

Xamarin, XAML Programming



자마린(Xamarin)

1.1 자마린 소개?	6
1.2 자마린 설치	6
1.3 자마린 안드로이드(Xamarin.Android)	11
1.4 자마린 특징	13
2. Xamarin.Android	14
2.1 Hello Android(Xamarin.Android Application) Example	14
2.2 Hello Xamarin Android 구조?	26
2.2.1 Resources	27
2.2.2 Xamarin.Android Activity 란?	27
2.2.3 Activity Life Cycle	28
2.2.4 Hello Xamarin Android 의 기타 요소들	33
2.2.5 Xamarin Android Intent(인텐트)	34
2.2.6 Simple Intent Example(웹페이지 오픈하기)	35
2.2.7 Android Service 개요	38
2.2.8 Xamarin.Android Service 생성 및 시작하기	41
2.3 Hello Android MultiScreen Example	43
2.4 Built-In List Item Layouts(내장 리스트아이템 레이아웃)	53
2.5 Xamarin.Android(With .JAR, .AAR, Native Android Library)	62
2.5.1 Binding Java Library(Consuming Java libraries from C#)	62
2.5.2 Xamarin.Android EmbeddedJar Binding(안드로이드 JAR 라이브러리 바인딩)	66
2.5.3 Xamarin.Android .AAR Binding(안드로이드 .AAR File 을 자마린 바인딩 자바 라이브러리로 구현 후 Xamarin.Android 프로젝트에서 호출하기)	77

3. Xamarin.iOS.....	90
3.1 Xamarin.iOS 설치, 개발환경.....	90
3.2 Xamarin.iOS HelloWorld(단일 뷰) 실습.....	90
3.3 Xamarin.iOS HelloWorld 자세히 살펴보기	100
3.3.1 Xamarin.iOS HelloWorld 해부하기	101
3.3.2 Architecture and App Fundamentals.....	104
3.3.3 User Interface(iOS Designer, Storyboards)	105
3.4.4 View Controllers and the View Lifecycle	108
3.3.5 추가적인 사항	111
3.4 Xamarin.iOS HelloWorld(멀티 뷰) 실습	112
3.4.1 Xamarin.iOS HelloWorld(멀티 뷰) 자세히 살펴보기_MVC, Navigation Controller, View Controller	123
3.5 네비게이션 컨트롤러(Navigation Controller).....	127
3.6 루트 뷰 컨트롤러(Root View Controller).....	129
4. Xamarin.Forms.....	130
4.1 Xamarin.Forms Requirements	130
4.2 Xamarin.Forms Quick Start.....	131
4.3 Xamarin.Forms HelloWorld 분석	132
4.3.1 Xamarin.Forms HelloWorld 프로젝트 구조.....	132
4.3.2 Xamarin.Forms HelloWorld Fundamentals.....	132
4.3.3 Xamarin.Forms HelloWorld PCL 및 플랫폼별 코드 분석	133
4.3.4 Xamarin.Forms HelloWorld 사용자 인터페이스	135
4.3.5 Xamarin.Forms HelloWorld User Interaction.....	136
4.3.6 Xamarin.Forms HelloWorld 추가적인 개념.....	137
4.4 Xamarin.Forms Multiscreen Quick Start Example.....	139

4.5 Views And Layout.....	144
4.5.1 Stack Layout	144
4.5.2 Lists in Xamarin.Forms	147
4.5.3 ListView Data Sources	147
4.5.4 Selecting an Item in a ListView	148
4.5.5 DataTemplateSelector.....	149
4.5.6 ListView, DataTemplateSelector Example.....	150
4.6 MVVM 개요(Model/View/ViewModel) 및 MVVM Example	156
4.6.1 ViewModel 을 View 에 연결하기.....	158
4.6.2 Creating a View Model Declaratively	158
4.6.3 Creating a View Model Programmatically.....	159
4.6.4 Xamarin.Forms MVVM HelloWorld(Command Data Binding)	159
4.7 XAML 데이터 바인딩(Data Binding).....	163
4.7.1 데이터 바인딩(Data Binding) 개요	163
4.7.2 View-to-View 데이터 바인딩	167
4.7.3 ListView 심플 데이터 바인딩, 컬렉션 바인딩(Collection Binding), ListView 에서 클릭시 새창 띄우면서 데이터 넘기기.....	168
4.7.4 Backwards 바인딩	175
4.7.5 MVVM 에서 데이터 바인딩 사용하기.....	178
4.7.6 MVVM, ViewModel 을 이용한 ListView 데이터 바인딩.....	181
4.7.7 MVVM, XAML 을 이용한 간단한 계산기 구현	186
4.8 SQLite.Net with Xamarin.Forms.....	198
4.8.1 Local SQLite Access Example	198
4.8 Hierarchical Navigation	215
4.8.1 Pushing Pages to the Navigation Stack	217

4.8.2 Popping Pages from the Navigation Stack.....	217
4.8.3 Passing Data when Navigating.....	218
4.8.4 Hierarchical Navigation Example	221
4.8.5 Login Flow Example	227
5. Xamarin.Forms & REST WebService	237
5.1 Rest service 를 위한 클래스(<code>HttpClient</code> , <code>HttpResponseMessage</code> , <code>HttpContent</code> , <code>HttpWebRequest</code>).....	237
5.2 Xamarin.Forms 안드로이드에서 자바기반 스프링 프레임워크(스프링 부트)로 작성한 웹서비스 호출 실습.....	238
5.3 자마린 앱에서 스프링프레임워크/스프링부트 RESTful 기반 웹서비스 Call 실습, JSON 파싱하기[웹서비스는 자바,스프링으로 모바일 앱은 자마린으로!].....	247

1. 자마린 개요

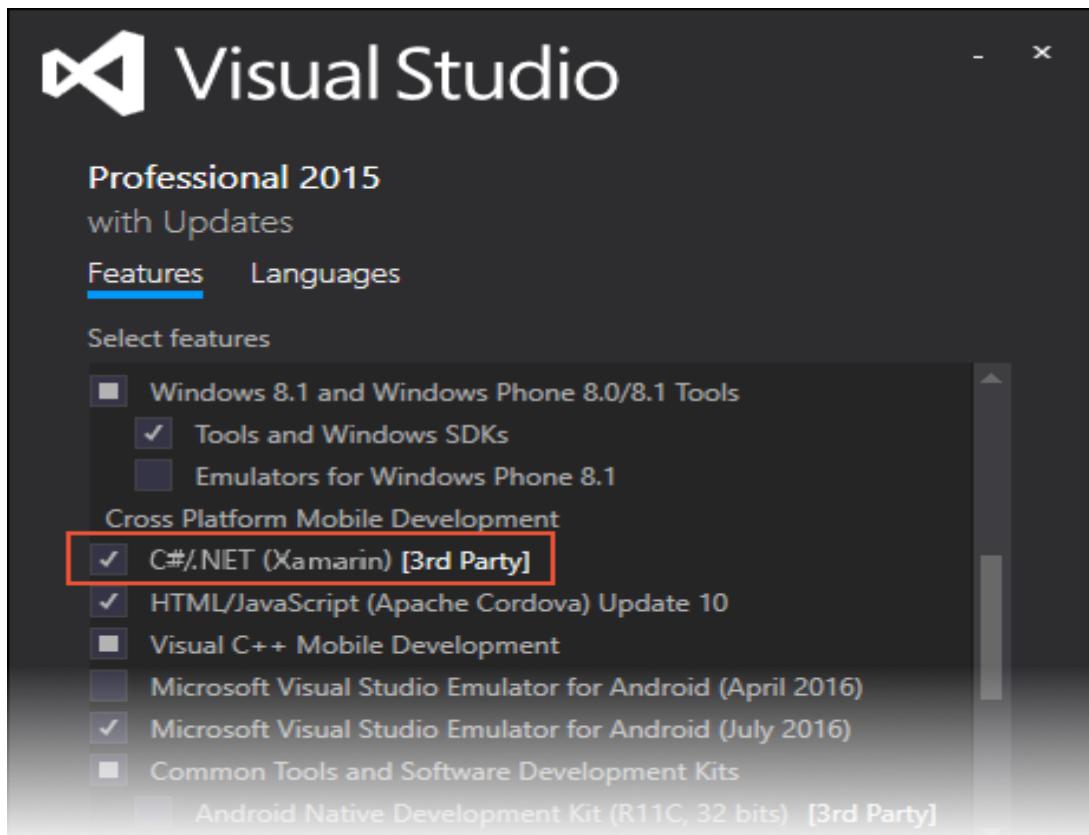
1.1 자마린 소개?

- MonoTouch 6.0 이후로 자마린으로 이름이 변경되었으며 크로스플랫폼(Mac, ios, android, windows 등 다양한 OS를 하나의 소프트웨어가 지원하도록하는 기술)을 지원하는 기술로 C#으로 안드로이드 앱과 iOS 앱, 윈도우폰용 앱을 만들 수 있게 지원한다.
- MS가 크로스플랫폼 개발도구인 “자마린”을 무료로 전환했고 비주얼 스튜디오 사용자라면 누구나 사용가능하고 Mac OS 사용자를 위한 ‘Xamarin Studio For OS X’도 무료로 배포된다.
- **지원 플랫폼**
 - Xamarin.iOS : 아이폰, 아이패드 앱 개발용
 - Xamarin.Android : 안드로이드 앱 개발용
 - Xamarin.Forms : 아이폰, 안드로이드, 윈도우폰, 윈도우10 UWP 까지 개발 가능
 - Xamarin.Mac : Objective-C 및 Xcode에서 개발할 때 사용되는 동일한 OS X 라이브러리 및 인터페이스 컨트롤을 사용하여 C# 및 .NET에서 **Mac 응용 프로그램을 개발**, Xamarin.Mac은 Xcode와 직접 통합되기 때문에 개발자는 Xcode의 인터페이스 빌더를 사용하여 응용 프로그램의 사용자 인터페이스를 만들거나 선택적으로 C # 코드로 직접 만들 수 있다.

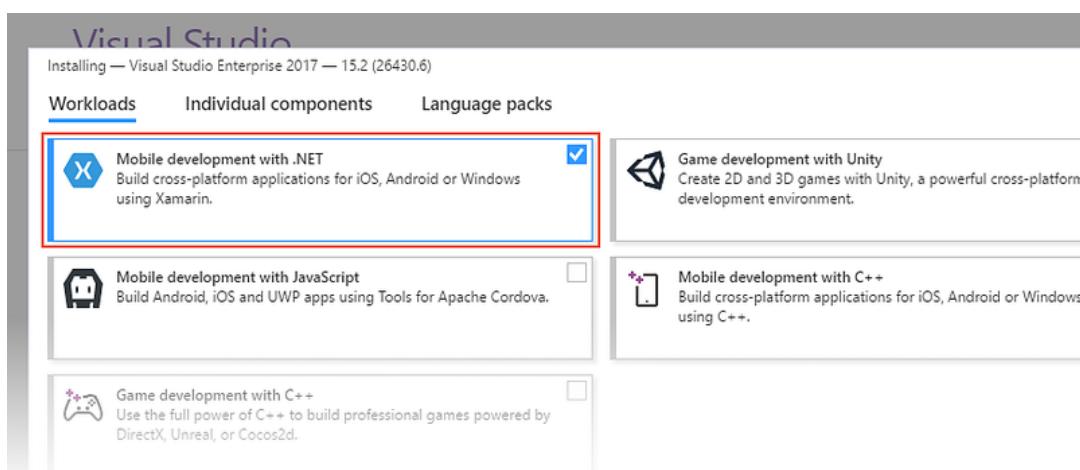
1.2 자마린 설치

■ 설치 환경

- 원도우 10 64Bit
- JDK 설치
- Visual Studio Community 2015, Visual Studio Professional 2015, or Visual Studio Enterprise 2015 설치시 사용자 설치 선택 후 **“Cross Platform Mobile Development”** 선택 후 설치
(안드로이드 SDK 체크 확인)

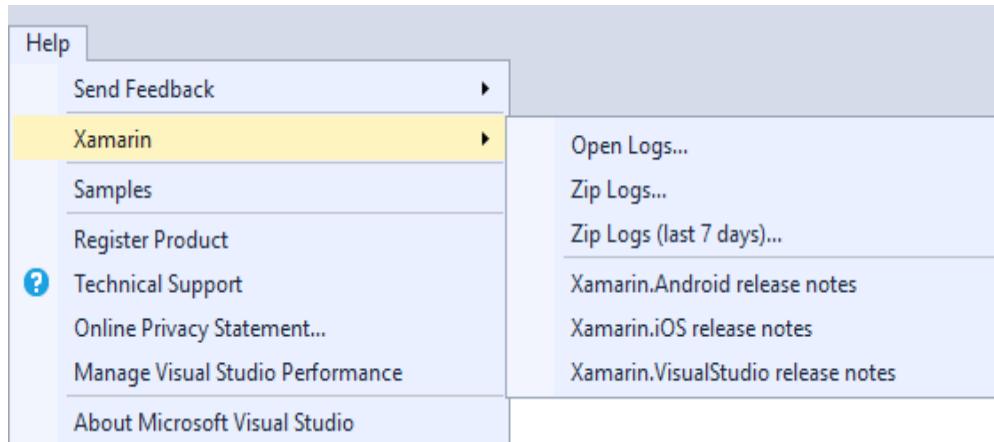


- Visual Studio Community 2017, Visual Studio Professional 2017, or Visual Studio Enterprise 2017 설치시 “Mobile development with .NET” 선택후 설치



