니다.

책 810, 811쪽에 ViewModel의 코드가 있습니다. 책을 보기 전에 View와 Model의 코드로부터 얼마만큼 ViewModel 코드를 작성할 수 있나요? BasicStopwatch 컨트롤을 메인 윈도우에 추가해 보세요. 그리고 코드를 작성한 뒤, 810, 811쪽과 비교해 보세요.

스톱워치 앱 완성하기

몇 가지 해결해야 할 부분이 있습니다. 여러분의 스톱워치 사용자 정의 컨트롤에는 이벤트 핸들러가 없어서 이를 추가해야 합니다. 그리고 메인 윈도우에 컨트롤을 추가해야 합니다.

① 먼저, BasicStopwatch.xaml.cs에서 아래의 이벤트 핸들러를 코드-비하인드에 추가합니다.

```

ViewModel.StopwatchViewModel viewModel;

public BasicStopwatch() {
    InitializeComponent();

    viewModel = FindResource("viewModel") as ViewModel.StopwatchViewModel;
}

private void StartButton_Click(object sender, RoutedEventArgs e) {
    viewModel.Start();
}

private void StopButton_Click(object sender, RoutedEventArgs e) {
    viewModel.Stop();
}

private void ResetButton_Click(object sender, RoutedEventArgs e) {
    viewModel.Reset();
}

private void LapButton_Click(object sender, RoutedEventArgs e) {
    viewModel.Lap();
}

```

View의 버튼은 단지 ViewModel의 메서드를 호출합니다. 이것은 View의 깔끔하고 일반적인 패턴이죠.

부록에서 두 페이지 모드를 유지하기 위해서 일부러 페이지를 비웁니다. 연습문제와 연습문제 정답이 서로 마주하는 면에 나오면 안 되기 때문이죠. 그리고 두 페이지 모드의 짝수 쪽에는 오른쪽 위에, 홀수 쪽에는 왼쪽 위에 작은 캡션이 있습니다.

② MainWindow.xaml에 대한 XAML 코드입니다.

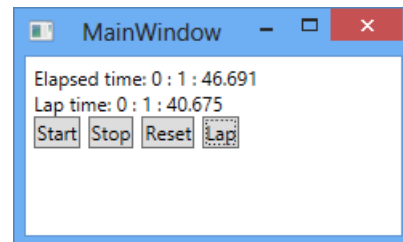
```

<Window x:Class="Stopwatch.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="150" Width="250"
    xmlns:view="clr-namespace:Stopwatch.View">
    <Grid>
        <view:BasicStopwatch Margin="5"/>
    </Grid>
</Window>

```

모든 행위는 사용자 정의 컨트롤에 있습니다. 그래서 메인 윈도우에 대한 코드-비하인드가 없습니다.

이제 앱을 실행해 봅시다. Start, Stop, Reset, Lap 버튼을 눌러 스톱워치가 작동하는지 확인해 봅시다.



변환기는 자동으로 바인딩 값을 변환해 줍니다

디지털 시계는 보통 “분” 앞의 숫자에 0이 있습니다. 우리의 스톱워치도 마찬가지로 두 숫자의 “분”을 보여 줍니다. 그리고 “초” 역시도 두 숫자를 표시하고, 백분의 일초에 가깝게 시간이 돌아갑니다. 적절한 형식의 문자열 값을 표시하기 위해서 뷰모델을 수정할 수 있습니다. 하지만 이것은 같은 데이터를 다른 형식으로 표시할 때마다 계속 다른 속성을 추가해야 하는 것을 의미하죠. 그래서 값 변환기(Value converter)는 매우 편리합니다. 값 변환기는 XAML 바인딩을 이용한 객체입니다. 데이터가 컨트롤에 전달되기 전에 수정해 주죠. IValueConverter 인터페이스(System.Windows.Data 네임스페이스에 있는)를 구현함으로써 값 변환기를 만들 수 있습니다. 이제 값 변환기를 스톱워치에 추가해 봅시다.



변환기는 뷰모델을 만드는 데 유용한 도구입니다.

1 ViewModel 폴더에 TimeNumberFormatConverter 클래스를 추가합니다.

using System.Windows.Data;를 클래스의 맨 위에 추가한 뒤, IValueConverter 인터페이스를 구현합니다. IDE를 이용하여 자동으로 인터페이스를 구현합니다. 이것은 두 메서드 스텝을 추가합니다(Convert(), ConvertBack() 메서드).

2 Convert() 메서드를 구현합니다.

Convert() 메서드는 몇몇의 매개변수가 있습니다. 그 중에서 두 개를 사용해 봅시다. value 매개변수는 바인딩을 통해 전달되는 원본 값이고, parameter 매개변수는 XAML에서 매개변수를 지정할 수 있습니다.

```
using System.Windows.Data;
```

```
class TimeNumberFormatConverter : IValueConverter {
    public object Convert(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture) {
        if (value is decimal)
            return ((decimal)value).ToString("00.00");
        else if (value is int) {
            if (parameter == null)
                return ((int)value).ToString("d1");
            else
                return ((int)value).ToString(parameter.ToString());
        }
        return value;
    }
    public object ConvertBack(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture) {
        throw new NotImplementedException();
    }
}
```

변환기는 어떻게 decimal을 int 유형으로 변환시키는지 알고 있습니다. int 값의 경우, 선택적으로 매개변수에 전달할 수 있습니다.

ConvertBack() 메서드는 양방향 바인딩을 사용합니다. 이 프로젝트에서 이 메서드를 사용하지 않습니다. 그래서 이 메서드 스텝을 아래와 같이 남겨둡니다.

코드에서 아직 구현되지 않았다는 NotImplementedException 예외를 남기는 좋은 예입니다. 프로젝트에서 이 메서드가 절대로 실행되지 않습니다. 하지만, 만약 이 메서드가 실행된다면, 사용자는 이것을 보지 못한 채 조용히 실패를 처리하는 게 좋을까요? 아니면 문제를 추적할 수 있도록 예외를 던지는 것이 좋을까요? 이 중 어느 것이 더 좋다고 생각하나요? 반드시 하나의 정답이 있는 것은 아닙니다.

3 BasicStopwatch.xaml의 정적 리소스에 변환기를 추가합니다. ViewModel 객체를 다음과 같이 추가합니다.

```
<UserControl.Resources>
    <viewModel:StopwatchViewModel x:Key="viewModel"/>
    <viewModel:TimeNumberFormatConverter x:Key="timeNumberFormatConverter"/>
</UserControl.Resources>
```



이 라인을 추가한 후, 디자이너에서 프로젝트가 실행되지 않는다면 놓루션 다시 빌드를 해야 할 수도 있습니다. 드물지만 프로젝트를 언로드하고 다시 로드해야 할 수도 있습니다.

4 값 변환기를 사용하기 위해 XAML 코드를 수정합니다.

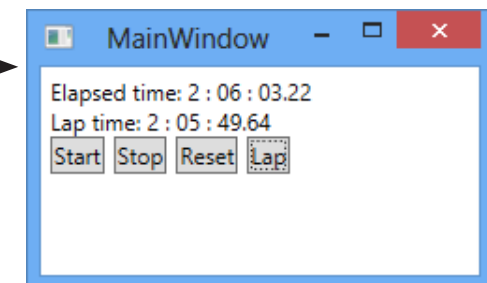
각 <Run> 태그에 있는 “Converter=”를 추가해서 {Binding} 마크업을 수정합니다.

```
<TextBlock>
    <Run>Elapsed time: </Run>
    <Run Text="{Binding Hours, Mode=OneWay,
        Converter={StaticResource timeNumberFormatConverter}}"/>
</Run></Run>
    <Run Text="{Binding Minutes, Mode=OneWay,
        Converter={StaticResource timeNumberFormatConverter}, ConverterParameter=d2}"/>
</Run></Run>
    <Run Text="{Binding Seconds, Mode=OneWay,
        Converter={StaticResource timeNumberFormatConverter}}"/>
</TextBlock>
<TextBlock>
    <Run>Lap time: </Run>
    <Run Text="{Binding LapHours, Mode=OneWay,
        Converter={StaticResource timeNumberFormatConverter}}"/>
</Run></Run>
    <Run Text="{Binding LapMinutes, Mode=OneWay,
        Converter={StaticResource timeNumberFormatConverter}, ConverterParameter=d2}"/>
</Run></Run>
    <Run Text="{Binding LapSeconds, Mode=OneWay,
        Converter={StaticResource timeNumberFormatConverter}}"/>
</TextBlock>
```

매개변수가 없다면, 닫는 괄호(})를 추가하는 것을 잊지 마세요.

ConverterParameter를 이용하여 매개변수를 변환기에 전달해줍니다.

이제 스톱워치는 TextBlock 컨트롤에 값을 전달하기 전에 변환기를 통해 값을 실행합니다. 그리고 창에서 숫자가 올바른 형식으로 나타납니다.



변환기는 다양한 유형으로 작업할 수 있습니다

TextBlock과 TextBox 컨트롤은 텍스트를 취급해서, 문자열 혹은 숫자를 Text 속성에 바인딩합니다. 하지만 다양한 속성들이 존재하고, 마찬가지로 그 속성들도 바인딩할 수 있습니다. 만약 뷰모델에 부울 속성이 있다면, 그것은 모든 true/false 속성에 바인딩될 수 있습니다. 심지어 열거형을 이용하여 속성을 바인딩할 수도 있습니다(IsVisible 속성은 값 변환기를 쓸 수 있는 Visibility 열거형을 사용합니다). 부울 속성과 Visibility 바인딩을 추가하여 스톱워치를 바꿔봅시다.

여기에 유용한 두 컨버터가 있군요.

바인딩된 속성이 false일 경우, 컨트롤이 활성화되도록 IsEnabled 같은 부울 속성을 바인딩하고 싶을 때가 있습니다. 우리는 BooleanNotConverter라 불리는 새로운 변환기를 추가할 겁니다. ! 연산자는 대상 부울 속성을 반전시킵니다.

```
IsEnabled="{Binding Running, Converter={StaticResource notConverter}}"
```

여러분은 종종 데이터 컨텍스트에서 부울 속성을 기준으로 컨트롤을 표시하거나 숨기고 싶어 합니다. 여러분은 컨트롤의 Visibility 속성을 Visibility 유형의 대상 속성으로 바인딩할 수 있습니다(Visibility.Collapsed와 같이 값을 반환하는 것을 의미합니다). BooleanVisibilityConverter라 불리는 변환기를 추가하여 컨트롤의 Visibility 속성을 대상 부울 속성에 바인딩하여, 컨트롤을 표시하거나 숨겨 봅시다.

```
Visibility="{Binding Running, Converter={StaticResource visibilityConverter}}"
```

1 뷰모델의 Tick 이벤트 핸들러를 수정합니다.

Running 속성의 값이 변했을 때 PropertyChanged 이벤트를 발생시키기 위해서 DispatcherTimer의 Tick 이벤트 핸들러를 수정합니다.

```
int _lastHours;
int _lastMinutes;
decimal _lastSeconds;
bool _lastRunning;
void TimerTick(object sender, object e) {
    if (_lastRunning != Running) {
        _lastRunning = Running;
        OnPropertyChanged("Running");
    }
    if (_lastHours != Hours) {
        _lastHours = Hours;
        OnPropertyChanged("Hours");
    }
    if (_lastMinutes != Minutes) {
        _lastMinutes = Minutes;
        OnPropertyChanged("Minutes");
    }
    if (_lastSeconds != Seconds) {
        _lastSeconds = Seconds;
        OnPropertyChanged("Seconds");
    }
}
```

Running을
추가하여 타이머를
체크합니다. 모델이
대신 이벤트를
발생시키는 것이 더
좋을까요?



2 부울 값을 반전시키는 변환기를 추가합니다.

아래에 true를 false 혹은 그 반대로 반전시켜주는 값 변환기가 있습니다. 컨트롤에 IsEnabled 속성처럼 부울 속성을 사용할 수 있습니다.

```
using System.Windows.Data;
```

```
class BooleanNotConverter : IValueConverter {
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture) {
        if ((value is bool) && ((bool)value) == false)
            return true;
        else
            return false;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture) {
        throw new NotImplementedException();
    }
}
```



3 부울 속성을 Visibility 열거형으로 바꾸는 변환기를 추가합니다.

여러분은 Visibility 속성을 Visible 혹은 Collapsed로 설정함으로써 컨트롤을 보이거나 보이지 않게 만드는 방법을 이미 알고 있습니다. Visible와 Collapsed의 값은 System.Windows 네임스페이스의 Visibility의 열거형에 있습니다. 아래에 부울 값을 Visibility 값으로 바꿔주는 변환기가 있네요.

```
using System.Windows;
using System.Windows.Data;
```

```
class BooleanVisibilityConverter : IValueConverter {
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture) {
        if ((value is bool) && ((bool)value) == true)
            return Visibility.Visible;
        else
            return Visibility.Collapsed;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture) {
        throw new NotImplementedException();
    }
}
```

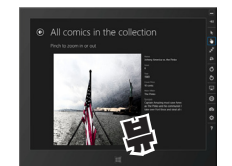
4 변환기를 사용하기 위해서 스톱워치 컨트롤을 수정합니다.

BasicStopwatch.xaml을 수정하여 정적 리소스의 변환기 인스턴스를 추가합니다.

```
<viewModel:BooleanVisibilityConverter x:Key="visibilityConverter"/>
<viewModel:BooleanNotConverter x:Key="notConverter"/>
```

이제 컨트롤의 IsEnabled와 Visibility 속성을 뷰모델의 Running 속성에 바인딩할 수 있습니다.

```
<StackPanel Orientation="Horizontal">
    <Button IsEnabled="{Binding Running, Converter={StaticResource notConverter}}">Start</Button>
    <Button IsEnabled="{Binding Running}" Click="StopButton_Click" Margin="0,0,5,0">Stop</Button>
    <Button Click="ResetButton_Click" Margin="0,0,5,0">Reset</Button>
    <Button IsEnabled="{Binding Running}" Click="LapButton_Click">Lap</Button>
</StackPanel>
<TextBlock Text="Stopwatch is running"
    Visibility="{Binding Running, Converter={StaticResource visibilityConverter}}"/>
```



← 스톱워치가 실행되지 않은 경우에만, Start 버튼을 활성화합니다.

이것은 스톱워치가 실행될 때, TextBlock 컨트롤을 보이게 해줍니다.

같은 뷰모델을 이용하여 아날로그 스톱워치를 만들어 봅시다

MVVM 패턴은 뷰모델에서 뷰를, 모델에서 뷰모델을 분리(decouple)합니다. MVVM 패턴은 이 중 하나를 변경해야 할 경우 매우 유용합니다. 역할을 분리하는 디커플링(decoupling) 덕분에 “산탄 총 수술” 효과 없이 쉽게 수정할 수 있고, 다른 계층에서도 파문이 일어나지 않습니다. 그래서 스톱워치 프로그램의 뷰모델에서 뷰의 디커플링을 잘 했을까요? 이것을 확인하는 한 가지 방법이 있습니다. 뷰모델에 있는 클래스의 변경 없이 전체로 새로운 뷰를 만들어 봅시다. C# 코드에서 변경해야 할 단 한 가지는 숫자로 된 분과 초를 아날로그 형태로 바꾸는 뷰모델의 새 변환기입니다.

14장에서 저미의 만화책에 데이터 클래스 만들어서 어떻게 사용했고, 클래스를 대사용해서 아무런 변화 없이 분할 앱을 생성한 거 기억하시죠? 이것과 같은 발상입니다.

직접 해 봅시다!

1 디지털 시간을 아날로그로 바꾸는 변환기를 추가합니다.

ViewModel 폴더에 AngleConverter 클래스를 추가해 주세요. 동그란 시계의 침을 위해 사용됩니다.

```
using System.Windows.Data;
class AngleConverter : IValueConverter {
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture) {
        double parsedValue;
        if ((value != null)
            && double.TryParse(value.ToString(), out parsedValue)
            && (parameter != null))
            switch (parameter.ToString()) {
                case "Hours":
                    return parsedValue * 30;
                case "Minutes":
                case "Seconds":
                    return parsedValue * 6;
            }
        return 0;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture) {
        throw new NotImplementedException();
    }
}
```



시간의 범위는 0~11이고, 이를 각도로 변환하려면 30을 곱해줍니다(360/12 = 30).

분과 초의 범위는 0~59이고, 이를 각도로 변환하면 6을 곱해줍니다(360/60=6).

부록에서 두 페이지 모드를 유지하기 위해서 일부러 페이지를 비웠습니다. 연습문제와 연습문제 정답이 서로 마주하는 면에 나오면 안 되기 때문이죠. 그리고 두 페이지 모드의 짝수 쪽에는 오른쪽 위에, 홀수 쪽에는 왼쪽 위에 작은 캡션이 있습니다.

2 새 UserControl을 추가합니다.

View 폴더에 AnalogStopwatch라 불리는 새로운 사용자 정의 컨트롤을 추가합니다. 그리고 ViewModel의 네임스페이스에 <UserControl> 태그를 추가합니다. 너비와 높이도 바꿔주세요.

```
d:DesignHeight="300"
d:DesignWidth="400"
xmlns:viewmodel="using:Stopwatch.ViewModel">
```

그리고 뷰모델과 두 변환기, 스타일을 사용자 정의 컨트롤의 정적 리소스에 추가해 주세요.

```
<UserControl.Resources>
    <viewmodel:StopwatchViewModel x:Key="viewModel"/>
    <viewmodel:BooleanNotConverter x:Key="notConverter"/>
    <viewmodel:AngleConverter x:Key="angleConverter"/>
</UserControl.Resources>
```



3 그리드에 동그란 시계 판과 침을 추가합니다.

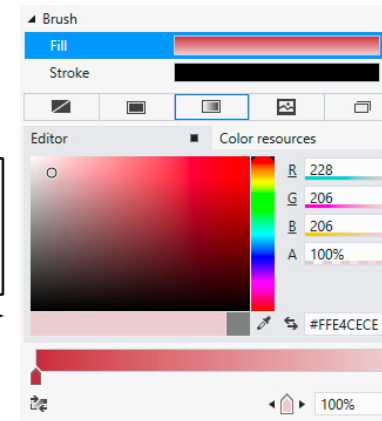
<Grid> 태그를 수정하여 동그란 스톱워치 판과, 4개의 사각형으로 침을 추가해 주세요.



```
<Grid x:Name="baseGrid" DataContext="{StaticResource ResourceKey=viewModel}">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="400"/>
  </Grid.ColumnDefinitions>
  <Ellipse Width="300" Height="300" Stroke="Black" StrokeThickness="2">
    <Ellipse.Fill>
      <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
        <LinearGradientBrush.RelativeTransform>
          <CompositeTransform CenterY="0.5" CenterX="0.5" Rotation="45"/>
        </LinearGradientBrush.RelativeTransform>
        <GradientStop Color="#FFB03F3F"/>
        <GradientStop Color="#FFE4CECE" Offset="1"/>
      </LinearGradientBrush>
    </Ellipse.Fill>
  </Ellipse>
  <Rectangle RenderTransformOrigin="0.5,0.5" Width="2" Height="150" Fill="Black">
    <Rectangle.RenderTransform>
      <TransformGroup>
        <TranslateTransform Y="-60"/>
        <RotateTransform Angle="{Binding Seconds, Converter={StaticResource ResourceKey=angleConverter}, ConverterParameter=Seconds}"/>
      </TransformGroup>
    </Rectangle.RenderTransform>
  </Rectangle>
  <Rectangle RenderTransformOrigin="0.5,0.5" Width="4" Height="100" Fill="Black">
    <Rectangle.RenderTransform>
      <TransformGroup>
        <TranslateTransform Y="-50"/>
        <RotateTransform Angle="{Binding Minutes, Converter={StaticResource ResourceKey=angleConverter}, ConverterParameter=Minutes}"/>
      </TransformGroup>
    </Rectangle.RenderTransform>
  </Rectangle>
  <Rectangle RenderTransformOrigin="0.5,0.5" Width="1" Height="150" Fill="Yellow">
    <Rectangle.RenderTransform>
      <TransformGroup>
        <TranslateTransform Y="-60"/>
        <RotateTransform Angle="{Binding LapSeconds, Converter={StaticResource ResourceKey=angleConverter}, ConverterParameter=Seconds}"/>
      </TransformGroup>
    </Rectangle.RenderTransform>
  </Rectangle>
  <Rectangle RenderTransformOrigin="0.5,0.5" Width="2" Height="100" Fill="Yellow">
    <Rectangle.RenderTransform>
      <TransformGroup>
        <TranslateTransform Y="-50"/>
        <RotateTransform Angle="{Binding LapMinutes, Converter={StaticResource ResourceKey=angleConverter}, ConverterParameter=Minutes}"/>
      </TransformGroup>
    </Rectangle.RenderTransform>
  </Rectangle>
  <Ellipse Width="10" Height="10" Fill="Black"/>
</Grid>
```

이것은 스톱워치의 동그란 판입니다. 검은 윤곽선과 회색 그라데이션 배경으로 되어 있습니다.

스톱워치 판은 Save the Humans에 사용한 배경화면처럼 회색 그라데이션 브러시로 채워져 있습니다.



여기에 두 번째 침이 있네요. 길고 가는 사각형이고, 숫자를 각도로 변환해 줍니다.

<TranslateTransform Y="-60"/>

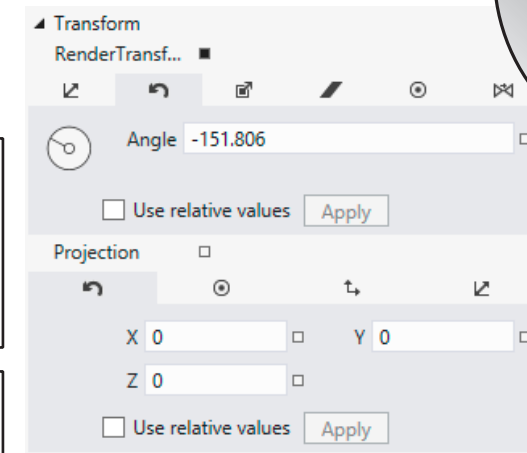
<RotateTransform Angle="{Binding Seconds, Converter={StaticResource ResourceKey=angleConverter}, ConverterParameter=Seconds}"/>

두 번째 변환은 올바른 각도로 회전하는 겁니다. 회전의 Angle 속성은 뷰모델의 분과 초에 바인딩되어 있고, 각도 변환기를 이용해서 숫자 값을 각도로 변환해 줍니다.

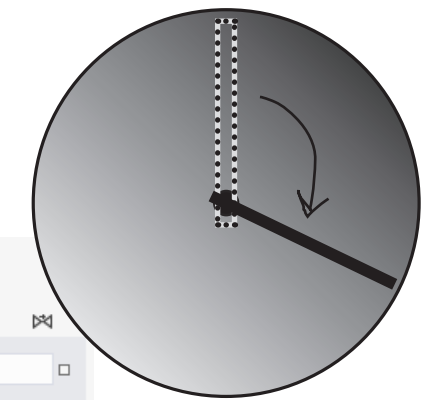
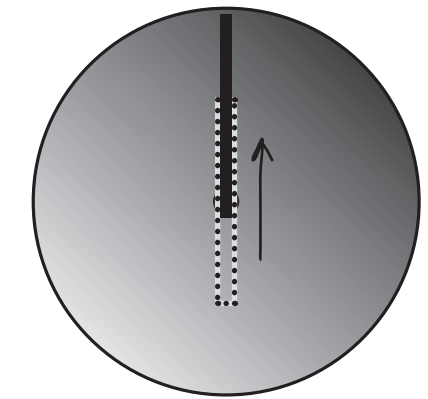
모든 컨트롤은 하나의 RenderTransform 섹션을 가지고 있습니다.

모든 컨트롤은 표시될 방법을 바꿀 수 있는 하나의 RenderTransform 요소를 가질 수 있습니다. 이것은 회전, 오프셋 이동, 크기를 확장하거나 축소하는 것 등을 포함하죠.

Save the Humans에서 외계인처럼 보이려고 적군의 타원 모양을 바꾸기 위해서 "변형(Transform)"을 사용했습니다.



침이 겹치는 것을 가리기 위해서, 중간에 다른 원을 그려줍니다. 이것은 Grid의 하단에 있기 때문에 마지막으로 그려서, 상단으로 오게 합니다.



눈침이 추가되자마자 스톱워치가 틱(tick)을 시작합니다. 디자이너에서 컨트롤을 그리기 위해서, 정적 리소스인 뷰모델의 인스턴스가 생성되기 때문이죠. 디자이너가 갱신을 하지 않을 수도 있습니다. 이 경우 디자이너 탭을 다시 전환해서 다시 확인할 수 있습니다.

4 스톱워치에 버튼을 추가합니다.