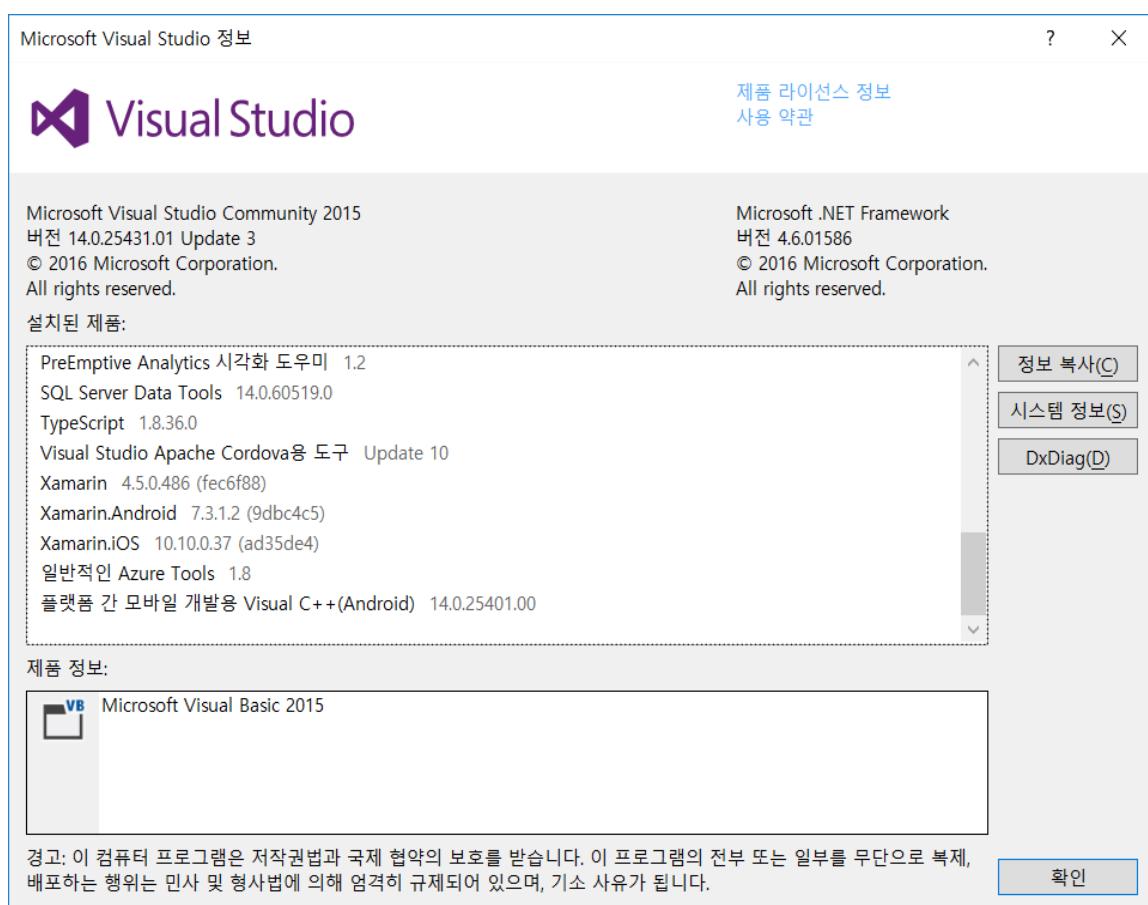
■ 설치 확인

1. Visual Studio 2017



2. Visual Studio 2017 이전

도움말(Help) > About Microsoft Visual Studio



■ 개발 환경

	MAC OS	WINDOWS
Development Environment	VISUAL STUDIO FOR MAC	VISUAL STUDIO
Xamarin.iOS	Yes	Yes (with Mac computer)
Xamarin.Android	Yes	Yes
Xamarin.Forms	iOS & Android only (macOS in preview)	Android, Windows/UWP (iOS with Mac computer)
Xamarin.Mac	Yes	Open project & compile only ^

■ macOS 요구사항(2018.7 월말 기준)

	RECOMMENDED	NOTES
Operating System	macOS High Sierra	Xcode 9 macOS High Sierra 를 필요
Xamarin.iOS	iOS 11 SDK	This iOS 11 SDK ships with Xcode 9.
Xamarin.Android	Android 6.0 / API level 23	최신 SDK 를 사용하면서 기존 Android 버전을 대상으로 하거나, 필요한 경우 이번 버전의 SDK 를 대상으로 빌드할 수 있다.
Xamarin.Forms	MacOS에서 빌드된 Xamarin.Forms 앱은 위의 SDK 요구 사항에 따라 iOS, Android 및 macOS 프로젝트를 포함할 수 있다. Windows/UWP용 Xamarin.Forms 프로젝트는 macOS에서 빌드할 수 없다.	
Xamarin.Mac	macOS High Sierra (10.13 SDK) (10.13 SDK)	MacOS High Sierra (10.13) SDK 와 함께 제공 되 Xcode 9 이며 최신 macOS Api 를 사용 하는 앱을 빌드 하려는 경우 필요하다.

■ macOS에서의 테스팅 & 디버깅

자마린 모바일 응용프로그램은 테스트와 디버깅을 위해 USB를 통해 폰에 배포가 가능하며 Xamarin.Mac 응용 프로그램은 개발 컴퓨터에서 직접 테스트 할 수 있고, Apple Watch 응용 프로그램은 먼저 페어링 된 iPhone에 배포할 수 있다.

TESTING NOTES	
Xamarin.iOS	Xcode에 포함 된 iPhone, iPad, Apple Watch 및 Apple TV 시뮬레이터를 사용하는 것이다.
Xamarin.Android	Xamarin 설치 프로그램에는 테스트를 위해 Google Android 에뮬레이터를 구성 할 수 있는 Google 에뮬레이터 관리자가 포함되어 있다.
Xamarin.Forms	iOS 및 Android 용 Xamarin.Forms 앱은 위에서 설명한대로 관련 플랫폼에 배포 할 수 있다.
Xamarin.Mac	Xamarin.Mac 앱은 개발 컴퓨터에서 직접 테스트 할 수 있다.

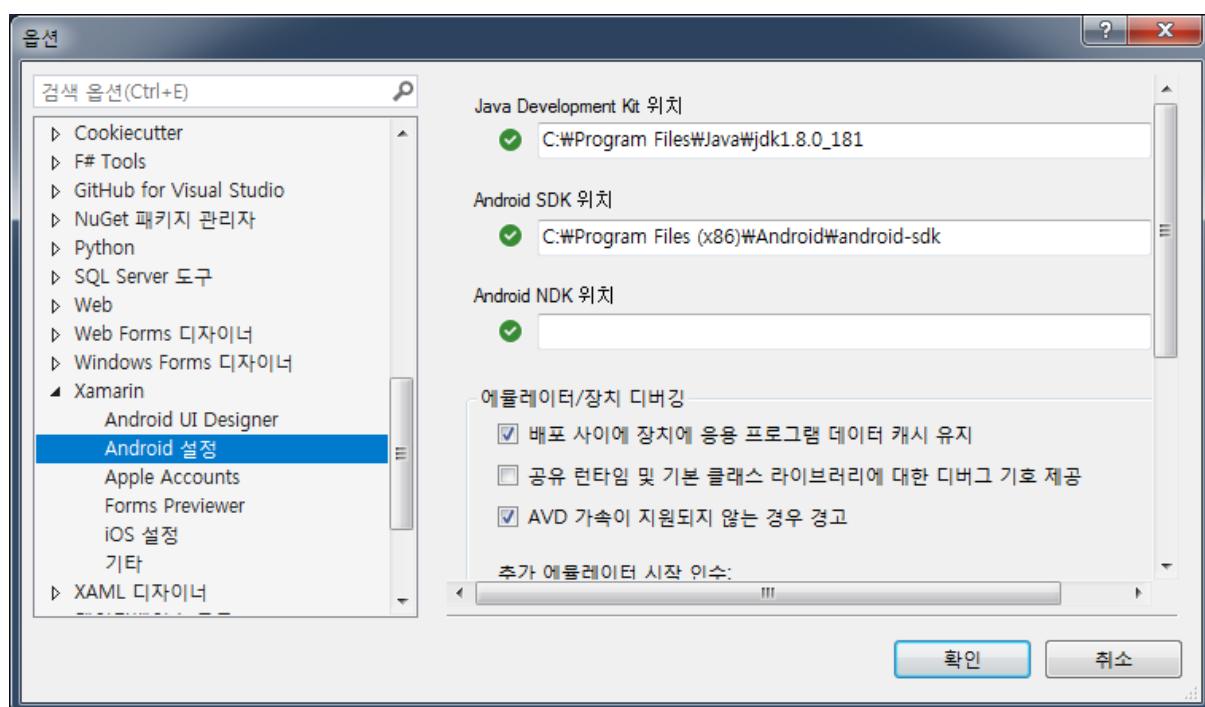
■ 자마린 개발을 위한 윈도우 요구사항(2018년 7월말 기준)

윈도우 환경의 컴퓨터에서 자마린 개발을 위해 아래 소프트웨어/SDK 버전이 필요하다. Visual Studio 2015 및 2017 설치 프로그램에는 Xamarin을 자동으로 설치하는 옵션이 포함되어 있으니 활용하면 좋다. UWP (Universal Windows Platform) 용 Xamarin.Forms 응용 프로그램을 개발하려면 Windows 10에 Visual Studio 2015 또는 2017이 필요하다.