ViewModel이 동일하기 때문에 버튼도 같은 방식으로 동작합니다. BasicStopwatch.xaml에서 사용했던 버튼을 AnalogStopwatch.xaml에 추가합니다.

```
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom">
  <Button IsEnabled="{Binding Running, Converter={StaticResource notConverter}}"
    Click="StartButton_Click" Margin="0,0,5,0">Start</Button>
  <Button IsEnabled="{Binding Running}"
    Click="StopButton_Click" Margin="0,0,5,0">Stop</Button>
  <Button Click="ResetButton_Click" Margin="0,0,5,0">Reset</Button>
  <Button IsEnabled="{Binding Running}" Click="LapButton_Click">Lap</Button>
</StackPanel>
```

AnalogStopwatch.xaml.cs에 대한 코드-비하인드입니다.

```
ViewModel.StopwatchViewModel viewModel;

public AnalogStopwatch() {
  InitializeComponent();

  viewModel = FindResource("viewModel") as ViewModel.StopwatchViewModel;
}

private void StartButton_Click(object sender, RoutedEventArgs e) {
  viewModel.Start();
}

private void StopButton_Click(object sender, RoutedEventArgs e) {
  viewModel.Stop();
}

private void ResetButton_Click(object sender, RoutedEventArgs e) {
  viewModel.Reset();
}

private void LapButton_Click(object sender, RoutedEventArgs e) {
  viewModel.Lap();
}
```

5 두 스톱워치를 보여 주기 위해서 메인 윈도우를 수정합니다.

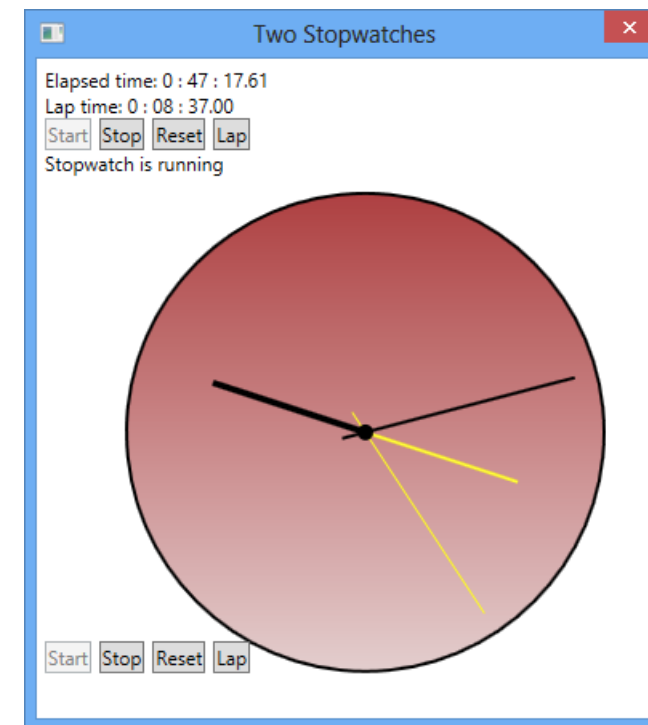
AnalogStopwatch 컨트롤을 추가하기 위해서 MainWindow.xaml을 수정합니다.

```
<Window x:Class="Stopwatch.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Two Stopwatches" Height="450" Width="400" ResizeMode="NoResize"
  xmlns:view="clr-namespace:Stopwatch.View">
  <Grid>
    <StackPanel>
      <view:BasicStopwatch Margin="5"/>
      <view:AnalogStopwatch Margin="5"/>
    </StackPanel>
  </Grid>
</Window>
```

앱을 실행해 봅시다. 창에 두 개의 스톱워치 컨트롤이 있군요.

각 스톱워치는 자신의 시간을 유지합니다. 각자 정적 리소스로서 뷰모델의 분리된 인스턴스를 가지기 때문이죠.

_stopwatchModel 필드를 static으로 만들어서 뷰모델을 바꿔보세요. 이것이 스톱워치 앱의 어떤 행동을 변화시킬까요? 무슨 일이 일어나고 있는지 맞춰보세요.



UI 컨트롤도 인스턴스화할 수 있습니다

XAML 코드가 System.Windows 네임스페이스의 클래스를 인스턴스화하는 것을 이미 알고 있습니다. 10장에서 조사식 창을 통해 살펴보았습니다. 그러나 코드 내부에서 컨트롤을 만들려고 하면 무엇을 해야 할까요? 글쎄요, 컨트롤은 단지 객체이기 때문에 컨트롤을 생성할 수 있고, 다른 어느 객체와 마찬가지로 컨트롤을 다룰 수 있습니다. **아날로그 스톱워치의 코드-비하인드를 수정하여, 시계 판에 눈금을 추가해 봅시다.**

```
public sealed partial class AnalogStopwatch : UserControl {
    public AnalogStopwatch() {
        InitializeComponent();

        viewModel = FindResource("viewModel") as ViewModel.StopwatchViewModel;
        AddMarkings(); ← 눈금을 추가하는 메서드는
                       생성자에서 호출합니다.

    private void AddMarkings() {
        for (int i = 0; i < 360; i += 3) {
            Rectangle rectangle = new Rectangle();
            rectangle.Width = (i % 30 == 0) ? 3 : 1;
            rectangle.Height = 15;
            rectangle.Fill = new SolidColorBrush(Colors.Black);
            rectangle.RenderTransformOrigin = new Point(0.5, 0.5);

            TransformGroup transforms = new TransformGroup();
            transforms.Children.Add(new TranslateTransform() { Y = -140 });
            transforms.Children.Add(new RotateTransform() { Angle = i });
            rectangle.RenderTransform = transforms;
            baseGrid.Children.Add(rectangle); ← 시험과 분침의 XAML 코드로
                                             돌아가보세요. 이 코드는 Angle
                                             속성을 바인딩하는 대신, 값을
                                             설정하는 것은 제외하고 정확히
                                             같은 변형을 설정합니다.

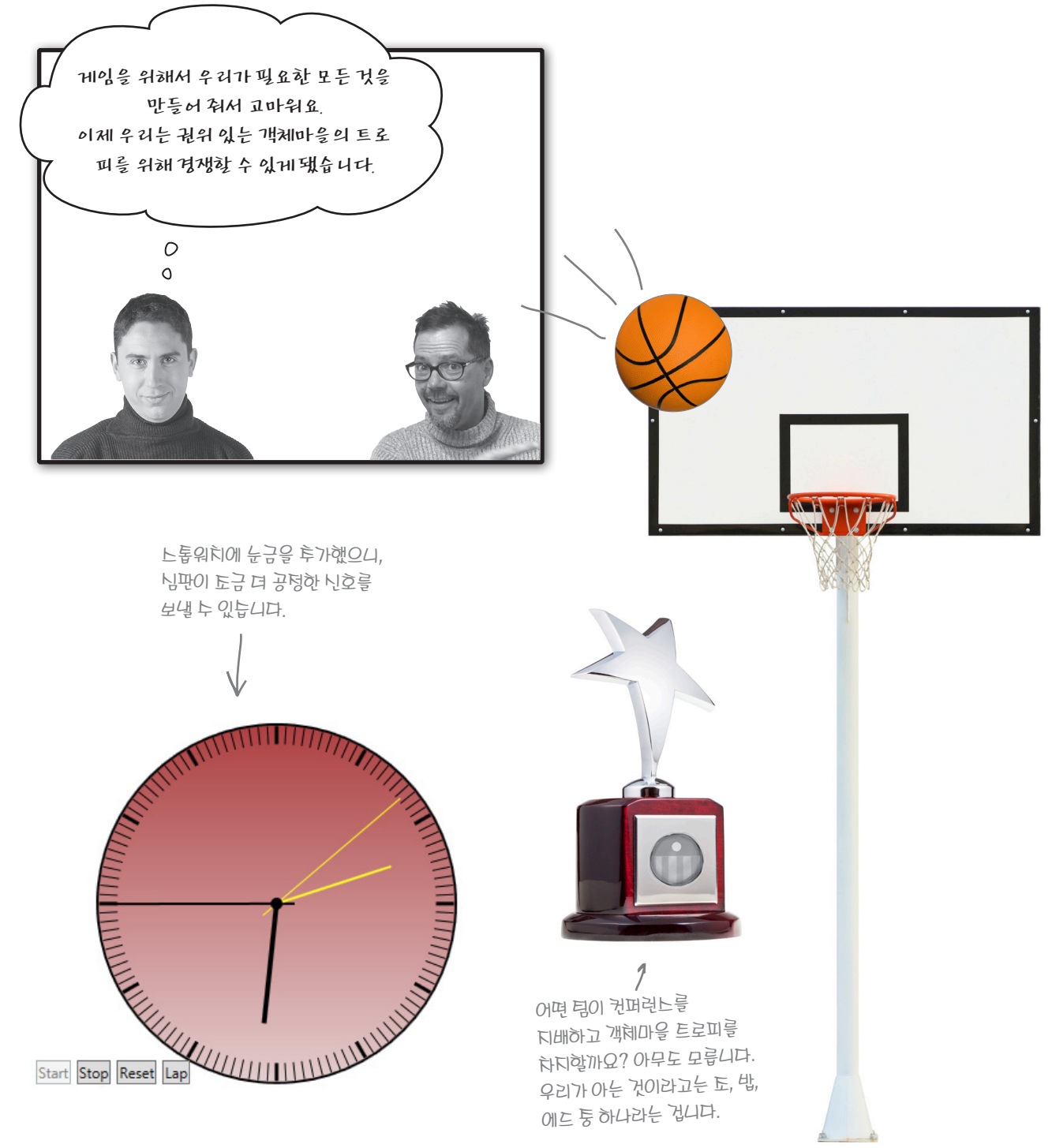
        }
    }
    // ... 버튼 이벤트 핸들러는 똑같음
}
```

<Rectangle> 태그와 같은 Rectangle 객체의 인스턴스를 생성합니다.

이 문장은 % 나머지 연산자를 이용하여 시간의 눈금을 분의 눈금보다 두껍게 만듭니다. i % 30은 30으로 나누어 떨어질 경우에만 0을 반환합니다.

Grid, StackPanel, Canvas와 같은 컨트롤은 다른 모든 컨트롤의 참조를 담을 수 있는 Children 컬렉션이 있습니다. Add() 메서드로 그리드에 컨트롤을 담을 수 있고, Clear() 메서드를 호출하여 모든 컨트롤을 지울 수 있습니다. TransformGroup도 같은 방식으로 변형을 추가합니다.

11장에서도 마찬가지로 C# 코드에서 Binding 객체를 이용하여 데이터 바인딩을 설정했습니다. 시험과 분침의 Rectangle 컨트롤을 생성하기 위해서 XAML 코드를 어떻게 지우고, 같은 일을 하기 위해서 XAML 코드를 C# 코드로 어떻게 교체하는지 아시겠죠?



이미지 애니메이션을 위한 사용자 정의 컨트롤 만들기

모든 프레임별로 애니메이션 코드를 캡슐화해 봅시다. **View** 폴더에 **AnimatedImage**라는 WPF 사용자 정의 컨트롤을 추가합니다. 이 XAML 코드는 매우 작습니다. 모든 핵심은 코드-비하인드에 있죠. XAML 의 <UserControl> 안의 내용은 이게 전부입니다.

```
<Grid>
    <Image x:Name="image" Stretch="Fill"/>
</Grid>
```

코드-비하인드 차례입니다. 오버로드된 생성자를 살펴보면, StartAnimation() 메서드를 호출합니다. 이 메서드는 Image 컨트롤의 Source 속성을 애니메이션화하기 위해서 **스토리보드와 키 프레임 애니메이션 객체를 생성**합니다.

```
using System.Windows.Media.Animation;
using System.Windows.Media.Imaging;

public partial class AnimatedImage : UserControl {
    public AnimatedImage() {
        InitializeComponent();
    }

    public AnimatedImage(IEnumerable<string> imageNames, TimeSpan interval)
        : this() {
        StartAnimation(imageNames, interval);
    }

    public void StartAnimation(IEnumerable<string> imageNames, TimeSpan interval) {
        Storyboard storyboard = new Storyboard();
        ObjectAnimationUsingKeyFrames animation = new ObjectAnimationUsingKeyFrames();
        Storyboard.SetTarget(animation, image);
        Storyboard.SetTargetProperty(animation, new PropertyPath(Image.SourceProperty));

        TimeSpan currentInterval = TimeSpan.FromMilliseconds(0);
        foreach (string imageName in imageNames) {
            ObjectKeyFrame keyFrame = new DiscreteObjectKeyFrame();
            keyFrame.Value = CreateImageFromAssets(imageName);
            keyFrame.KeyTime = currentInterval;
            animation.KeyFrames.Add(keyFrame);
            currentInterval = currentInterval.Add(interval);
        }

        storyboard.RepeatBehavior = RepeatBehavior.Forever;
        storyboard.AutoReverse = true;
        storyboard.Children.Add(animation);
        storyboard.Begin();
    }

    private static BitmapImage CreateImageFromAssets(string imageFilename) {
        try {
            Uri uri = new Uri(imageFilename, UriKind.RelativeOrAbsolute);
            return new BitmapImage(uri);
        } catch (System.IO.IOException) {
            return new BitmapImage();
        }
    }
}
```

Media.Imaging 네임스페이스에 비트맵 이미지의 BitmapImage 클래스가 있습니다. 스토리보드와 다른 애니메이션 클래스는 Media.Animation 네임스페이스에 있습니다.

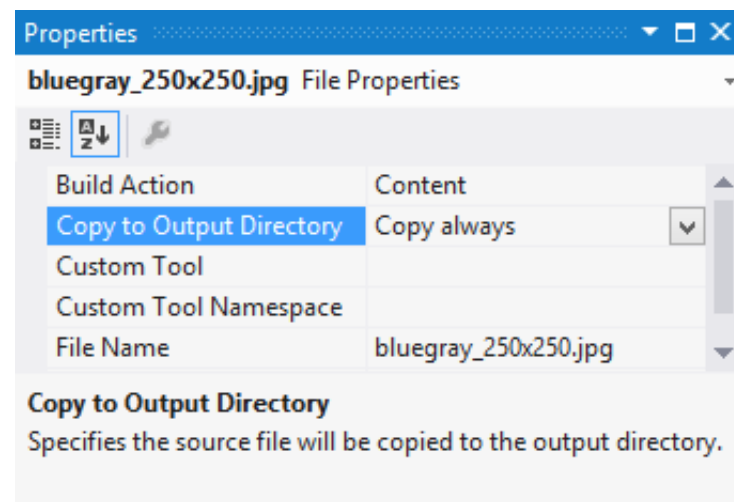
만약 XAML을 이용하여 컨트롤의 인스턴스를 생성한다면, 모든 컨트롤은 반드시 매개변수가 없는 생성자가 있어야 합니다. 생성자를 오버로드할 순 있지만, 컨트롤을 생성하기 위해서 코드를 쓰는 경우에서만 유용합니다.

Storyboard 클래스의 static SetTarget()과 SetTargetProperty() 메서드는 애니메이션화된 "이미지"(image)의 대상 객체와 "원본"(Source)을 바꿀 속성을 설정합니다.

Storyboard 객체가 생성되고 애니메이션이 Children 컬렉션에 추가되고 됐으면, 애니메이션을 호출하기 위해서 Begin() 메서드를 호출합니다.

14장에서 본 같은 메서드군요.

다음 프로젝트를 위해서 여기(<http://www.hanbit.co.kr/exam/2165p>)에서 별 이미지를 내려받습니다. 지미의 만화책 앱에서 했던 것처럼, 최상위 폴더의 프로젝트에 이미지를 추가해야 됩니다. 또한 솔루션 탐색기에서 각 이미지 파일을 선택합니다. 그리고 속성 창의 빌드 작업(Build Action)에서 내용(Content)을 출력 디렉터리로 복사(Copy to Output Directory)에서 항상 복사(Copy always)를 선택합니다. 이 작업을 지미의 만화책 앱에서 했던 것처럼 각 이미지 파일을 선택해서 해 주세요.



Bee animation 1.png, Bee animation 2.png, Bee animation 3.png, Bee animation 4.png 파일에 적용해 주세요.

벌이 날아다닙니다

시험 비행을 위해서 AnimatedImage 컨트롤을 사용해 봅시다.



1 View 폴더에서 MainPage.xaml을 삭제하고 새 기본 페이지로 교체합니다.

View 폴더에 FlyingBees.xaml이라는 윈도우를 추가합니다. 프로젝트의 MainWindow.xaml 파일은 삭제해 주세요. 그런 다음 App.xaml의 <Application> 태그에서 StartupUri 속성을 수정합니다.

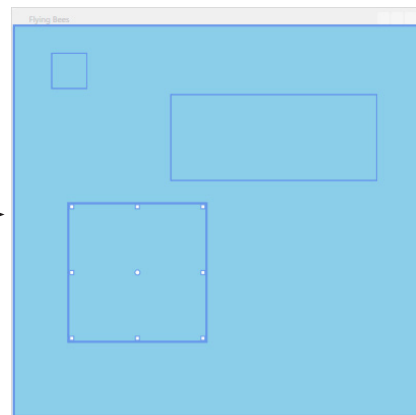
```
StartupUri="View\FlyingBees.xaml"
```

2 벌은 Canvas 컨트롤 주위를 맴돕니다.

아래에 윈도우에 대한 코드가 있습니다(다른 프로젝트 이름을 사용했다면, AnimatedBee 네임스페이스를 바꿔주세요). 이 코드에서 FlyingBees.xaml에서 Canvas 컨트롤을 사용합니다. Canvas 컨트롤은 컨테이너기 때문에 Grid 혹은 StackPanel과 같은 다른 컨트롤을 담을 수 있습니다. Canvas의 특징은 Canvas.Left와 Canvas.Top 속성을 이용하여 컨트롤의 좌표를 설정합니다. 1장의 Save the Humans에서 플레이 영역을 만들기 위해서 Canvas를 사용했습니다. FlyingBees.xaml에서도 Canvas를 추가해 봅시다.

```
<Window x:Class="AnimatedBee.View.FlyingBees"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:view="clr-namespace:AnimatedBee.View"
  Title="Flying Bees" Height="600" Width="600">
<Grid>
  <Canvas Background="SkyBlue">
    <view:AnimatedImage Canvas.Left="55" Canvas.Top="40"
      x:Name="firstBee" Width="50" Height="50"/>
    <view:AnimatedImage Canvas.Left="80" Canvas.Top="260"
      x:Name="secondBee" Width="200" Height="200"/>
    <view:AnimatedImage Canvas.Left="230" Canvas.Top="100"
      x:Name="thirdBee" Width="300" Height="125"/>
  </Canvas>
</Grid>
</Window>
```

AnimatedImage 컨트롤은 CreateFrameImage() 메서드가 호출될 때까지 보이지 않습니다. Canvas의 컨트롤의 윤곽선만 표시하죠. 문서 개요를 이용하여 선택할 수 있습니다. Canvas.Left와 Canvas.Top 속성이 변하는지 보려면 컨트롤을 캔버스 주위로 이동해 보세요.



3 윈도우의 코드-비하인드를 추가합니다.

Storyboard와 DoubleAnimation을 포함하는 네임스페이스의 using문을 추가합니다.

```
using System.Windows.Media.Animation;
```

벌 애니메이션을 구동하기 위해서 FlyingBees.xaml.cs의 생성자를 수정해 봅시다. 또한 Canvas.Left 속성을 움직이는 DoubleAnimation 객체를 생성해 봅시다. 그리고 이번 장 앞에서 본 XAML 코드에서 스토리보드와 <DoubleAnimation> 태그의 애니메이션을 생성하는 코드를 비교해 보세요.

```
public FlyingBees() {
  this.InitializeComponent();

  List<string> imageNames = new List<string>();
  imageNames.Add("Bee animation 1.png");
  imageNames.Add("Bee animation 2.png");
  imageNames.Add("Bee animation 3.png");
  imageNames.Add("Bee animation 4.png");

  firstBee.StartAnimation(imageNames, TimeSpan.FromMilliseconds(50));
  secondBee.StartAnimation(imageNames, TimeSpan.FromMilliseconds(10));
  thirdBee.StartAnimation(imageNames, TimeSpan.FromMilliseconds(100));

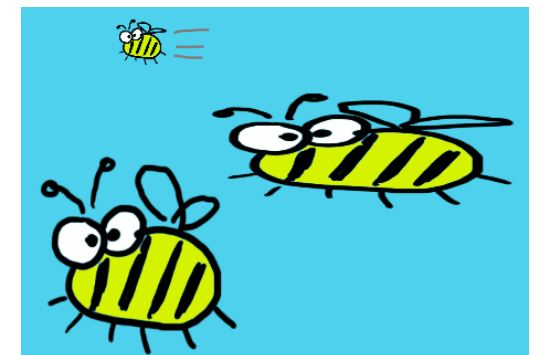
  Storyboard storyboard = new Storyboard();
  DoubleAnimation animation = new DoubleAnimation();
  Storyboard.SetTarget(animation, firstBee);
  Storyboard.SetTargetProperty(animation, new PropertyPath(Canvas.LeftProperty));
  animation.From = 50;
  animation.To = 450;
  animation.Duration = TimeSpan.FromSeconds(3);
  animation.RepeatBehavior = RepeatBehavior.Forever;
  animation.AutoReverse = true;
  storyboard.Children.Add(animation);
  storyboard.Begin();
}
```

이번 장의 앞에서 본 <Storyboard> 태그와 <DoubleAnimation> 태그를 쓰는 대신, 코드에서 Storyboard 객체와 DoubleAnimation 객체를 생성하고, 속성을 설정할 수 있습니다.

CreateFrameImages() 메서드는 이미지 이름의 눈서와 프레임이 변하는 속도를 설정하는 TimeSpan 객체를 취합니다.

애니메이션이 끝난 후 Storyboard 객체는 가비지 컬렉션으로 갑니다. 이 과정을 보기 위해서 조사식 창에서 개체 ID 만들기를 누르고, 애니메이션이 끝난 후, Make Object ID 버튼을 클릭하세요.

프로그램을 실행합니다. 이제 세 마리의 벌이 날개로 날아다니는 것을 볼 수 있습니다. 벌들은 다른 시간 간격을 갖고 있어서, 서로 다른 속도로 날아다닙니다. 프레임이 바뀌기 전에 서로 다른 시간을 기다리기 때문이죠. 맨 위의 벌은 Canvas를 가지고 있습니다. Left 속성은 50에서 450으로 움직이다가 다시 반대로 돌아옵니다. 이 속성은 벌들이 페이지 주변을 돌아다니게 합니다. DoubleAnimation 객체에 설정된 속성을 자세히 살펴보고, 앞서 사용한 XAML 속성과 비교해 보세요.



뭔가 이상한데... 뭐가 잘못된 걸까요?

뭔가 이상하군요. Model이나 ViewModel 폴더가 없고, View 폴더에는 더미 데이터만 있군요. 이것은 MVVM이 아니에요!

더 많은 벌을 추가하고 싶다면, View 폴더에 더 많은 컨트롤을 추가한 뒤, 개별적으로 초기화 작업을 해줘야 합니다. 만약 서로 다른 종류나 크기의 벌을 만들고 싶거나 애니메이션을 다르게 주고 싶다면 어떻게 해야 될까요? 데이터에 최적화된 모델이 있다면, 이 작업이 조금 더 쉬워질 겁니다. 이 프로젝트에 MVVM 패턴을 어떻게 적용할까요?



누워서 떡 먹기죠. 컨트롤의 ObservableCollection 컬렉션을 추가하고, Canvas 객체의 Children 속성을 컬렉션에 바인딩합니다. 어려울 게 뭐가 있죠?



그렇게 하면 안 됩니다. 데이터 바인딩은 컨테이너 컨트롤의 Children 속성과 동작할 수 없습니다. 데이터 바인딩은 XAML 코드에 있는 속성들과 연결되어 동작하도록 만들어졌습니다. Canvas 객체는 public Children 속성을 가지지만, XAML 코드 (Children="{Binding ...}")로 바인딩을 설정한다면, 컴파일이 되지 않습니다. 하지만 여러분은 ItemsSource 속성을 이용하여 ListView와 GridView 컨트롤을 바인딩하면서, 어떻게 객체의 컬렉션을 XAML 컨트롤에 바인딩하는지 알고 있습니다. Canvas 객체에 자식(하위) 컨트롤을 추가하기 위해서 데이터 바인딩을 활용할 수 있습니다.