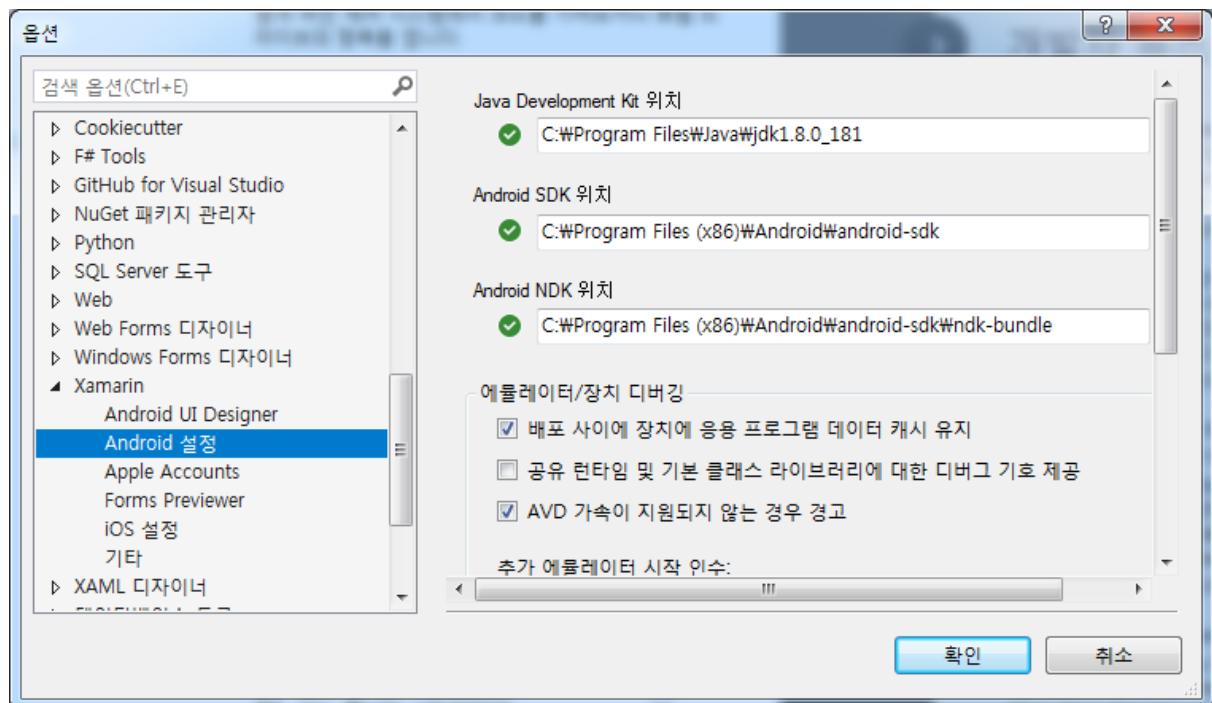
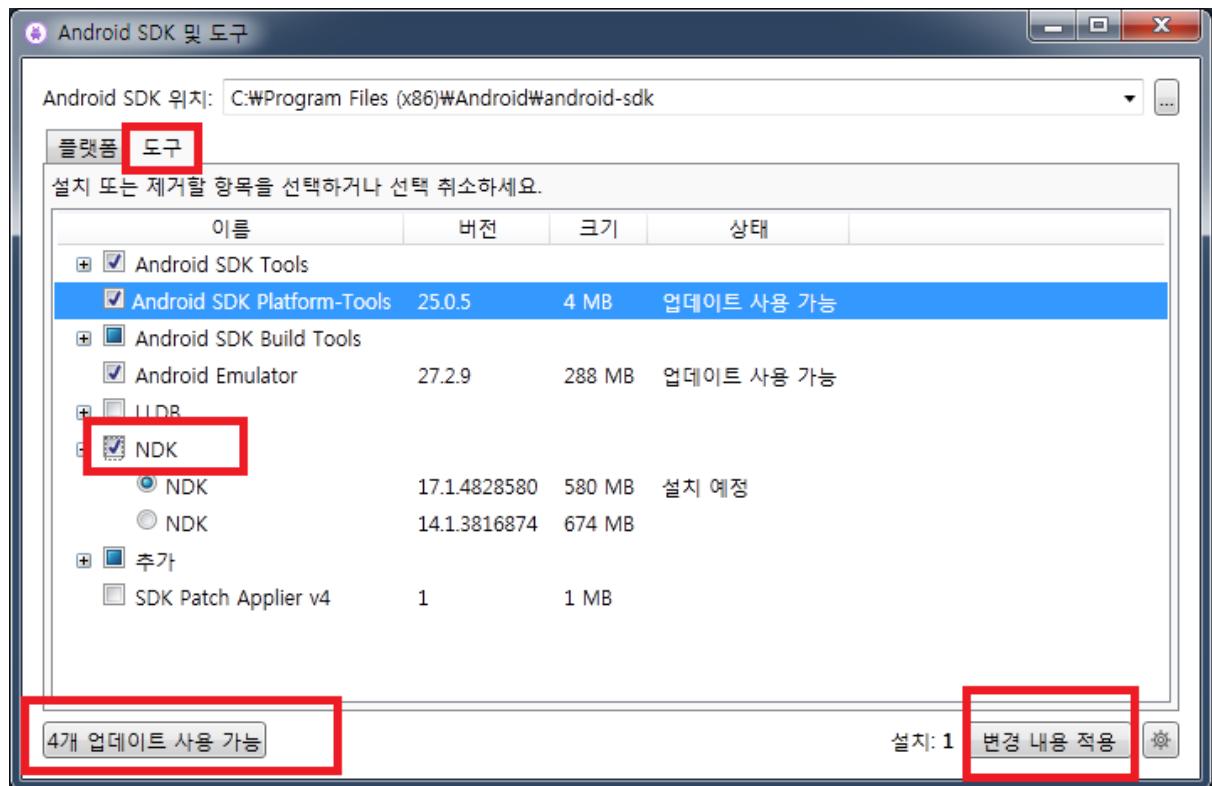
	RECOMMENDED	NOTES
Operating System	Windows 10	최소 운영 체제 버전은 Windows 7. Xamarin.Forms는 Windows 8.1이 필요하며 Xamarin.Forms UWP용 위해 윈도우 10 필요.
Xamarin.iOS	iOS 10 SDK installed on a Mac	Windows에서 iOS 프로젝트를 작성하려면 Visual Studio 2013 이상 및 MacOS에서 Xamarin을 실행하기 위한 최소 요구 사항을 준수하는 Windows 컴퓨터에서 네트워크 액세스 가능한 Mac 컴퓨터.
Xamarin.Android	Android 6.0 / API level 23	최신 SDK를 사용하는 중에도 이전 Android 버전을 타겟팅하거나 필요할 경우 이전 버전의 SDK에 대해 빌드 할 수 있다.
Xamarin.Forms	iOS 및 Android 용 Xamarin.Forms 앱은 위방법대로 관련 플랫폼에 배포 할 수 있으며 Visual Studio에서는 MS의 에뮬레이터를 사용하여 Windows 용 응용 프로그램과 Windows 10 용 Universal Windows Platform을 테스트 할 수 있다. Windows 응용 프로그램은 개발 컴퓨터에서 직접 테스트 할 수 있다.	
Xamarin.Mac	Xamarin.Mac 프로젝트 (macOS 데스크탑 응용 프로그램)는 Visual Studio에서 오픈할 수 있으며 컴파일 할 수 있지만 Visual Studio에서 배포 할 수 없다.	

1.3 자마린 안드로이드(Xamarin.Android)

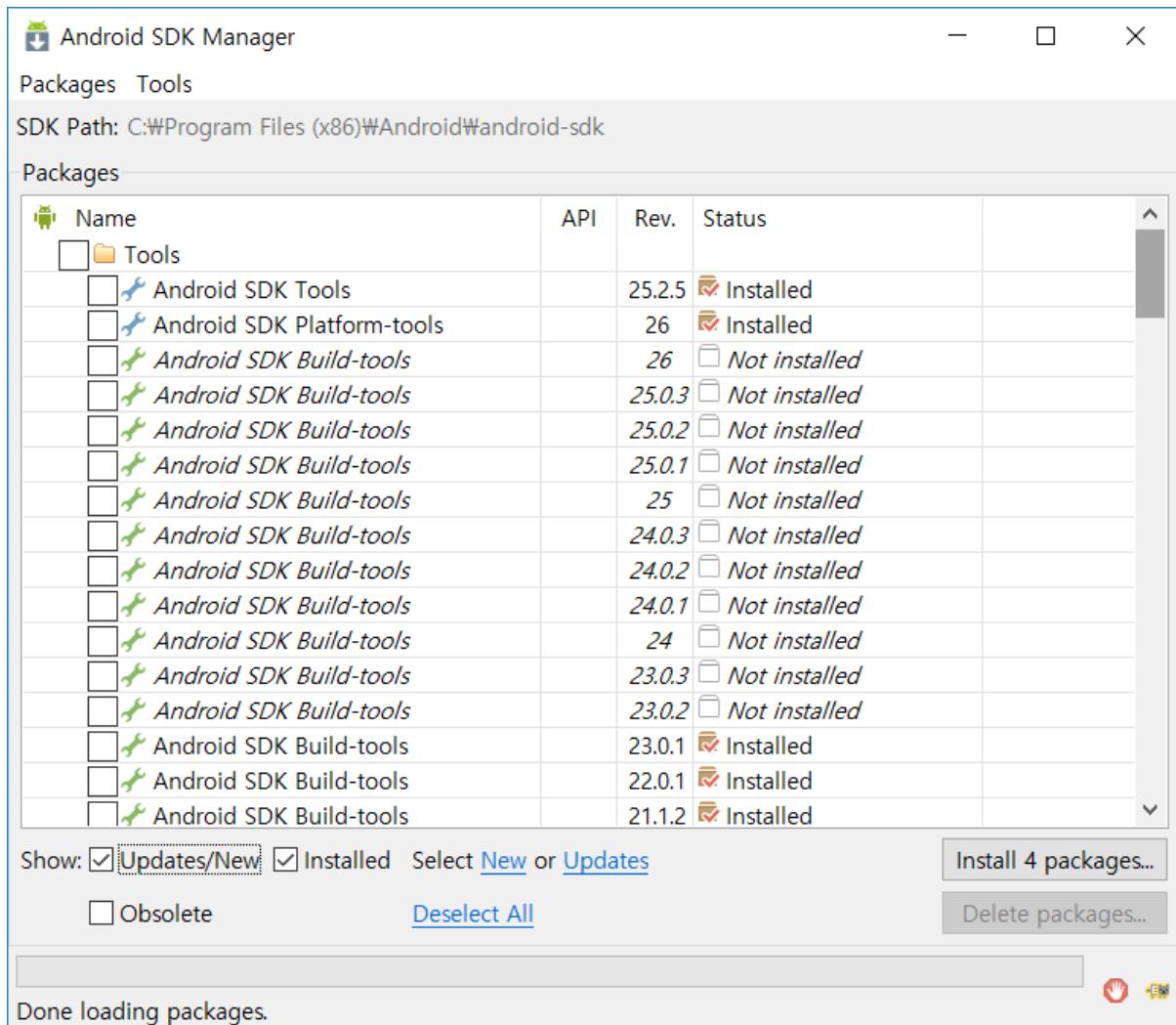
- Visual Studio 설치 관리자를 사용하여 Xamarin.Android 를 다운로드하고 설치할 수 있다.
- 설치는 1.2 절 참조
- 요구사항
 - Window7 이상
 - Visual Studio 2015 or 2017 (Community, Professional, or Enterprise)
 - Xamarin for Visual Studio.
- Xamarin.Android는 Java JDK와 Android SDK를 사용하여 앱을 만드는데 자마린을 설치하는 동안 Visual Studio 설치 관리는 이러한 도구를 기본 위치에 배치하고 적절한 경로 구성을 사용하여 개발 환경을 구성한다. 도구> 옵션> Xamarin> Android 설정을 클릭하여 위치를 보고 변경할 수 있다.



혹시 NDK가 설치되지 않았다면 도구 >> Android >> Android SDK Manager를 실행



- Android는 Android API 레벨 설정을 사용해서 다양한 Android 버전에서 앱 호환성을 결정하는데 타겟팅 할 Android API 수준에 따라 추가 Android SDK 구성 요소를 다운로드하여 설치해야 한다. 또한 Android SDK에 제공되는 선택적 도구 및 에뮬레이터 이미지를 설치해야 할 수 있는데 이렇게 하려면 Android SDK 관리자를 사용하면 된다. 도구 > Android > Android SDK 관리자를 클릭하여 Android SDK 관리자를 시작할 수 있다.



1.4 자마린 특징

- Xamarin은 C# 언어, 클래스 라이브러리(BCL)와 iOS, 안드로이드 그리고 윈도우 폰(윈도우 폰은 네이티브 언어가 이미 C#이다.) 이 세가지 크로스 모바일 플랫폼에서 작동하는 런타임이란 점에서 독특한 위치에 있으며 iOS와 안드로이드의 플랫폼 SDK들을 거의 완전하게 Xamarin에서 바인딩할수 있다.
- Xamarin은 Objective-C, Java, C, and C++ 라이브러리를 직접 호출하는 기능을 제공하고 이미 만들어져있는 강력한 3rd party의 기능들을 사용 할수 있게 해주며 Objective-C, Java, C/C++로 작성된 기존의 iOS, 안드로이드 라이브러리를 사용할 수가 있다.
- 모바일 앱 개발자는 자마린같은 크로스플랫폼을 이용하기보다 네이티브 플랫폼을 대체로 선호한다, 예를 들어 안드로이드 개발자는 JAVA를 이용하고, IDE는 아클립스나 안드로이드 스튜디오를 주로 활용한다. iOS 개발자는 오브젝티브C나 스위프트를 이용하고 IDE는 Xcode를 주로 이용한다. 이런 문화가 정착된 과정에서 모바일 개발자가 자마린같은 크로스플랫폼으로

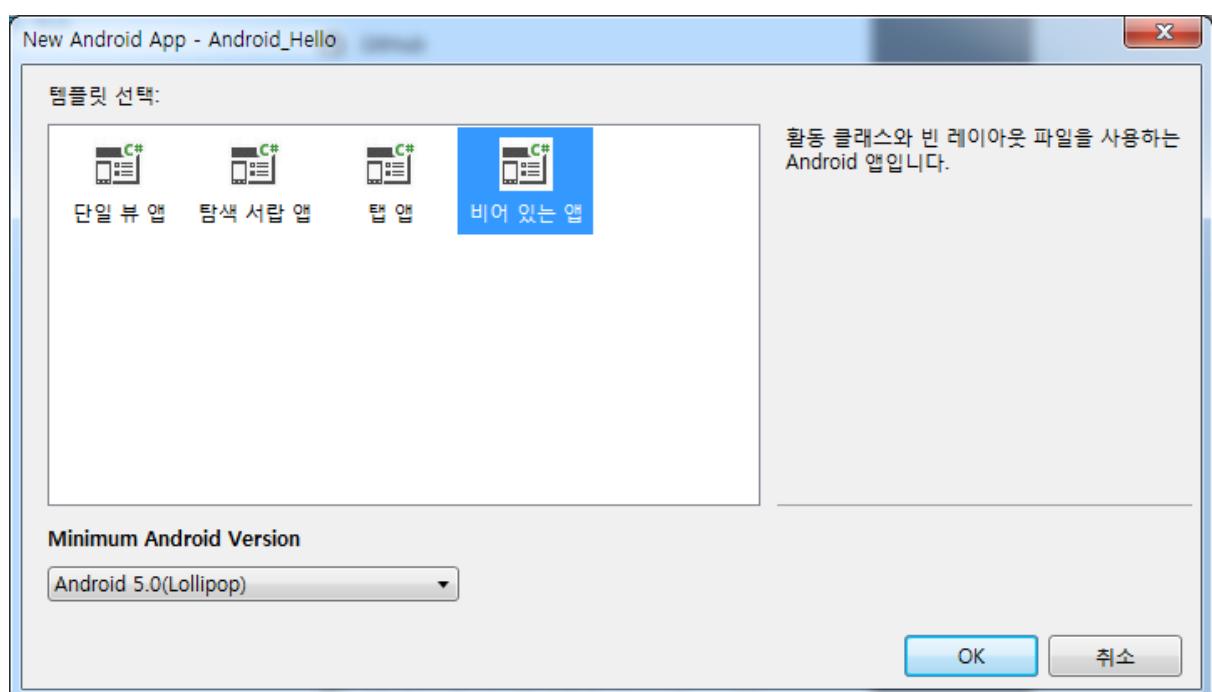
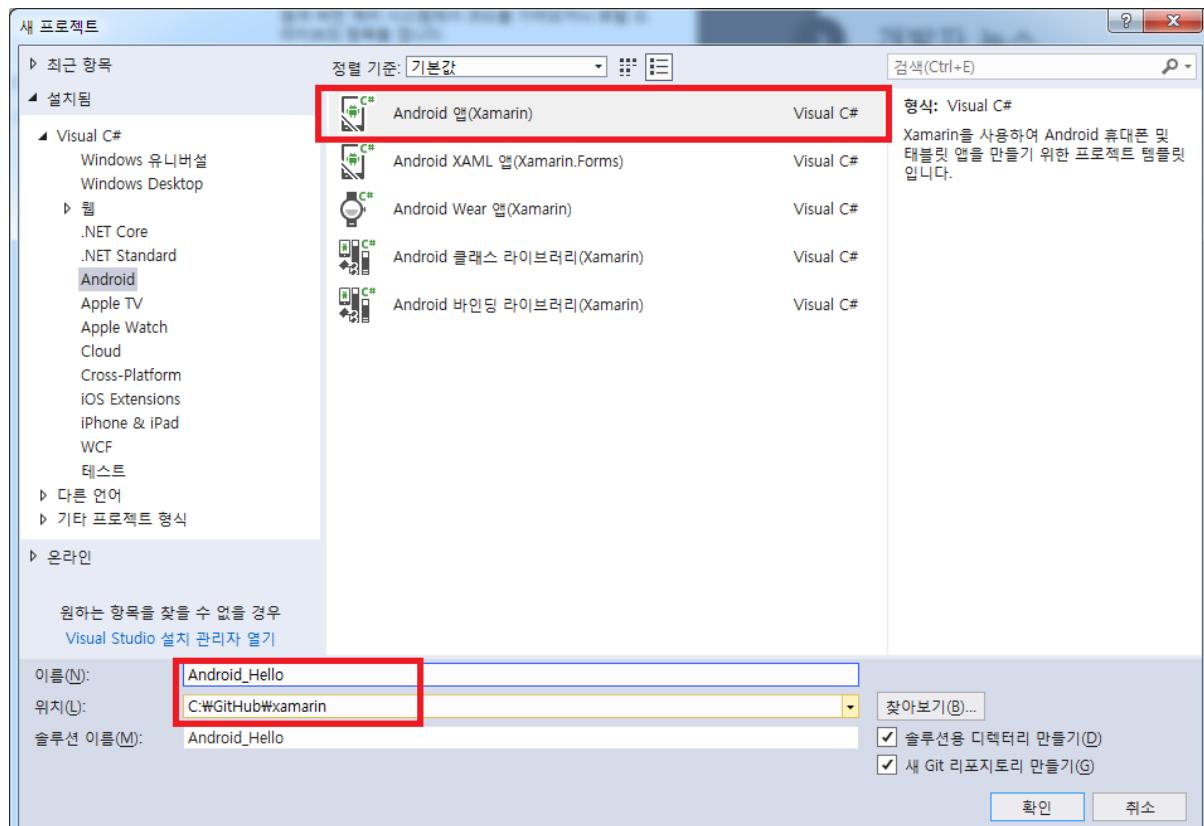
넘어갈지는 아직 미지수이지만 편한 개발방법을 제공하는 것은 사실이며 자마린은 네이티브 형태로 디플로이 되므로 성능면에서도 이점이 있다.

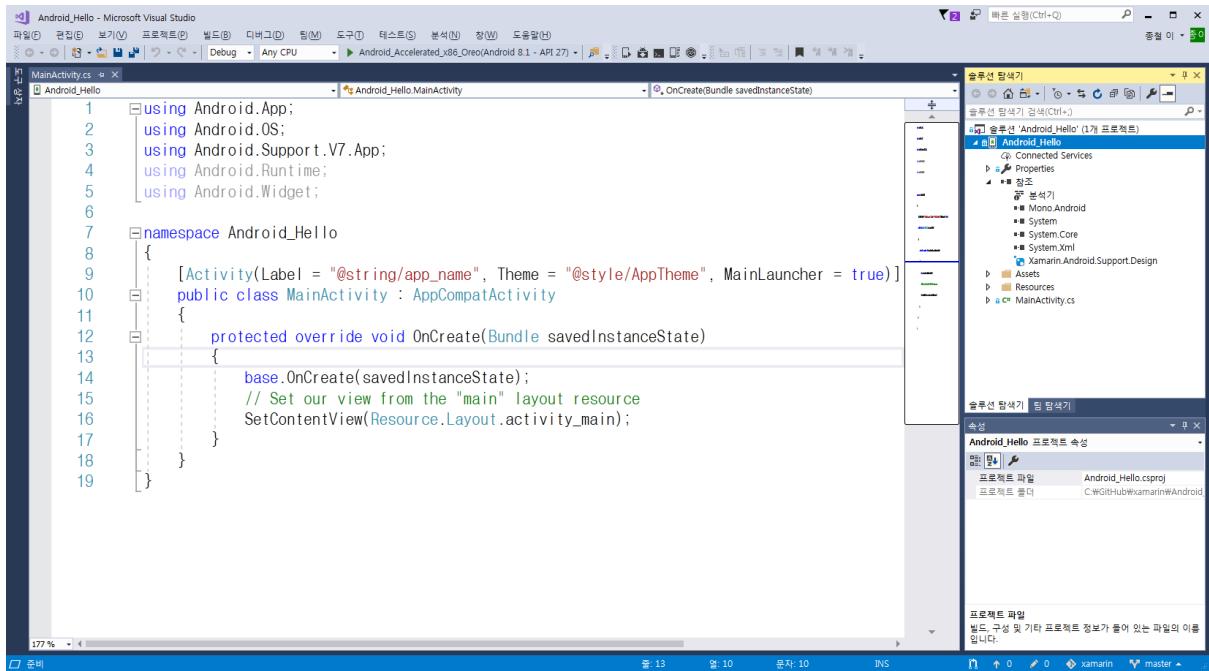
- iOS 기반에서 개발할 때 자마린 스튜디오 또는 비주얼 스튜디오중 어느것을 사용해 코딩하던지 간에 iOS 기반의 애플 맥킨토시 컴퓨터가 있어야 한다. Xamarin 어플리케이션은 단넷 BCL 기반이고 C#으로 작성되었지만 Xamarin.iOS는 컴파일 하기 위해 iOS SDK와 Xcode가 필요하며 iOS 디바이스의 시뮬레이터는 iOS SDK의 한 부분으로 Mac 환경에서만 작동한다.
- C#으로 작성된 Xamarin 어플리케이션은 Objective-C 와 Java의 동적 언어 기능 , 람다식 , 링크(LINQ) , 병렬 프로그래밍 기능, 제네릭 등 사용 가능하다.
- Xamarin 어플리케이션은 .NET BCL을 사용하여 강력한 XML, Database, Serialization, IO, String, and Networking 지원과 같은 편리하고 유연한 기능들을 가지고 있다. 추가적으로 기존 C# 코드들을 어플리케이션에서 사용 가능하도록 컴파일 될수도 있고 BCL에서 커버되지 못한 기능들도 사용 할 수 있게 수천개의 라이브러리들의 액세스를 제공한다.
- Xamarin은 Mac OS X에서는 Xamarin Studio를, 윈도우에서는 Xamarin Studio 또는 Visual Studio를 사용한다. 이 두개의 코드 자동 완성, 세련된 프로젝트와 솔루션 관리 시스템, 편리한 프로젝트 템플릿 라이브러리, 통합된 소스 컨트롤 그리고 기타 여러가지 기능들을 포함하는 현대적인 IDE이다.
- Xamarin은 세개의 메이저 모바일 플랫폼인 iOS, 안드로이드, 윈도우 폰을 지원하는 세련된 크로스 플랫폼 지원을 제공하고 어플리케이션들은 90%이상의 코드를 공유하도록 작성될 수 있으며 Xamarin.Mobile 라이브러리는 세개 플랫폼의 공용 리소스에 액세스할 수 있게 통일된 API를 제공한다.
- Xamarin 어플리케이션이 컴파일 될때, 결과물은 어플리케이션 패키지인데 iOS에서는 .app파일로 안드로이드에서는 .apk파일로 나오게 된다.

2. Xamarin.Android

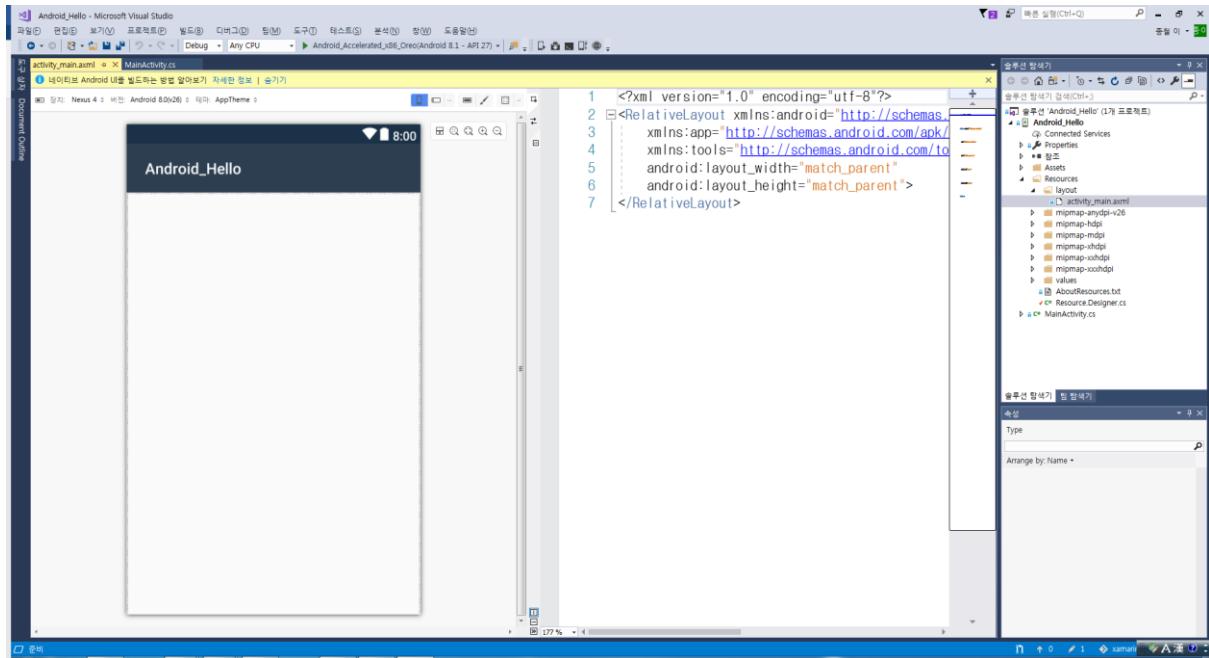
2.1 Hello Android(Xamarin.Android Application) Example

- Xamarin.Android를 이용하여 전화번호를 입력받은 후 폰의 전화걸기 앱을 호출하여 전화거는 기능을 구현해 보자.
- Visual Studio를 실행하여 파일 -> 새로만들기 -> 프로젝트를 클릭 후 좌측 템플릿에서 Android -> 비어있는 앱(Blank App)을 선택하고 프로젝트 이름을 "**Android_Hello**"라고 입력을 하자.