ItemsPanelTemplate을 이용하여 캔버스에 컨트롤을 바인딩해 봅시다

항목들을 ListView, GridView 혹은 ListBox로 바인딩하는 ItemSource 속성을 사용했을 때, 어떤 항목을 바인딩하는지는 중요하지 않았습니다. 왜냐하면 ItemSource 속성은 항상 같은 방식으로 동작했기 때문이죠. 만약 정확히 같은 행동을 하는 세 개의 클래스를 만들려고 한다면, 그 행동을 베이스 클래스에 넣고, 세 클래스를 확장하면 될까요? 마이크로소프트팀은 항목을 선택하는 컨트롤을 만들었을 때, 정확히 같은 동작을 하도록 만들었습니다. ListView, GridView, ListBox 모두는 Selector 클래스를 확장합니다. 이 클래스는 컬렉션의 항목을 표시하는 ItemsControl 클래스의 서브 클래스입니다.

- 1 항목의 레이아웃을 조절하는 패널의 템플릿을 설정하기 위해서 ItemPanel 속성을 사용합니다. 먼저, FlyingBees.xaml에서 ViewModel 네임스페이스를 추가합니다.


```
xmlns:viewmodel="clr-namespace:AnimatedBee.ViewModel"
```
- 2 다음, BeeViewModel이라는 빈 클래스를 ViewModel 폴더에 추가하고, 이 클래스의 인스턴스를 FlyingBees.xaml의 정적 리소스에 추가합니다.


```
<viewmodel:BeeViewModel x:Key="viewModel"/>
```

 FlyingBees.xaml.cs를 편집해 봅시다. FlyingBees 컨트롤의 FlyingBees() 생성자에서 추가했던 코드만 지웁니다. 원래 있던 코드를 지우면 안 돼요.
- 3 ItemsControl에 대한 XAML 코드입니다. FlyingBees.xaml 파일을 열어서 여러분이 추가한 <Canvas> 태그를 지우고, ItemsControl로 바꿔줍니다.

다른 프로젝트 이름을 사용했다면, AnimatedBee 네임스페이스를 바꿔주세요.

정적 리소스의 뷰모델을 데이터 컨텍스트로 사용합니다. 그리고 ItemSource를 Sprites 속성에 바인딩합니다.

```
<ItemsControl
    DataContext="{StaticResource viewModel}"
    ItemSource="{Binding Path=Sprites}" >
    <ItemsControl.ItemsPanel>
        <ItemsPanelTemplate>
            <Canvas Background="SkyBlue" />
        </ItemsPanelTemplate>
    </ItemsControl.ItemsPanel>
</ItemsControl>
```

패널을 이렇게 설정할 수 있지만, 여러분이 원하는 대로 해도 상관없습니다. Canvas는 하늘색 배경이군요.

ItemsPanelTemplate을 설정하기 위해서 ItemPanel 속성을 사용합니다. ItemsPanelTemplate은 단일의 Panel 컨트롤과 Panel 클래스를 확장한 GridView와 Canvas를 포함하고 있습니다. ItemSource에 바인딩되어 있는 어떤 항목이든지 Panel의 Children에 추가될 수 있습니다.

ItemsControl이 생성될 때, 컨트롤의 모든 항목이 있는 Panel을 만들고, ItemsPanelTemplate 컨트롤 템플릿을 사용합니다.

꼭 알아야 할 기본 내용

4 View 폴더에 새 BeeHelper 클래스를 생성합니다. 뷰모델의 벌 관리를 돕기 위해서, static 메서드만 가진 static 클래스입니다.

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media.Animation;
```

```
static class BeeHelper {
    public static AnimatedImage BeeFactory(
        double width, double height, TimeSpan flapInterval) {
        List<string> imageNames = new List<string>();
        imageNames.Add("Bee animation 1.png");
        imageNames.Add("Bee animation 2.png");
        imageNames.Add("Bee animation 3.png");
        imageNames.Add("Bee animation 4.png");

        AnimatedImage bee = new AnimatedImage(imageNames, flapInterval);
        bee.Width = width;
        bee.Height = height;
        return bee;
    }

    public static void SetBeeLocation(AnimatedImage bee, double x, double y) {
        Canvas.SetLeft(bee, x);
        Canvas.SetTop(bee, y);
    }

    public static void MakeBeeMove(AnimatedImage bee,
        double fromX, double toX, double y) {
        Canvas.SetTop(bee, y);
        Storyboard storyboard = new Storyboard();
        DoubleAnimation animation = new DoubleAnimation();
        Storyboard.SetTarget(animation, bee);
        Storyboard.SetTargetProperty(animation,
            new PropertyPath(Canvas.LeftProperty));

        animation.From = fromX;
        animation.To = toX;
        animation.Duration = TimeSpan.FromSeconds(3);
        animation.RepeatBehavior = RepeatBehavior.Forever;
        animation.AutoReverse = true;
        storyboard.Children.Add(animation);
        storyboard.Begin();
    }
}
```

이 팩토리 메서드는 Bee 컨트롤을 생성합니다. 모든 게 사용자 인터페이스에 관한 내용이라서 뷰에 포함합니다.

팩토리 메서드 패턴
MVM은 많은 디자인 패턴 중 하나입니다. 가장 평범하고 유용한 패턴 중 하나는 팩토리 메서드 패턴(Factory method pattern)입니다. 객체를 생성하는 “팩토리(공장)” 메서드가 있죠. 팩토리 메서드는 대부분 static이고, 메서드 이름 끝에 “Factory”가 붙습니다. 그래야 무슨 일을 하는지 잘 알 수 있으니까요.

재사용할 일이 많은 작은 블록의 코드를 메서드(혹은 static 메서드)에 넣을 때, 때때로 이것을 헬퍼 메서드(helper method)라고 부릅니다. 이름 끝이 Helper인 static 클래스 안에 헬퍼 메서드를 놓는다면, 가독성이 조금 더 좋아집니다.

윈도우 생성자에 있는 것과 같은 코드입니다. static 헬퍼 메서드 안에 있군요.

마지막 실습에 유용한 정보입니다.

모든 XAML 컨트롤은 System.Windows 네임스페이스의 UIElement 베이스 클래스로부터 상속받습니다. ViewModel에서 UI와 관련된 코드를 위해서 using문을 쓰는 대신 클래스의 몸체에 System.Windows.UIElement 네임스페이스를 명시적으로 사용합니다. 대부분의 클래스가 스프라이트를 확장하는 abstract 클래스인 UIElement 클래스를 사용합니다. 어떤 프로젝트들은 FrameworkElement와 같은 서브 클래스가 좀 더 적합할 수도 있습니다. 왜냐하면 Width, Height, Opacity, HorizontalAlignment 등과 같은 많은 속성들이 정의되어 있기 때문이죠.

5 ViewModel 폴더에 추가한 빈 BeeViewModel 클래스입니다. UI와 관련된 코드를 뷰에 이동함으로써, 뷰모델에 있는 코드가 간단해지고, 벌과 관련된 로직에 관해서만 관리할 수 있습니다.

```
using View;
using System.Collections.ObjectModel;
using System.Collections.Specialized;

class BeeViewModel {
    private readonly ObservableCollection<Windows.UI.Xaml.UIElement>
        _sprites = new ObservableCollection<Windows.UI.Xaml.UIElement>();
    public INotifyCollectionChanged Sprites { get { return _sprites; } }

    public BeeViewModel() {
        AnimatedImage firstBee =
            BeeHelper.BeeFactory(50, 50,
                TimeSpan.FromMilliseconds(50));
        _sprites.Add(firstBee);

        AnimatedImage secondBee =
            BeeHelper.BeeFactory(200, 200, TimeSpan.FromMilliseconds(10));
        _sprites.Add(secondBee);

        AnimatedImage thirdBee =
            BeeHelper.BeeFactory(300, 125, TimeSpan.FromMilliseconds(100));
        _sprites.Add(thirdBee);

        BeeHelper.MakeBeeMove(firstBee, 50, 450, 40);
        BeeHelper.SetBeeLocation(secondBee, 80, 260);
        BeeHelper.SetBeeLocation(thirdBee, 230, 100);
    }
}
```

스프라이트(Sprite)는 2D 이미지나 게임 혹은 애니메이션에서 사용하는 용어입니다.

ItemsControl의 ItemSource 속성에 바인딩되어 있는 _sprites 필드에 AnimatedImage 컨트롤을 추가했을 때, 컨트롤에서 IItemPanelTemplate을 바탕으로 생성된 항목 패설을 추가합니다.

Sprites 속성을 캡슐화하기 위해서 두 가지의 과정이 필요합니다. 나중에 덮어 쓰이지 않기 위해서, 백킹 필드를 readonly로 선언합니다. 그리고 다른 클래스에서는 볼 수 있으나, 수정하지 못하도록 INotifyCollectionChanged 속성으로 노출합니다.

ObservableCollection에 컨트롤을 추가한 뒤, 속성을 변경하고 애니메이션을 추가하고 있습니다. 왜 작동할까요?

6 앱을 실행해 봅시다. 이전과 똑같이 실행되어야 하지만, 이제 각 계층에서 일을 분배합니다. UI와 관련된 코드는 뷰에서, 벌이 이동하는 코드는 뷰모델에 있습니다.

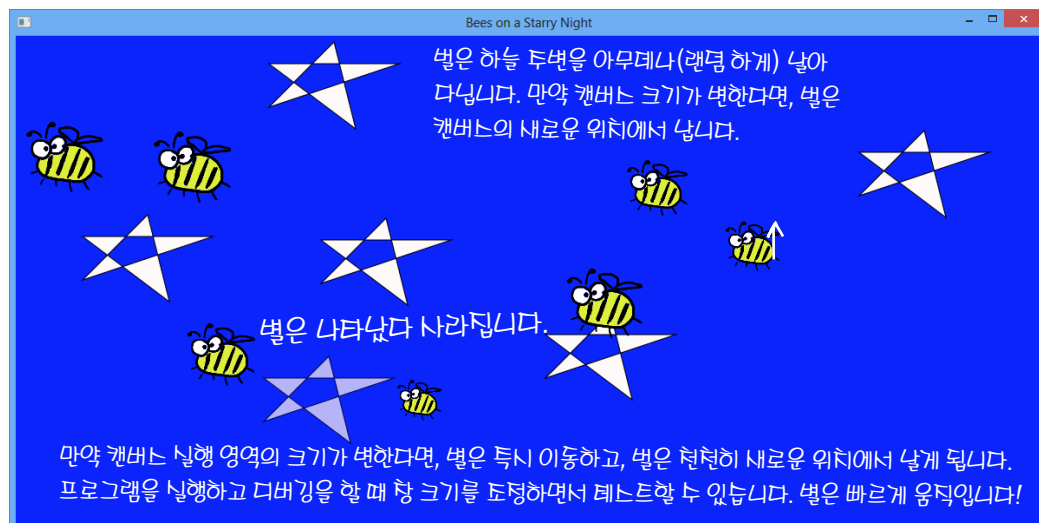
readonly 키워드
캡슐화를 하는 가장 중요한 이유는 실수로 다른 클래스의 데이터를 덮어 쓰는 것을 방지하는 겁니다. 원래 데이터에 뭔가 덮어 쓰는 것을 막을 수 없을까요? 필드에 readonly 키워드가 있다면, 선언이나 생성자에서만 쓸 수 있습니다.



이 책의 마지막 연습문제입니다. 별과 벌을 움직이는 프로그램을 만들어 봅시다. 여기에는 엄청나게 많은 코드가 있습니다. 하지만 여러분이 어떻게 작업하느냐에 따라 코드의 양이 다를 수 있습니다. 그리고 이 긴 연습문제를 끝낸다면, 비디오 게임을 만드는 데 필요한 도구를 얻게 됩니다(실습 3에서 무엇을 할까요?).

1 여러분이 만들어야 할 앱입니다.

어두운 파란색 배경의 캔버스 주위에 벌들이 날아 다닙니다. 그 뒤에 별들이 나타났다가 사라집니다. 이것을 보여 주기 위해서 별과 벌, 페이지를 담은 뷰를 만듭니다. 그리고 벌이 움직이고 별이 반짝일 때, 상태를 파악하고 이벤트를 발생시키는 데 필요한 모델과 뷰와 모델을 연결하는 뷰모델을 만들어 봅시다.



2 새로운 윈도우 스토어 앱 프로젝트를 생성합니다.

새 프로젝트의 이름을 (별이 빛나는) 총총한 밤을 뜻하는 StarryNight으로 합니다. 다음, Model과 View, ViewModel 폴더를 추가합니다. 그리고 ViewModel 폴더에 빈 BeeStarViewModel 클래스를 추가합니다.

3 View 폴더에 기본 페이지를 생성합니다.

MainWindow.xaml을 지워주세요. 그리고 나서 View 폴더에 새 윈도우인 BeesOnAStarryNight.xaml 파일을 추가합니다. BeesOnAStarryNight.xaml의 최상위 레벨에 있는 태그에서 네임스페이스를 추가합니다(프로젝트 이름인 StarryNight과 같아야 합니다).

```
xmlns:viewmodel="clr-namespace:StarryNight.ViewModel"
```

정적 리소스에 ViewModel을 추가하고, 윈도우 이름을 바꿉니다.

```
<Window.Resources>
  <viewmodel:BeeStarViewModel x:Key="viewModel"/>
</Window.Resources>
```

Canvas 컨트롤의 배경이 Blue인 것과 SizeChanged 이벤트 핸들러를 제외하고, 전 프로젝트의 FlyingBees.xaml과 똑같은 윈도우에 대한 XAML입니다.

```
<Canvas Background="Blue" SizeChanged="SizeChangedHandler" />
응용 프로그램을 시작할 때, 새 창을 열기 위해서 App.xaml의 <application> 태그를 수정합니다.
StartupUri="View\BeesOnAStarryNight.xaml"
```

컨트롤의 크기가 변할때, SizeChanged 이벤트가 발생합니다. 새로운 크기에 대한 정보는 EventArgs에 있습니다.

비주얼 스튜디오는 도형을 그리는 데 도와주는 환상적인 도구가 있습니다. C# 프로젝트에서 복사/붙여 넣기 할 수 있는 XAML 도형을 생성하기 위해서 Blend for Visual Studio 2013을 사용합니다. 자세한 내용은 여기에 있습니다. http://msdn.microsoft.com/ko-kr/library/vstudio/jj171012(v=vs.120).aspx

과정 4에 있는 코드는 과정 9의 PlayAreaSize 속성을 만들 때까지 컴파일되지 않습니다. 속성 트리를 생성하여 컴파일되게 할 수 있습니다.

4 윈도우와 앱에 대한 코드-비하인드를 추가합니다.

View 폴더의 BeesOnAStarryNight.xaml.cs에 SizeChanged 이벤트 핸들러를 추가합니다.

```
ViewModel.BeeStarViewModel viewModel;

public BeesOnAStarryNight() {
  InitializeComponent();

  viewModel = FindResource("viewModel") as ViewModel.BeeStarViewModel;
}

private void SizeChangedHandler(object sender, SizeChangedEventArgs e) {
  viewModel.PlayAreaSize = new Size(e.NewSize.Width, e.NewSize.Height);
}
```



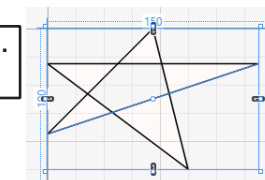
5 View 폴더에 AnimatedImage라는 사용자 정의 컨트롤을 추가합니다.

View 폴더로 돌아가서 AnimatedImage 사용자 정의 컨트롤을 추가합니다. 이 컨트롤은 이번 장 앞에서 본 컨트롤과 같습니다. 프로젝트에 애니메이션 프레임용 이미지 파일을 추가합니다. 그리고 각 파일을 선택한 뒤, 속성 창의 빌드 작업에서 내용을 선택하고, 출력 디렉터리로 복사에서 항상 복사를 선택해 주세요.

6 View 폴더에 StarControl이라는 사용자 정의 컨트롤을 추가합니다.

이 컨트롤은 별을 그립니다. 두 개의 스토리보드에서 하나는 별이 나타나고, 또 하나는 별이 사라집니다. 스토리보드를 작동시키기 위해서 FadeIn()과 FadeOut() 메서드를 코드-비하인드에 추가해 주세요.

Polygon 컨트롤은 다각형을 그리기 위해서 포인트 집합을 사용합니다. UserControl은 Polygon 컨트롤로 별을 그리고 있네요.



```
<UserControl
  // IDE가 생성하는 일반적인 XAML 코드는 아무런 문제가 없습니다.
  // 이 사용자 정의 컨트롤을 위해서 별도의 네임스페이스가 필요하지 않습니다.
  >

  <UserControl.Resources>
    <Storyboard x:Name="fadeInStoryboard">
      <DoubleAnimation From="0" To="1" Storyboard.TargetName="starPolygon"
        Storyboard.TargetProperty="Opacity" Duration="0:0:1.5" />
    </Storyboard>
    <Storyboard x:Name="fadeOutStoryboard">
      <DoubleAnimation From="1" To="0" Storyboard.TargetName="starPolygon"
        Storyboard.TargetProperty="Opacity" Duration="0:0:1.5" />
    </Storyboard>
  </UserControl.Resources>

  <Grid>
    <Polygon Points="0,75 75,0 100,100 0,25 150,25" Fill="Snow"
      Stroke="Black" x:Name="starPolygon"/>
  </Grid>
</UserControl>
```

코드-비하인드에서 스토리보드를 실행시키는 public FadeIn()과 FadeOut() 메서드를 추가합니다. 별이 나타났다가 사라지게 하죠.

starPolygon은 별을 그립니다. 다른 모양으로 그려도 상관없습니다.

타원, 사각형, 다각형을 뛰어넘는 셰이프(모양)에 대해서 조금 더 자세히 알고 싶다면, http://msdn.microsoft.com/ko-kr/library/windows/apps/xaml/hh465055.aspx



7 View 폴더에 BeeStarHelper 클래스를 추가합니다.

헬퍼 클래스입니다. 몇 가지 친숙한 것도 있고, 새로운 것도 보이네요. View 폴더에 클래스를 추가합니다.

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

static class BeeStarHelper {
    public static AnimatedImage BeeFactory(double width, double height, TimeSpan flapInterval) {
        List<string> imageNames = new List<string>();
        imageNames.Add("Bee animation 1.png");
        imageNames.Add("Bee animation 2.png");
        imageNames.Add("Bee animation 3.png");
        imageNames.Add("Bee animation 4.png");

        AnimatedImage bee = new AnimatedImage(imageNames, flapInterval);
        bee.Width = width;
        bee.Height = height;
        return bee;
    }

    public static void SetCanvasLocation(UIElement control, double x, double y) {
        Canvas.SetLeft(control, x);
        Canvas.SetTop(control, y);
    }

    public static void MoveElementOnCanvas(UIElement uiElement, double toX, double toY) {
        double fromX = Canvas.GetLeft(uiElement);
        double fromY = Canvas.GetTop(uiElement);

        Storyboard storyboard = new Storyboard();
        DoubleAnimation animationX = CreateDoubleAnimation(uiElement, fromX, toX, new PropertyPath(Canvas.LeftProperty));
        DoubleAnimation animationY = CreateDoubleAnimation(uiElement, fromY, toY, new PropertyPath(Canvas.TopProperty));
        storyboard.Children.Add(animationX);
        storyboard.Children.Add(animationY);
        storyboard.Begin();
    }

    public static DoubleAnimation CreateDoubleAnimation(UIElement uiElement, double from, double to, PropertyPath propertyToAnimate) {
        DoubleAnimation animation = new DoubleAnimation();
        Storyboard.SetTarget(animation, uiElement);
        Storyboard.SetTargetProperty(animation, propertyToAnimate);
        animation.From = from;
        animation.To = to;
        animation.Duration = TimeSpan.FromSeconds(3);
        return animation;
    }

    public static void SendToBack(StarControl newStar) {
        Canvas.SetZIndex(newStar, -1000);
    }
}
```



Canvas에는 컨트롤의 x축의 위치를 설정하고 얻어오는, SetLeft()와 GetLeft() 메서드가 있습니다. SetTop()와 GetTop() 메서드는 y축을 맡고 있죠. 컨트롤이 Canvas에 추가되고 난 후에 x, y축을 담당하는 메서드들이 실행됩니다.

3초의 DoubleAnimation 을 생성하는 CreateDoubleAnimation() 헬퍼 메서드를 추가합니다. 이 메서드에서 Canvas.Left 와 Canvas.Top 속성을 이동시켜서, 현재 위치에서 새로운 위치로 이동합니다.

ZIndex는 컨트롤이 패널에 놓인 순서를 의미합니다. 높은 ZIndex 의 컨트롤은 낮은 ZIndex 의 위에 그려집니다.



8 Bee, Star, EventArgs 클래스를 Model 폴더에 추가합니다.

모델은 별의 크기와 위치, 별의 위치를 파악해서 이벤트를 발생시킵니다. 그래서 뷰모델은 별이나 별이 언제 바뀌는지 알고 있죠.

```
using System.Windows;
class Bee {
    public Point Location { get; set; }
    public Size Size { get; set; }
    public Rect Position { get { return new Rect(Location, Size); } }
    public double Width { get { return Position.Width; } }
    public double Height { get { return Position.Height; } }

    public Bee(Point location, Size size) {
        Location = location;
        Size = size;
    }
}
```

```
using System.Windows;
class Star {
    public Point Location { get; set; }

    public Star(Point location) {
        Location = location;
    }
}
```

Star 클래스에서 부울 값을 가진 회전 속성인 Rotating을 추가해서, 몇 개의 별이 두위를 천천히 회전하도록 만들어 보세요.

```
using System.Windows;
class BeeMovedEventArgs : EventArgs {
    public Bee BeeThatMoved { get; private set; }
    public double X { get; private set; }
    public double Y { get; private set; }

    public BeeMovedEventArgs(Bee beeThatMoved, double x, double y) {
        BeeThatMoved = beeThatMoved;
        X = x;
        Y = y;
    }
}
```

모델은 EventArgs를 사용하는 이벤트를 발생시켜서, 뷰모델에게 언제 변하는지 알려줍니다.

```
using System.Windows;
class StarChangedEventArgs : EventArgs {
    public Star StarThatChanged { get; private set; }
    public bool Removed { get; private set; }

    public StarChangedEventArgs(Star starThatChanged, bool removed) {
        StarThatChanged = starThatChanged;
        Removed = removed;
    }
}
```

Rect 구조체에는 몇 개의 오버로드된 생성자가 있습니다. 그리고 너비와 높이, 크기, 위치(Point 구조체 혹은 각 X, Y의 두 좌표)를 넘겨 추출해 두는 메서드가 있습니다.

Point, Size, Rect 구조체

System.Windows 네임스페이스는 몇 가지 매우 유용한 구조체가 있습니다. Point 구조체는 X와 Y의 double 속성을 이용해서 좌표 세트를 저장합니다. Size 구조체도 역시 Width와 Height의 double 속성을 가집니다. 또한 특별한 Empty 값도 있습니다. Rect 구조체는 직사각형의 왼쪽 상단과 오른쪽 하단 모서리의 두 좌표를 저장합니다. 다른 Rect 구조체와 그 외의 높이와 너비, 교차 지점을 찾는 많은 유용한 메서드가 있습니다.

Polygon 컨트롤의 Points 속성은 Point 구조체의 컬렉션입니다.