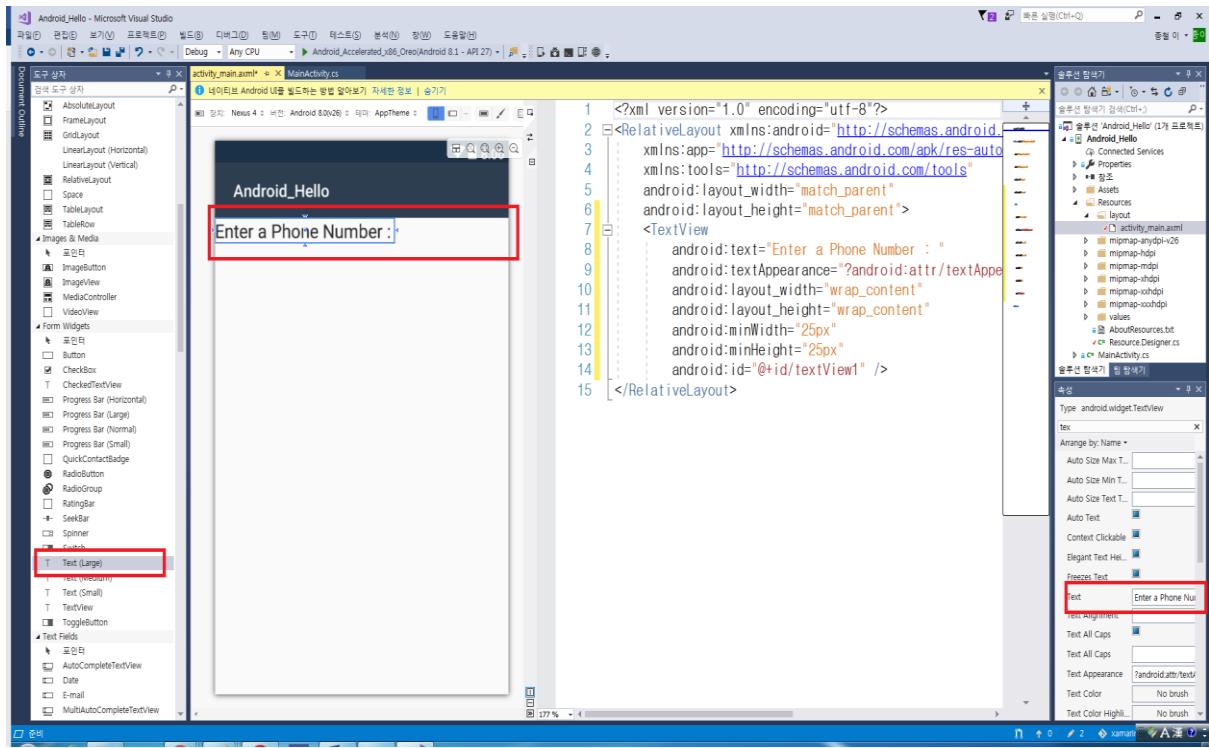




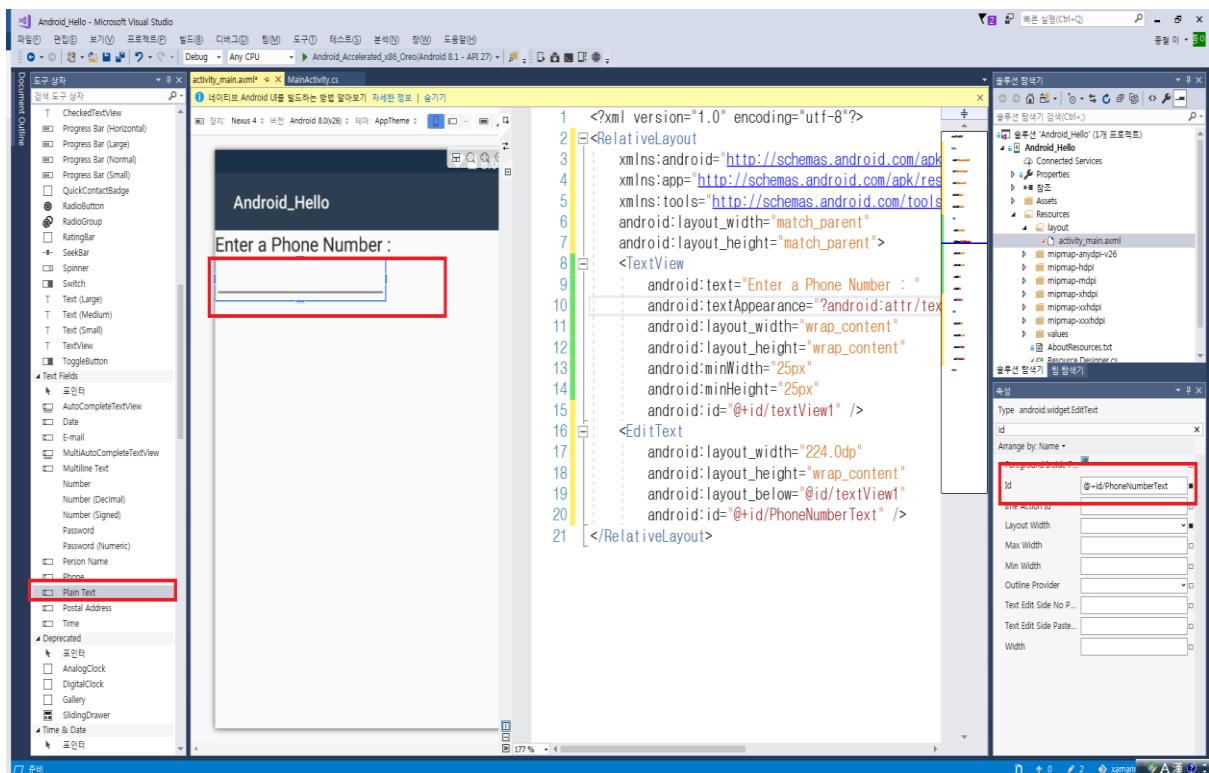
- 프로젝트가 생성되면 우측 솔루션 탐색기의 **Resources** -> **Layout** 아래의 **activity_main**을 더블 클릭하면 폰화면이 나타나는데 이 파일이 스크린의 Layout 파일이다.



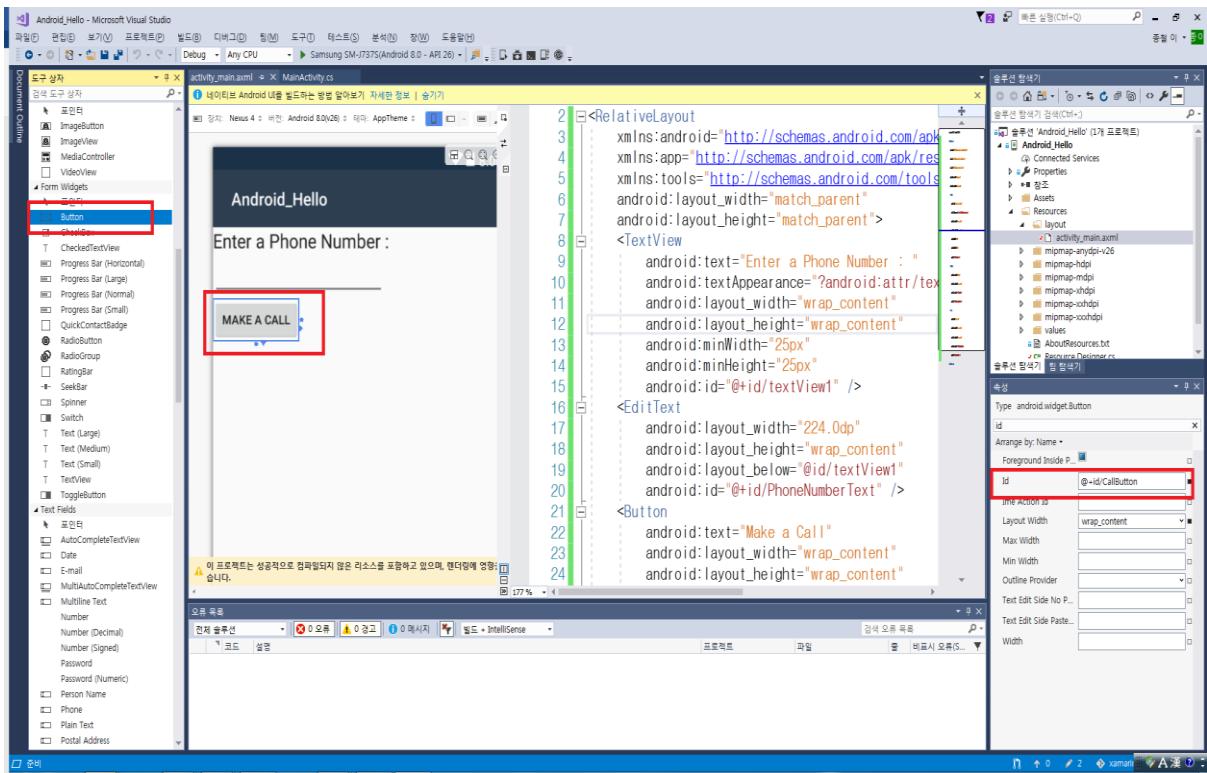
- 화면 좌측에 도구상자(ToolBox)가 보이지 않으면 **보기메뉴 -> 도구상자**를 클릭해서 좌측에 위치시킨 후 **Text(Large)**를 드래그 해서 폰화면의 상단 가운데 위치 시키고 우측하단 속성창에서 **Text 속성을 "Enter a Phone Number :"**로 입력하자.



- 툴박스(ToolBox)에서 **Plain Text** 위젯을 선택 후 Text(Large) 박스 아래에 위치 시키고 **id 속성을 "@+id/PhoneNumberText"**로 설정하자. (드래그해서 위치 시킬 때 마우스 커서가 화살표 모양이 될 때 컨트롤을 놓아야 한다)



- 도구상자에서 **Button**을 선택해서 Plain Text 아래에 위치시키고 **id 속성을 "@+id/CallButton"**, **Text 속성을 "Make a Call"**로 설정하자.



- "Make a Call" 버튼과 연관된 기능을 구현하는데 솔루션 탐색기에서 MainActivity.cs 파일을 오픈하고 **OnCreate** 메소드에서 전화걸기 버튼의 기능을 구현하자.