가나긴 연습문제 (계속)

9 BeeStarModel 클래스를 Model 폴더에 추가합니다.

private 필드와 몇 가지 유용한 메서드가 있군요.
여러분이 BeeStarModel 클래스를 마무리해 주세요.



```
using System.Windows;

class BeeStarModel {
    public static readonly Size StarSize = new Size(150, 100);

    private readonly Dictionary<Bee, Point> _bees = new Dictionary<Bee, Point>();
    private readonly Dictionary<Star, Point> _stars = new Dictionary<Star, Point>();
    private Random _random = new Random();

    public BeeStarModel() {
        _playAreaSize = Size.Empty;
    }

    public void Update() {
        MoveOneBee();
        AddOrRemoveAStar();
    }

    private static bool RectsOverlap(Rect r1, Rect r2) {
        r1.Intersect(r2);
        if (r1.Width > 0 || r1.Height > 0)
            return true;
        return false;
    }

    public Size PlayAreaSize {
        // CreateBees()와 CreateStarts() 메서드를 호출하는 set 접근자가 있는 백킹 필드를 추가해주세요.
    }

    private void CreateBees() {
        // 만약 실행 영역이 비어 있다면 return: 합니다. 만약 벌이 있다면, 움직이게 합니다.
        // 그렇지 않다면, 5에서 15사이 크기(50에서 150 픽셀)의 벌을 생성하고,
        // _bees 컬렉션에 추가합니다. 그리고 BeeMoved 이벤트를 발생시킵니다.
    }

    private void CreateStars() {
        // 만약 실행 영역이 비어 있다면 return: 합니다. 만약 벌이 있다면,
        // 각 별들의 위치를 새로운 위치로 설정하고, StarChanged 이벤트를 발생시킵니다.
        // 그렇지 않다면 5에서 10사이로 CreateAStar() 메서드를 호출합니다.
    }

    private void CreateAStar() {
        // 겹치지 않는 위치를 찾아서, 새로운 Star 객체를 _stars 컬렉션에 추가합니다.
        // 그리고 StarChanged 이벤트를 발생시킵니다.
    }

    private Point FindNonOverlappingPoint(Size size) {
        // 다른 벌과 별들이 겹치지 않게 하기 위해서 사각형의 왼쪽 상단의 모서리를 찾습니다.
        // 임의로 몇 개의 Rect 구조체를 만든 다음, LINQ 쿼리를 이용해서 겹치는
        // 벌이나 별을 찾습니다(RectsOverlap() 메서드를 이용하세요)
    }

    private void MoveOneBee(Bee bee = null) {
        // 만약 벌이 없다면, return: 합니다. 만약 bee 매개변수가 null이면, 임의의 벌을 선택합니다.
        // 그렇지 않으면 bee 인자 값을 이용해서 겹치지 않는 위치를 찾아서 벌의 위치를 갱신합니다.
        // 그리고 _bee 컬렉션을 업데이트한 다음, OnBeeMoved 이벤트를 발생시킵니다.
    }

    private void AddOrRemoveAStar() {
        // 동전을 뒤집는 확률로 (_random.Next(2) == 0)으로 CreateAStar() 메서드를 이용해서 벌을 생성하거나 제거하세요.
        // 그리고 OnStarChanged 이벤트를 발생시킵니다. 항상 벌이 5개보다 작으면 하나를 생성하고,
        // 20개보다 많으면 하나를 제거하세요. _stars.Keys.ToList()[_random.Next(_stars.Count)]는 임의의 벌을 찾아줍니다.
    }

    // BeeMoved와 StarChanged 이벤트와 이들을 호출하는 메서드를 추가합니다.
    // 이벤트는 BeeMovedEventArgs와 StarChangedEventArgs 클래스를 사용합니다.
}

```

상수 구조체 값을 생성하기 위해서 readonly를 사용합니다.

Size.Empty는 값이 없는 예약된(Size) Size 값입니다. 실행 영역의 크기가 바뀔 때, 벌과 별을 생성하는 데서만 사용됩니다.

뷰모델은 타이머를 이용해서 Update() 메서드를 주기적으로 호출합니다.

이 메서드는 두 Rect 구조체가 있습니다. 그리고 Rect.Intersect() 메서드를 이용해서 두 구조체가 겹칠 때 true를 반환합니다.

PlayAreaSize는 독성입니다.

만약 메서드가 겹치지 않는 위치를 찾으려고 임의로 1000번을 시도했다면, 실행 영역에 아마 공간이 없다고 판단하기 때문에, 아무 위치나 반환해 줍니다.



10 BeeStarViewModel 클래스를 ViewModel 폴더에 추가합니다.

메서드에 있는 주석을 채워주세요. Model이 어떻게 동작하고, View가 무엇을 하는지 자세히 살펴 봐야 합니다. 그리고 유용한 헬퍼 메서드를 잘 활용하세요.

```
using View;
using Model;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System.Windows;
using DispatcherTimer = Windows.UI.Xaml.DispatcherTimer;
using UIElement = Windows.UI.Xaml.UIElement;

class BeeStarViewModel {
    private readonly ObservableCollection<UIElement>
        _sprites = new ObservableCollection<UIElement>();
    public INotifyCollectionChanged Sprites { get { return _sprites; } }

    private readonly Dictionary<Star, StarControl> _stars = new Dictionary<Star, StarControl>();
    private readonly List<StarControl> _fadedStars = new List<StarControl>();

    private BeeStarModel _model = new BeeStarModel();

    private readonly Dictionary<Bee, AnimatedImage> _bees = new Dictionary<Bee, AnimatedImage>();

    private DispatcherTimer _timer = new DispatcherTimer();

    public Size PlayAreaSize { /* get 및 set 접근자는 _model.PlayAreaSize를 설정하고 반환합니다. */ }

    public BeeStarViewModel() {
        // BeeStarModel의 BeeMove와 StarChanged 이벤트에 이벤트 핸들러를 연결하고,
        // 2초마다 타이머가 틱(tick)합니다.
    }

    void timer Tick(object sender, object e) {
        // 매번 타이머가 틱할 때, _fadedStars 컬렉션에 있는 모든 StarControl 참조를 찾아서
        // 각 _sprites를 제거한 뒤, 스스로 업데이트 할 수 있도록 알려주기 위해서
        // BeeViewModel의 Update() 메서드를 호출합니다.
    }

    void BeeMovedHandler(object sender, BeeMovedEventArgs e) {
        // _bees 디렉터리는 Model의 Bee 객체와 View의 AnimatedImage 컨트롤과 매핑됩니다.
        // 벌이 움직일 때, BeeViewModel은 BeeMoved 이벤트를 발생시켜서
        // 벌이 움직이는 것을 듣고 있는 누군가에게 벌의 새로운 위치를 알려줍니다. 만약 _bees 디렉터리가
        // 이미 벌의 AnimatedImage 컨트롤을 포함하지 않는다면, 새것을 하나 만들어서
        // 마캔버스 위치를 설정하고, _bee와 _sprites 모두 업데이트 해줍니다.
        // 만약 _bee 디렉터리가 컨트롤을 이미 포함하고 있다면, 해당 AnimatedImage 컨트롤을 찾은 다음,
        // 애니메이션과 함께 캔버스의 새로운 위치로 이동시켜 줍니다.
    }

    void StarChangedHandler(object sender, StarChangedEventArgs e) {
        // _stars 디렉터리는 _bee 디렉터리와 같은 방식으로 동작합니다. Star 객체와 StarControl 컨트롤이
        // 매핑되는 것을 제외하고 말이죠. EventArgs는 (Location 속성을 가진) Star 객체의 참조를 포함하고 있습니다.
        // 그리고 만약 벌이 제거될 때 boolean 값으로 알려줍니다. 벌이 사라지는 과정은 _stars로부터 벌을 지우고,
        // _fadedStars에 벌을 추가합니다. 그리고 FadeOut() 메서드를 호출합니다(_sprites에서 벌을 삭제한 다음
        // Update() 메서드가 호출됩니다. 이것은 타이머의 틱 간격 설정이
        // StarControl의 사라지는 애니메이션 시간보다 커야 하는 이유입니다.
        //
        // 만약 벌이 사라지지 않았다면, _starts에서 벌을 포함하는지 확인해봅니다. 사라지지 않았다면,
        // StarControl 참조를 얻고, 그렇지 않다면 벌이 나타나는 새 StarControl을 생성하고,
        // _sprites에 추가해서 다시 벌을 보냅니다. 그래야 벌이 벌 앞에서 날 수 있습니다.
        // 그리고 나서 StarControl의 캔버스 위치를 설정합니다.
    }
}

```

DispatcherTimer와 UIElement는 ViewModel에서 사용하는 System.Windows 네임스페이스에서만 존재하는 클래스입니다. using 키워드는 다른 네임스페이스로부터 한 멤버를 선언하기 위해서 "="를 사용할 수 있습니다.

새로운 Canvas 위치를 설정할 때, 벌이 Canvas에 있더라도 컨트롤은 업데이트되어 있습니다. 이것은 실행 영역의 크기가 바뀔 때, 벌들이 느느로 그 두변을 움직이는 방법입니다.



기나긴 연습문제 정답

메서드가 채워진 BeeStarModel 클래스입니다.

```
using System.Windows;

class BeeStarModel {
    public static readonly Size StarSize = new Size(150, 100);

    private readonly Dictionary<Bee, Point> _bees = new Dictionary<Bee, Point>();
    private readonly Dictionary<Star, Point> _stars = new Dictionary<Star, Point>();
    private Random _random = new Random();
    public BeeStarModel() {
        _playAreaSize = Size.Empty;
    }

    public void Update() {
        MoveOneBee();
        AddOrRemoveAStar();
    }

    private static bool RectsOverlap(Rect r1, Rect r2) {
        r1.Intersect(r2);
        if (r1.Width > 0 || r1.Height > 0)
            return true;
        return false;
    }

    private Size _playAreaSize;
    public Size PlayAreaSize {
        get { return _playAreaSize; }
        set {
            _playAreaSize = value;
            CreateBees();
            CreateStars();
        }
    }

    private void CreateBees() {
        if (PlayAreaSize == Size.Empty) return;

        if (_bees.Count() > 0) {
            List<Bee> allBees = _bees.Keys.ToList();
            foreach (Bee bee in allBees)
                MoveOneBee(bee);
        } else {
            int beeCount = _random.Next(5, 10);
            for (int i = 0; i < beeCount; i++) {
                int s = _random.Next(50, 100);
                Size beeSize = new Size(s, s);
                Point newLocation = FindNonOverlappingPoint(beeSize);
                Bee newBee = new Bee(newLocation, beeSize);
                _bees[newBee] = new Point(newLocation.X, newLocation.Y);
                OnBeeMoved(newBee, newLocation.X, newLocation.Y);
            }
        }
    }
}
```

이들은 문제에서 미리 투여된 거죠.

PlayAreaSize 속성이 언제 변하든지 간에, Model은 백킹 필드인 _playAreaSize를 갱신한 다음 CreateBees()와 CreateStars() 메서드를 호출합니다. 이것은 ViewModel이 Model에게 크기가 변할 때마다 느스로 크기를 변경하라고 알려줍니다.

여기에 벌들이 있다면, 벌은 각각 움직입니다. MoveOneBee()는 각 벌들이 겹치지 않게 하기 위해서 새 위치를 찾고, BeeMove 이벤트를 발생시킵니다.

아직 모델에 벌이 하나도 없다면, Bee 객체를 생성하고, 위치를 설정해 줍니다. 언젠가 벌이 추가되거나 바뀔 때, BeeMoved 이벤트를 발생시켜야 합니다.

```
private void CreateStars() {
    if (PlayAreaSize == Size.Empty) return;

    if (_stars.Count > 0) {
        foreach (Star star in _stars.Keys) {
            star.Location = FindNonOverlappingPoint(StarSize);
            OnStarChanged(star, false);
        }
    } else {
        int starCount = _random.Next(5, 10);
        for (int i = 0; i < starCount; i++)
            CreateAStar();
    }
}

private void CreateAStar() {
    Point newLocation = FindNonOverlappingPoint(StarSize);
    Star newStar = new Star(newLocation);
    _stars[newStar] = new Point(newLocation.X, newLocation.Y);
    OnStarChanged(newStar, false);
}

private Point FindNonOverlappingPoint(Size size) {
    Rect newRect = new Rect();
    bool noOverlap = false;
    int count = 0;
    while (!noOverlap) {
        newRect = new Rect(_random.Next((int)PlayAreaSize.Width - 150),
            _random.Next((int)PlayAreaSize.Height - 150),
            size.Width, size.Height);

        var overlappingBees =
            from bee in _bees.Keys
            where RectsOverlap(bee.Position, newRect)
            select bee;

        var overlappingStars =
            from star in _stars.Keys
            where RectsOverlap(
                new Rect(star.Location.X, star.Location.Y, StarSize.Width, StarSize.Height),
                newRect)
            select star;

        if ((overlappingBees.Count() + overlappingStars.Count() == 0) || (count++ > 1000))
            noOverlap = true;
    }
    return new Point(newRect.X, newRect.Y);
}

private void MoveOneBee(Bee bee = null) {
    if (_bees.Keys.Count() == 0) return;
    if (bee == null) {
        int beeCount = _stars.Count();
        List<Bee> bees = _bees.Keys.ToList();
        bee = bees[_random.Next(bees.Count)];
    }
    bee.Location = FindNonOverlappingPoint(bee.Size);
    _bees[bee] = bee.Location;
    OnBeeMoved(bee, bee.Location.X, bee.Location.Y);
}
```

벌이 있다면, PlayArea에서 벌이 있던 위치에서 새로운 위치로 설정하고, StarChanged 이벤트를 발생시킵니다. 해당 이벤트를 처리하고, 컨트롤을 움직이는 것은 뷰모델의 일입니다.

임의의 Rect 구조체를 만들어서, 겹쳐져 있는지 확인합니다. 오른쪽에 250픽셀의 간격과 아래쪽에 150 픽셀의 간격을 뒤서 벌과 벌이 실행 영역을 벗어나지 않게 합니다.