[MainActivity.cs]

```
using Android.App;
using Android.OS;
using Android.Support.V7.App;
using Android.Widget;
using Android.Text;
using System;
using Android.Content;

namespace Android_Hello
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        protected override void OnCreate(Bundle savedInstanceState)
        {
```

```

base.OnCreate(savedInstanceState);
// Set our view from the "main" layout resource
SetContentView(Resource.Layout.activity_main);

EditText phoneNumberText =
    FindViewById<EditText>(Resource.Id.PhoneNumberText);

//전화걸기 버튼
Button callButton = FindViewById<Button>(Resource.Id.CallButton);

callButton.Enabled = false;

phoneNumberText.TextChanged +=
    (object sender, TextChangedEventArgs e) =>
{
    if (!string.IsNullOrWhiteSpace(phoneNumberText.Text))
        callButton.Enabled = true;
    else
        callButton.Enabled = false;
};

callButton.Click += (object sender, EventArgs e) =>
{
    //Make a Call 버튼 클릭시 전화를 건다.
    var callDialog = new Android.App.AlertDialog.Builder(this);
    callDialog.SetMessage("Call " + phoneNumberText.Text + "?");

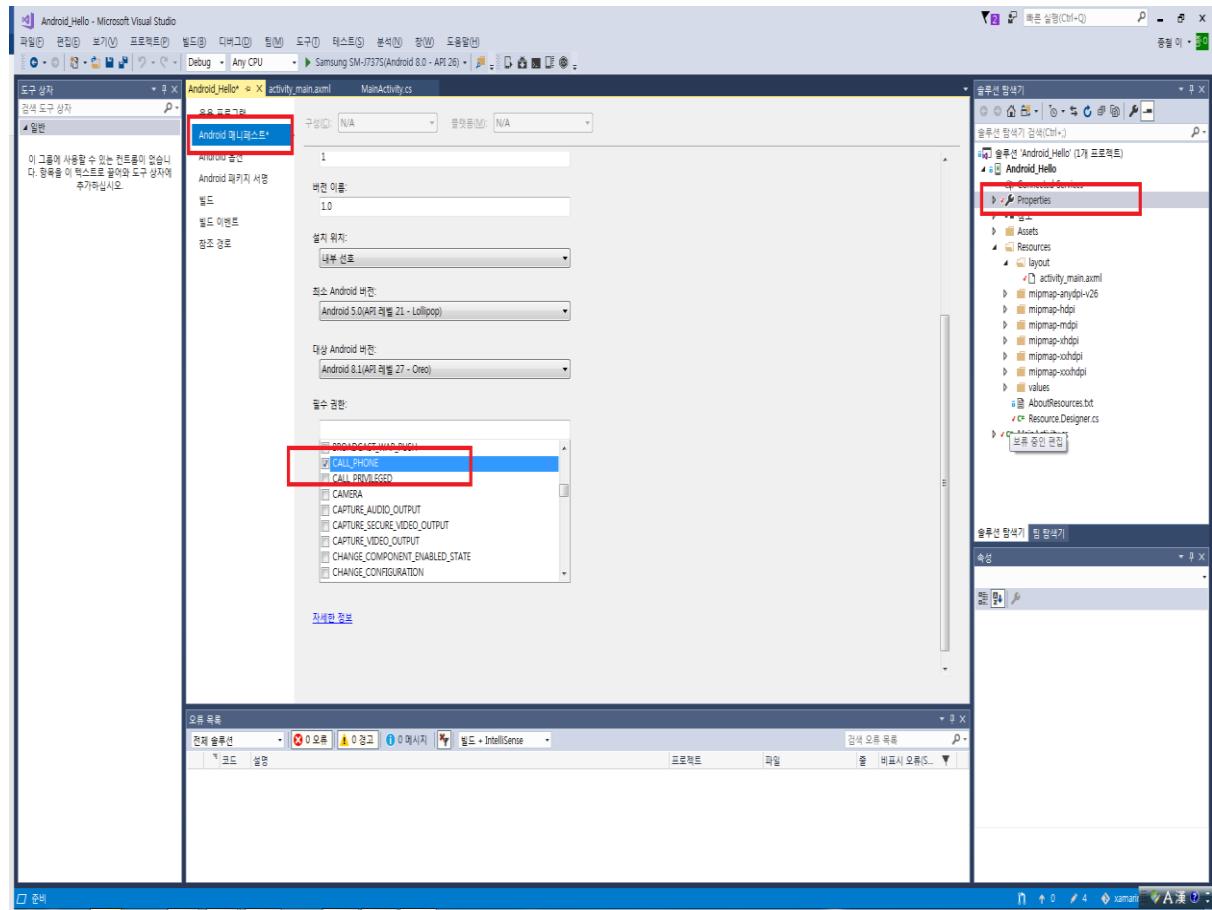
    // "Call"을 클릭하는 경우
    // 전화걸기 위한 인텐트 생성
    callDialog.SetNeutralButton("Call", delegate
    {
        // 인텐트는 액티비티의 전환이 일어날 때 호출하거나 메시지를 전달하는 매개체
        // 암시적 인텐트 : 전환될 곳을 직접 지정하지 않고 액션을 적어서 사용한다.
        // 명시적 인텐트 : 전환될 액티비티를 직접 적어서 표현하는 방법을 사용한다.
        var callIntent = new Intent(Intent.ActionCall);
        callIntent.SetData(Android.Net.Uri.Parse("tel:" +
phoneNumberText.Text));
        StartActivity(callIntent);
    });
    //Cancel을 클릭하는 경우
    callDialog.SetNegativeButton("Cancel", delegate { });

    callDialog.Show();
};
}
}

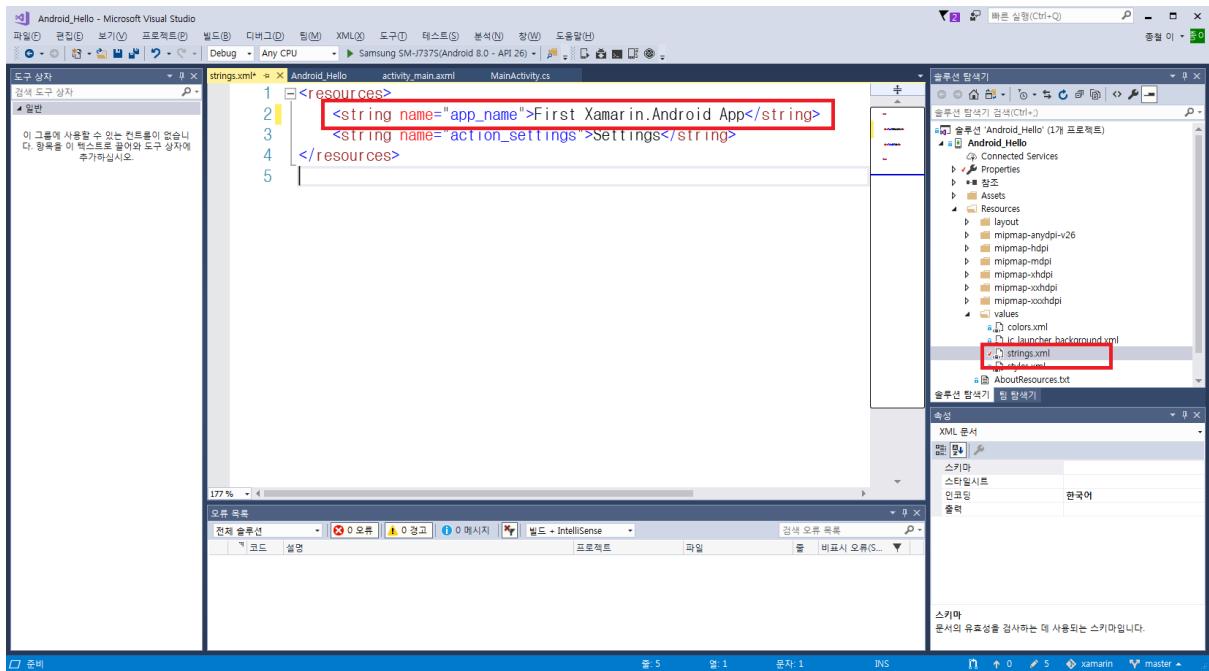
```

{}

- Application에 전화거는 기능을 활성화 시키자. Android Manifest안에 권한과 관련된 부분을 정의해야 하는데, 솔루션 탐색기에서 **Properties**를 더블클릭하고 좌측 Android Manifest를 선택 후 하단의 **Permission**중 “**CALL_PHONE**” permission을 활성화 하자.



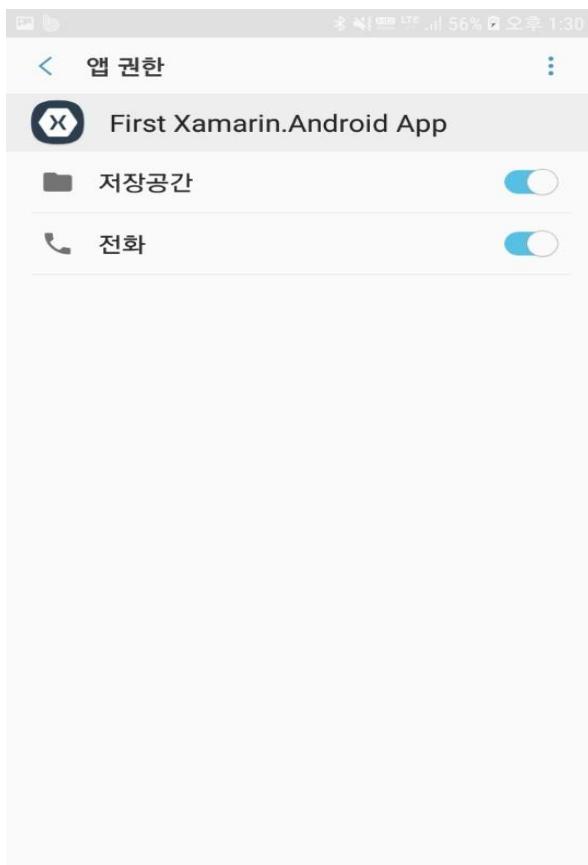
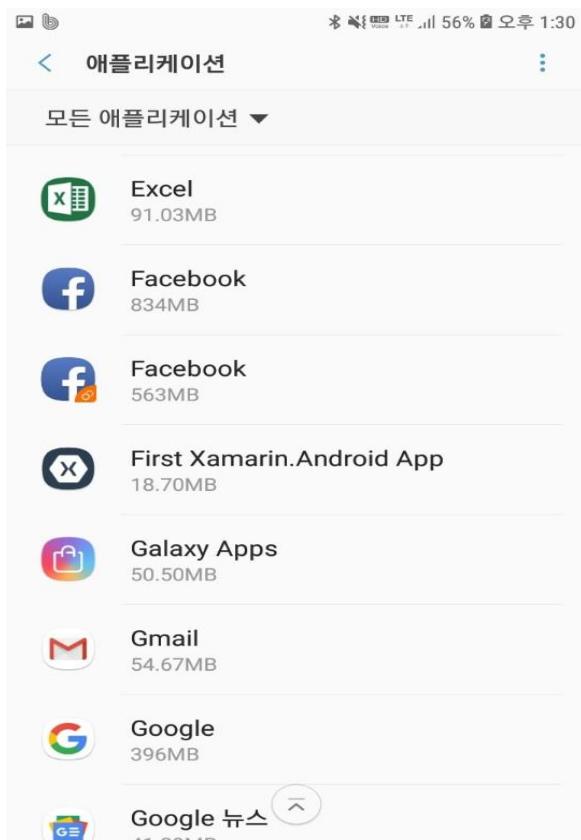
- 애플리케이션 이름을 지정하자. 기본적으로 프로젝트명으로 되어 있는데 “First Xamarin.Android App”으로 변경하자. (Resources/values/strings.xml)



- 모든 변경사항을 저장(Ctrl+Shift+S)하고 빌드하자.(Build > Rebuild Solution)
- MainActivity.cs의 Label은 응용프로그램의 스크린 최상단에 디스플레이되는 텍스트, 응용프로그램의 이름이니 필요하다면 수정을 하면 된다.

```
[Activity(Label = "Xamarin Android", MainLauncher = true, Icon = "@drawable/icon")]
public class MainActivity : Activity {
```

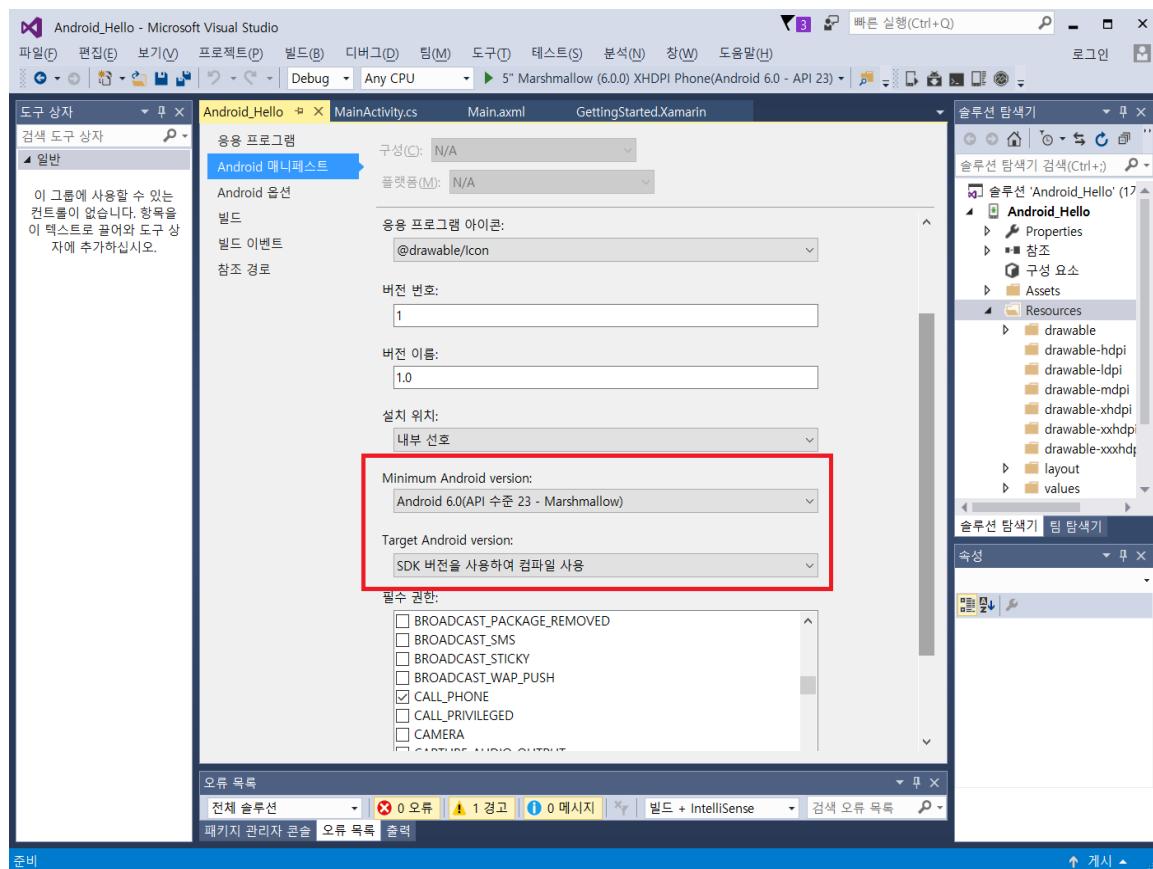
- 실습시 전화걸기가 권한/보안 오류로 인해 작동하지 않는다면 **설정 -> 애플리케이션 관리 -> 애플리케이션 관리자**를 실행하여 해당 앱을 선택하고 **권한 -> 전화**를 선택하여 권한을 주도록 하자.

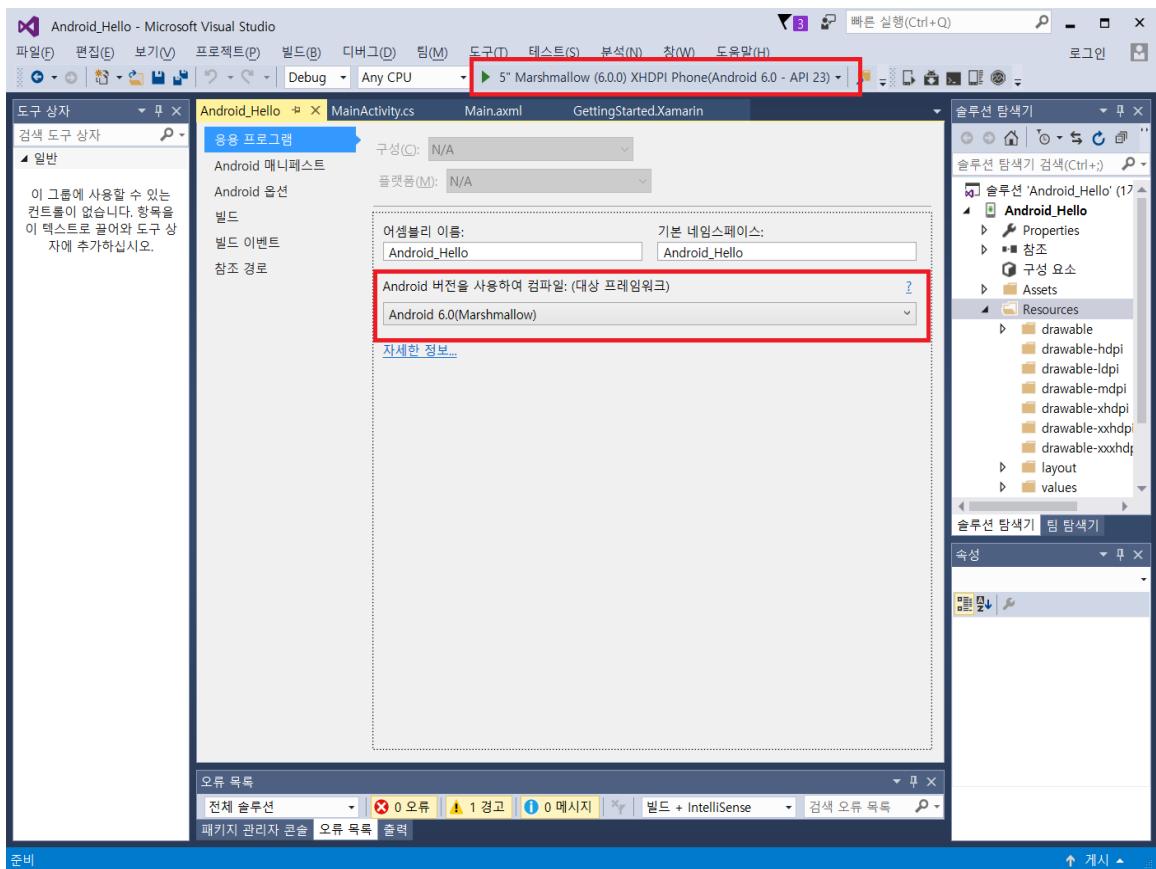


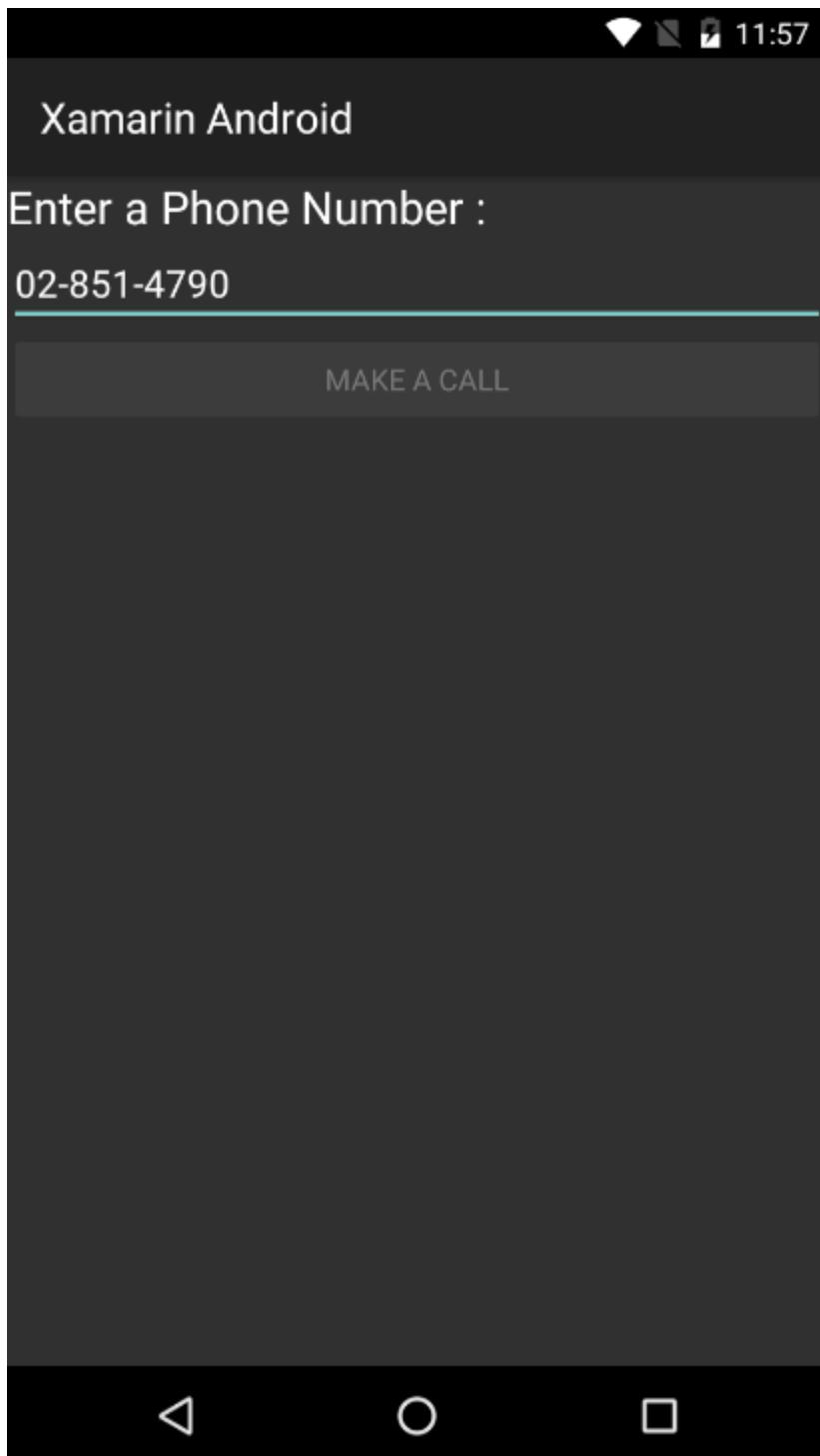
■ 에뮬레이터, 또는 안드로이드폰으로 테스트를 해보자.

실제 폰에서 테스트 하길 권고하며 USB 연결을 했는데도 상단에 본인의 “휴대폰 모델명”이 보이지 않으면 개인의 폰에 맞는 USB Driver를 설치 후 다시 USB를 연결하자.

Properties를 더블클릭 후 좌측 Application에서 API LEVEL 23(mashimallow)로 설정하고 이를 레이터를 실행했다. 상단의 실행단추를 눌러 실행하자.



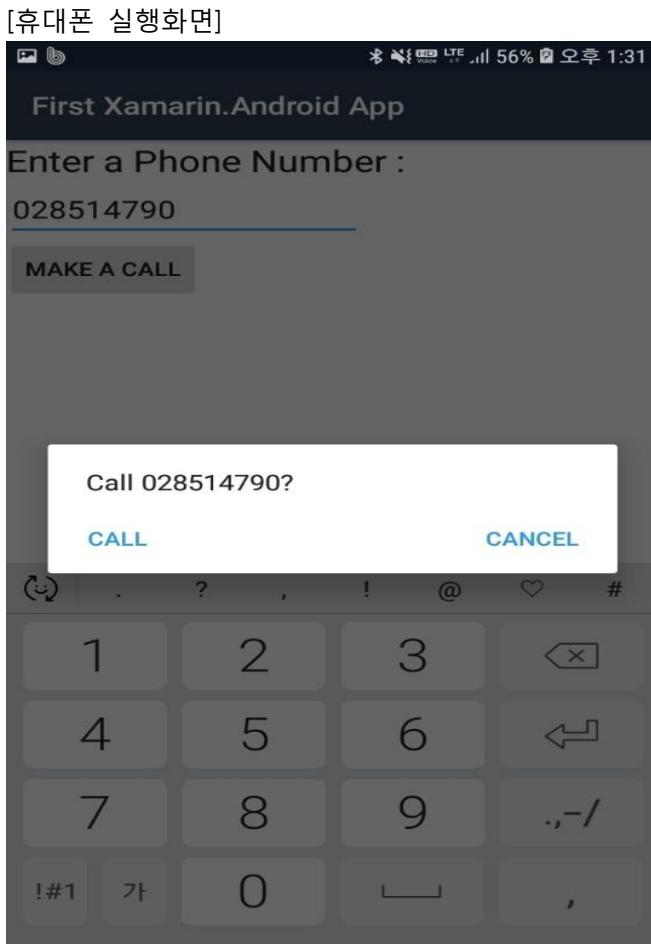




[에뮬레이터 실행화면]

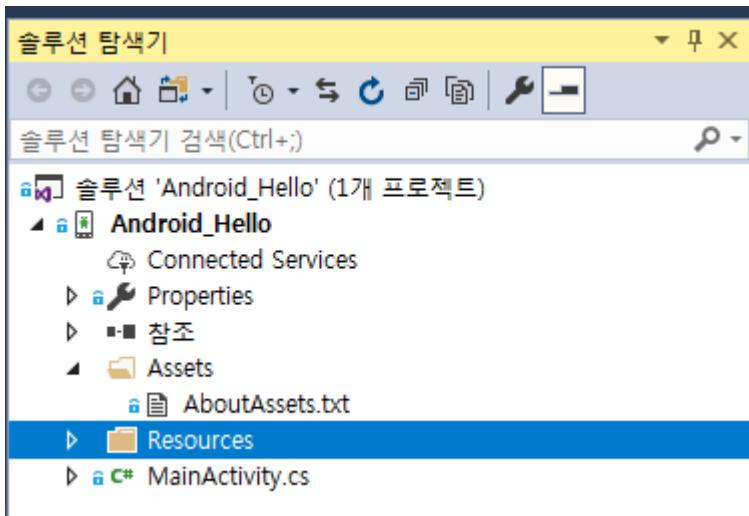
- 폰으로 테스트 하기 위해서는 휴대폰을 USB로 연결 후 폰에서 USB 디버깅을 활성화 해야 되는데 휴대폰 상단 설정을 클릭하고 더보기 -> 개발자옵션 ->USB 디버깅을 체크하자. 또는 갤럭시 S4이후, 안드로이드4.2 젤리빈 이후의 폰들은 개발자 옵션이 비활성화 되어있으므로 설정 -> 디바이스 정보 -> 빌드번호 를 7번 클릭하면 “개발자 모드가 실행되었습니다”라는 메시지가 나오면서 개발자 모드가 활성화 된다. 개발자 모드를 활성화하면 USB 디버깅도 같

이 활성화 하자.



2.2 Hello Xamarin Android 구조?

- **Properties** : 응용프로그램 이름, 버전, 접근권한과 같이 Xamarin.Android Application 요구사항들을 설명하는 AndroidManifest.xml을 포함하고 있으며, 자기자신 어셈블리를 서술하는 AssemblyInfo.cs 파일도 제공한다.
- **참조** : 응용프로그램이 참조하는 내부, 외부 어셈블리(.DLL)을 지정한다. References 디렉터리를 확장하면 System, System.Core 및 System.Xml과 같은 .NET 어셈블리에 대한 참조는 물론 Xamarin의 Mono.Android 어셈블리에 대한 참조가 표시된다.
- **Assets** : 글꼴, 로컬 데이터 파일 및 텍스트 파일을 비롯하여 응용 프로그램이 실행해야하는 파일이 들어 있으며 여기에 포함 된 파일은 생성 된 Assets 클래스를 통해 액세스 할 수 있습니다.
- **Resources** : 문자열, 이미지 및 레이아웃과 같은 응용 프로그램 자원을 포함하며 생성 된 Resource 클래스를 통해 코드에서 이러한 리소스에 액세스 할 수 있다.