LINQ 쿼리는 RectsOverlap() 메서드를 호출해서 새로운 Rect와 겹치는 벌 혹은 벌을 찾아줍니다. 만약 둘 중 하나에서 반환되는 숫자 값이 있으면, 새로운 Rect는 뭐가와 겹쳐져 있습니다.

1000번 동안 반복했다면, 아마도 실행 영역에 겹치지 않는 곳이 없을 겁니다. 그래서 무한대의 반복문을 멈춰야 하죠.



가나긴
연습문제
정답

BeeStarModel 클래스의 마지막 멤버들입니다.

```
private void AddOrRemoveAStar() {
    if ((_random.Next(2) == 0) || (_stars.Count <= 5)) && (_stars.Count < 20 )
        CreateAStar();
    else {
        Star starToRemove = _stars.Keys.ToList()[_random.Next(_stars.Count)];
        _stars.Remove(starToRemove);
        OnStarChanged(starToRemove, true);
    }
}

public event EventHandler<BeeMovedEventArgs> BeeMoved;

private void OnBeeMoved(Bee beeThatMoved, double x, double y)
{
    EventHandler<BeeMovedEventArgs> beeMoved = BeeMoved;
    if (beeMoved != null)
    {
        beeMoved(this, new BeeMovedEventArgs(beeThatMoved, x, y));
    }
}

public event EventHandler<StarChangedEventArgs> StarChanged;

private void OnStarChanged(Star starThatChanged, bool removed)
{
    EventHandler<StarChangedEventArgs> starChanged = StarChanged;
    if (starChanged != null)
    {
        starChanged(this, new StarChangedEventArgs(starThatChanged, removed));
    }
}
}
```

동전을 던져서 0이나 1을 무작위로 선택하는
것처럼 되어 있지만, 만약 별이 5개보다 적으면 별을
추가하고, 20개 보다 많다면 별을 제거합니다.

Update() 메서드가 호출될 때마다, 별을 추가하거나
삭제해야 됩니다. CreateAStar() 메서드는 별들을
생성합니다. 만약 하나의 별이 삭제되면, _stars에서 그
별을 제거하고, StarCahgned 이벤트를 발생시켜 줍니다.

평범한 이벤트 핸들러와
이벤트를 발생시키는
메서드입니다.

메서드가 채워진 BeeStarViewModel 클래스입니다.

```
using View;
using Model;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System.Windows;
using DispatcherTimer = System.Windows.Threading.DispatcherTimer;
using UIElement = System.Windows.UIElement;

class BeeStarViewModel {
    private readonly ObservableCollection<UIElement>
        _sprites = new ObservableCollection<UIElement>();
    public INotifyCollectionChanged Sprites { get { return _sprites; } }

    private readonly Dictionary<Star, StarControl> _stars = new Dictionary<Star, StarControl>();
    private readonly List<StarControl> _fadedStars = new List<StarControl>();

    private BeeStarModel _model = new BeeStarModel();

    private readonly Dictionary<Bee, AnimatedImage> _bees
        = new Dictionary<Bee, AnimatedImage>();

    private DispatcherTimer _timer = new DispatcherTimer();
}
```

문제에서 투여된
코드군요.

```
public Size PlayAreaSize {
    get { return model.PlayAreaSize; }
    set { _model.PlayAreaSize = value; }
}

public BeeStarViewModel() {
    _model.BeeMoved += BeeMovedHandler;
    _model.StarChanged += StarChangedHandler;

    _timer.Interval = TimeSpan.FromSeconds(2);
    _timer.Tick += timer_Tick;
    _timer.Start();
}

void timer_Tick(object sender, object e) {
    foreach (StarControl starControl in _fadedStars)
        _sprites.Remove(starControl);

    _model.Update();
}

void BeeMovedHandler(object sender, BeeMovedEventArgs e) {
    if (!_bees.ContainsKey(e.BeeThatMoved)) {
        AnimatedImage beeControl = BeeStarHelper.BeeFactory(
            e.BeeThatMoved.Width, e.BeeThatMoved.Height, TimeSpan.FromMilliseconds(20));
        BeeStarHelper.SetCanvasLocation(beeControl, e.X, e.Y);
        _bees[e.BeeThatMoved] = beeControl;
        _sprites.Add(beeControl);
    } else {
        AnimatedImage beeControl = _bees[e.BeeThatMoved];
        BeeStarHelper.MoveElementOnCanvas(beeControl, e.X, e.Y);
    }
}

void StarChangedHandler(object sender, StarChangedEventArgs e) {
    if (e.Removed) {
        StarControl starControl = _stars[e.StarThatChanged];
        _stars.Remove(e.StarThatChanged);
        _fadedStars.Add(starControl);
        starControl.FadeOut();
    } else {
        StarControl newStar;
        if (_stars.ContainsKey(e.StarThatChanged))
            newStar = _stars[e.StarThatChanged];
        else {
            newStar = new StarControl();
            _stars[e.StarThatChanged] = newStar;
            newStar.FadeIn();
            BeeStarHelper.SendToBack(newStar);
            _sprites.Add(newStar);
        }
        BeeStarHelper.SetCanvasLocation(
            newStar, e.StarThatChanged.Location.X, e.StarThatChanged.Location.Y);
    }
}
```

_fadedStars 컬렉션은 현재 사라지고, 제거된
컨트롤들을 포함하고 있습니다. 그 다음
ViewModel의 Update() 메서드가 호출되죠.

별이 추가될 때 FadeIn() 메서드를 호출해야 합니다.
별이 이미 거기에 있다면, 실행 영역의 크기가 변했기
때문에 별이 새 위치로 이동합니다. 어느 쪽이든 Canvas
의 새 위치로 이동하면 됩니다.



가나긴
연습문제
정답

StarControl의 코드-비하인드에 대한 메서드입니다.

```
using System.Windows.Media.Animation;

public partial class StarControl : UserControl {
    public StarControl()
    {
        InitializeComponent();
    }
    public void FadeIn() {
        Storyboard fadeInStoryboard = FindResource("fadeInStoryboard") as Storyboard;
        fadeInStoryboard.Begin();
    }
    public void FadeOut() {
        Storyboard fadeOutStoryboard = FindResource("fadeOutStoryboard") as Storyboard;
        fadeOutStoryboard.Begin();
    }
}
```

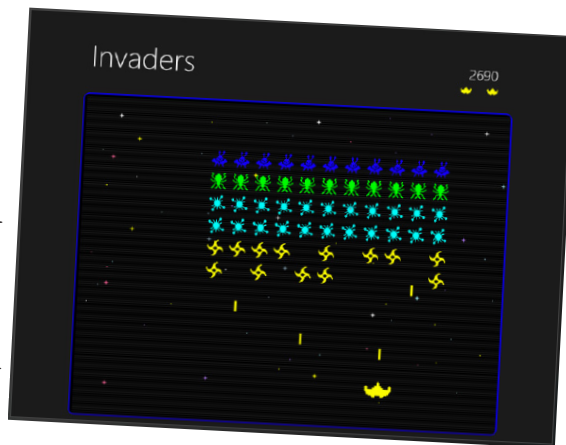
관심사 분리가 잘 되어 있을 때, 설계를 자연스럽게 결합도가 느껴지는 경향이 있습니다.

ViewModel의 PlayAreaSize 속성은 Model의 속성에 전달되지만, Model의 PlayAreaSize의 set 접근자는 BeeMoved와 StartChanged 이벤트를 발생시키는 메서드를 호출합니다. 그래서 화면의 해상도가 변경 될 때 일어나는 일은 다음과 같습니다.

- 1) Canvas는 SizeChanged 이벤트를 발생시킵니다. 2) 이 이벤트는 ViewModel의 PlayAreaSize 속성을 갱신합니다. 3) 이 속성은 Model의 속성을 갱신합니다. 4) 이 속성은 별과 별을 업데이트하는 메서드를 호출합니다. 5) 이 메서드는 BeeMoved와 StarChanged 이벤트를 발생시킵니다. 6) 이 이벤트는 ViewModel의 이벤트 핸들러를 작동시킵니다. 7) 이 이벤트 핸들러는 Sprites 컬렉션을 업데이트합니다. 8) 이 컬렉션은 Canvas에 있는 컨트롤을 갱신시킵니다. 이 예제는 작업을 조정하는 단일의 중앙 객체가 없기 때문에 결합도가 낮습니다. 각각의 객체는 다른 객체들이 어떻게 작동하는지에 대해 명확하게 알 필요가 없기 때문에, 소프트웨어 구축이 매우 안정적입니다. 이벤트를 발생하고, 처리하고, 메서드를 호출하고, 속성을 호출하는 등의 작은 일들만 알고 있으면 됩니다.

실습 3을 위한 모든 도구를 익혔습니다. 인베이더 게임을 만들어 봅시다!

마지막 실습을 위해서 흥미로운 프로젝트를 아껴뒀습니다. 이 책의 마지막 실습에서 고전 게임인 인베이더 게임을 만듭니다. 이 게임은 윈도우 스토어 앱을 위한 실습이지만, 이 책을 다 읽고 잘 이해했다면 인베이더 게임의 WPF 버전을 만들 수 있습니다. WPF에서도 마지막 실습을 위한 필요한 도구를 지원합니다. 다만, 플레이어가 우주선을 움직이는 방법이 다릅니다. 윈도우 스토어 앱은 터치 및 마우스 입력을 처리하는 제스처 이벤트가 있지만, WPF에서는 이러한 이벤트를 지원하지 않습니다. 대신 KeyUp과 KeyDown 이벤트를 사용하면 됩니다. 그리고 4장에서 키 게임에 관련된 예제를 진행했습니다. 인베이더 게임도 이와 같은 방법으로 키를 조작합니다.



축하합니다(아직 끝난 건 아니에요)!

마지막 연습문제를 다 풀었나요? 모든 게 어떻게 동작하는지 이해하셨죠? 그렇다면, 축하드립니다. 여러분은 전체에서 많은 부분의 C#을 배웠고, 여러분이 예상한 것보다는 아마 더 빨리 마쳤을 겁니다. 프로그래밍의 세계는 여러분을 기다립니다.

마지막 실습을 하기 전에, 아직 해야 할 몇 가지 일들이 있습니다. 만약 여러분이 배운 것들을 복습한다면, 머릿속에 그 내용이 오래 남게 될 겁니다.



Save the Humans를 마지막으로 다시 한 번 살펴봅시다.

책에서 우리가 요구한 것들을 다했다면, 여러분은 Save The Humans를 두 번 만들었습니다. 1장과 10장을 시작하기 전에 만들었죠. 심지어 두 번째로 만들 때도 마법처럼 느껴지는 부분이 있습니다. 하지만, **프로그래밍에서 마법이란 없습니다.** 여러분이 만든 코드를 다시 한 번 살펴보면, 얼마나 많이 이해하고 있는지에 대해서 깜짝 놀라게 됩니다. 긍정적으로 여러분의 두뇌에 배운 것들을 익히기만 하면 됩니다.

프로그래밍에서 마법이란 없습니다. 코드를 이해할 수 있고 실행되도록 만들어졌기 때문에, 모든 프로그램이 동작하는 거죠.

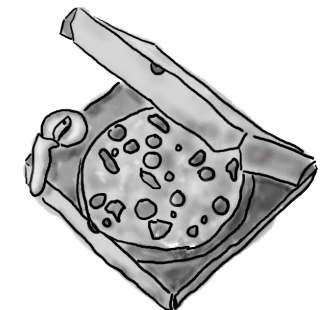


여러분의 친구한테 배운 것을 말해 보세요.

인간은 사회적 동물입니다. 그리고 더 나은 지혜를 사람(사회 구성원)들로부터 배웠습니다. 그리고 이것도 친구들에게 "말하는 것"도 소셜 네트워크를 의미합니다. 그리고 친구들에게 말하면서 뭔가 더 배울 수 있습니다. 친구들한테 배운 것들을 자랑해 보세요.

잠시 휴식을 취합시다. 낮잠을 자면 더 좋고요.

여러분의 두뇌는 많은 정보를 스폰지처럼 빨아 들였습니다. 가끔은 새로운 정보를 흡수하기 위해서 휴면상태로 들어가서 낮잠을 자는 것도 좋은 방법입니다. 수많은 신경과학의 연구에서 뇌의 정보 흡수가 **충분한 숙면을 취한 후** 현저하게 개선되었다는 것을 보여 줍니다. 여러분의 두뇌를 충분히 휴식시켜주세요.



프로그래머가 디자인 패턴과 객체지향 프로그래밍 원리를 잘 사용한다면, 코드가 더 이해하기 쉬워집니다.