2.2.1 Resources

- **Resources** 디렉토리는 Resource.Designer.cs파일과 drawable, layout, mipmap, values의 폴더를 담고 있다.
- **drawable** : 이미지, 비트맵등 drawable resources를 제공한다.
- **layout** : 스크린 또는 액티비티용 User Interface를 정의하는 Android designer 파일들(.axml) 담고 있다. 안드로이드 템플릿에서는 activity_main이라는 기본 레이아웃을 제공해 준다.
- **Resources.Designer.cs** : Resource class로 알려져 있는 이 파일은 각 리소스에 할당된 unique ID를 담고 있는 부분 클래스(Partial Class) 파일이다. Xamarin.Android tools가 자동으로 생성 하며 필요할때 재생성된다. 이 파일은 직접 수정할수 없고 Xamarin.Android에서 UI가 변경변 발생될 때 마다 자동으로 수정된다.

2.2.2 Xamarin.Android Activity란?

- 사용자에게 UI를 제공하는 메인이 되는 컴포넌트로, 사용자의 상호 작용에 응답하고 동적 인 사용자 환경을 만드는 일을 담당한다. 모바일 앱의 특성상 하나의 UI가 떠서 사용자로 부터 입력을 받고, 출력을 담당한다. 즉 하나의 화면 인터페이스에 해당한다고 보면된다. 폰 디스플레이 화면, 카메라 촬영 화면, 이메일 쓰기 화면, 지도 보기 화면 등과 같이 사용자들이 원하는 기능을 위해 상호작용을 할 수 있는 화면을 제공한다는 것이다. 각 액티비티는 하나의 윈도우에 UI를 그리며, 그 윈도우가 보통은 화면을 꽉 채우지만, 화면보다 작을수도 있고, 다른 윈도우의 위에 떠 있을 수도 있다.
- Phoneword 응용 프로그램에는 하나의 화면 (액티비티) 만 있습니다. 화면에 전원을 공급하는 클래스는 MainActivity라고하며 MainActivity.cs 파일에 있다. MainActivity라는 이름은 안드로이드에서 특별한 의미가 없습니다. 비록 MainActivity라는 어플리케이션에서 첫 번째 Activity의 이름을 짓는 것이 관습이지만 안드로이드는 다른 이름이 붙여도 상관 없습니다.

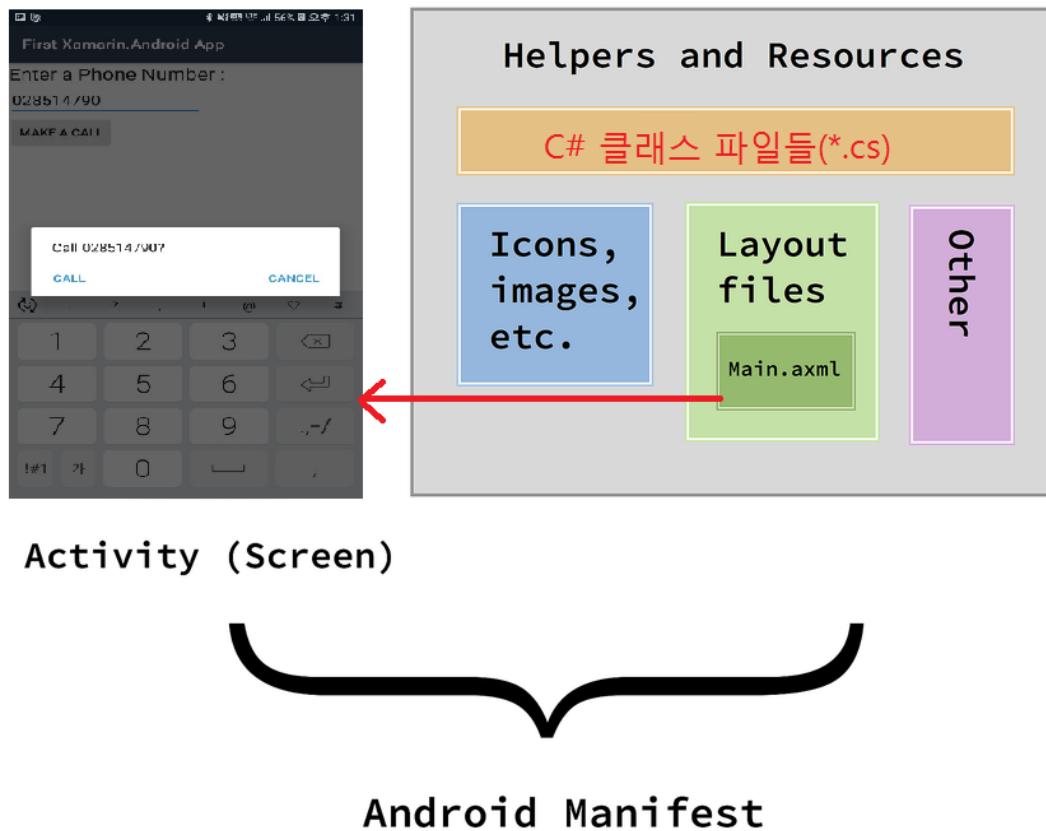
- MainActivity.cs를 열면 MainActivity 클래스가 Activity 클래스의 하위 클래스이고 Activity가 Activity 특성으로 장식되어 있음을 알 수 있습니다.
- 앱은 보통 여려개의 액티비티로 이루어져 있고, 각 액티비티는 서로 느슨한 관계를 갖는다. 일반적인 앱은 하나의 메인(main) 액티비티를 갖고 사용자가 앱을 처음 실행했을 때 보여지는 화면(액티비티)이 되고 각 액티비티는 다른 액티비티를 호출할 수 있는데 다른 액티비티가 실행되면 이전의 액티비티는 정지되지만(stopped), "백스택(BackStack)"이라 불리는 스택에 저장되기 때문에 없어지지는 않는다.
- 안드로이드 시스템은 새로운 액티비티를 시작하면 백스택에 담고 나서 사용자에게 보여주는 데 백스택은 스택 자료구조 이므로 LIFO의 스택 메커니즘을 따르며 사용자가 "뒤로" 버튼을 누를 경우 스택의 최상위(top)에 있는 현재 액티비티를 제거(pop and destroy)하고 이전의 액티비티를 시작 한다.
- Activity Class는 UI를 작동시키는 코드가 있는데 **사용자의 상호작용에 응답하고 동적으로 응답이 처리되는 구조**를 가진다.
- 이전의 Hello Android는 하나의 스크린(Activity)만 가지고 MainActivity 클래스가 그 일을 하며 MainActivity.cs로 존재했다. MainActivity는 Activity를 상속받아 구현하고 클래스 위에는 Activity Attributre로 필요한 속성을 정의한다.
- Activity Attribue는 Android Manifest와 함께 액티비티를 등록하고 안드로이드에게 이 클래스가 Manifest에 의해 관리되는 응용 프로그램의 일부라고 알리는 역할을 한다. Label 속성은 스크린의 맨 위에 표시되는 텍스트를 가리키고 Icon은 텍스트 옆에 표시되는 이미지를 말한다. **MainLauncher 속성은 안드로이드에게 응용 프로그램이 시작할 때 이 액티비티를 표시하도록 지정하므로 액티비티가 여러 개일 때 유용하다.**

```
[Activity(Label = "@string/app_name", MainLauncher = true, Icon = "@drawable/icon")]
public class MainActivity : AppCompatActivity { ...}
```

2.2.3 Activity Life Cycle

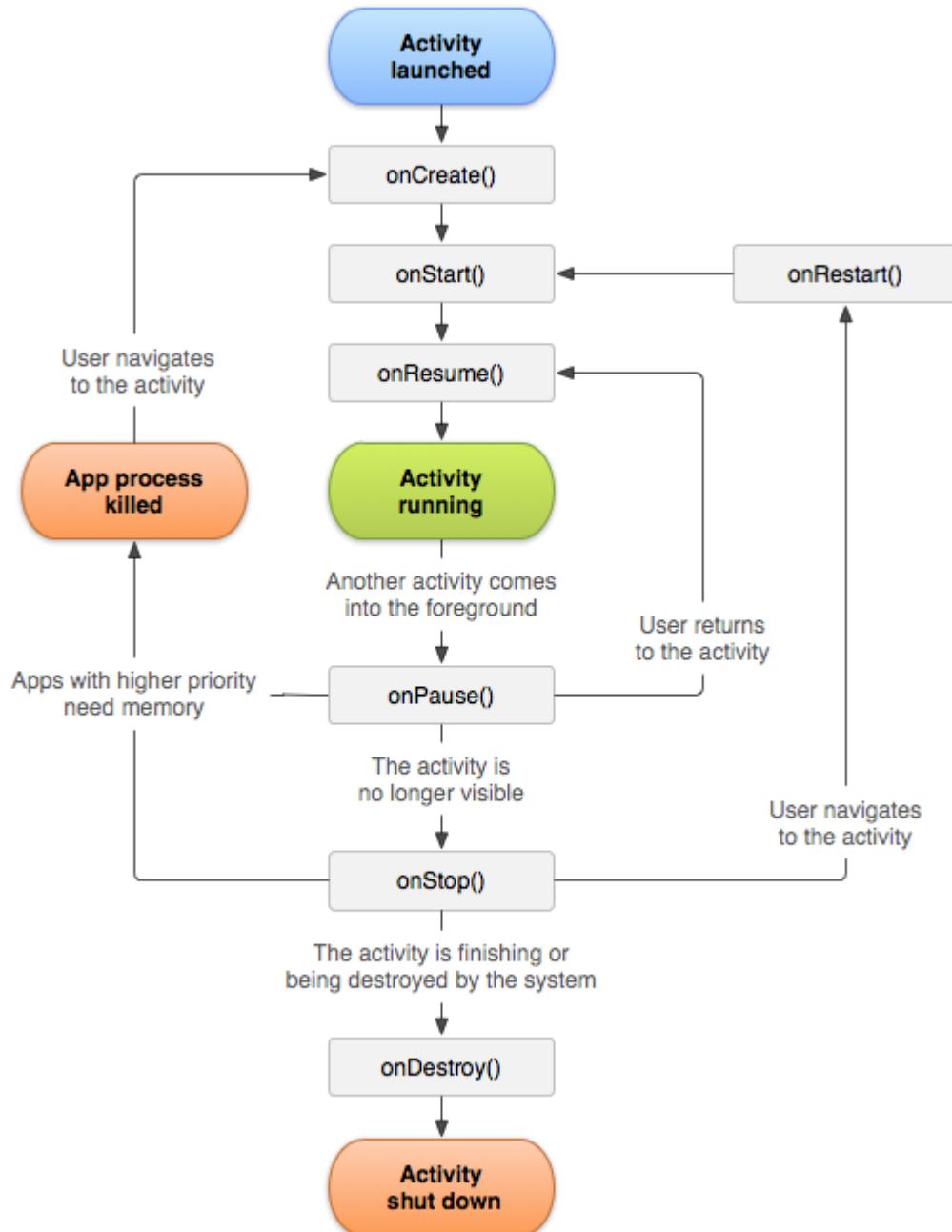
- Xamarin.Android 애플리케이션에는 main과 같은 단일 진입 점이 없다. 즉, 운영 체제가 응용 프로그램을 시작하기 위해 호출하는 응용 프로그램의 코드 줄이 하나도 없다는 말이다. 대신, 안드로이드가 클래스 중 하나를 인스턴스화 할 때 애플리케이션이 시작된다. 이때 안드로이드는 전체 애플리케이션의 프로세스를 메모리로딩한다.
- 에뮬레이터 또는 장치에서 전화걸기 앱 응용 프로그램을 처음 실행하면 운영 체제가 첫 번째 Activity를 만들고 이 액티비티는 하나의 애플리케이션 화면에 해당하는 특수한 Android 클래스이며, 사용자 인터페이스 그리기 및 전원 공급을 담당한다. Android는 애플리케이션의 첫 번째 Activity를 생성 할 때 전체 애플리케이션을 로드한다.
- 전화걸기 앱 예제에서는 애플리케이션을 구성하는 모든 부분이 Android Manifest라는 특수 XML 파일에 등록된다. Android Manifest의 역할은 애플리케이션의 콘텐츠, 속성 및 권한을

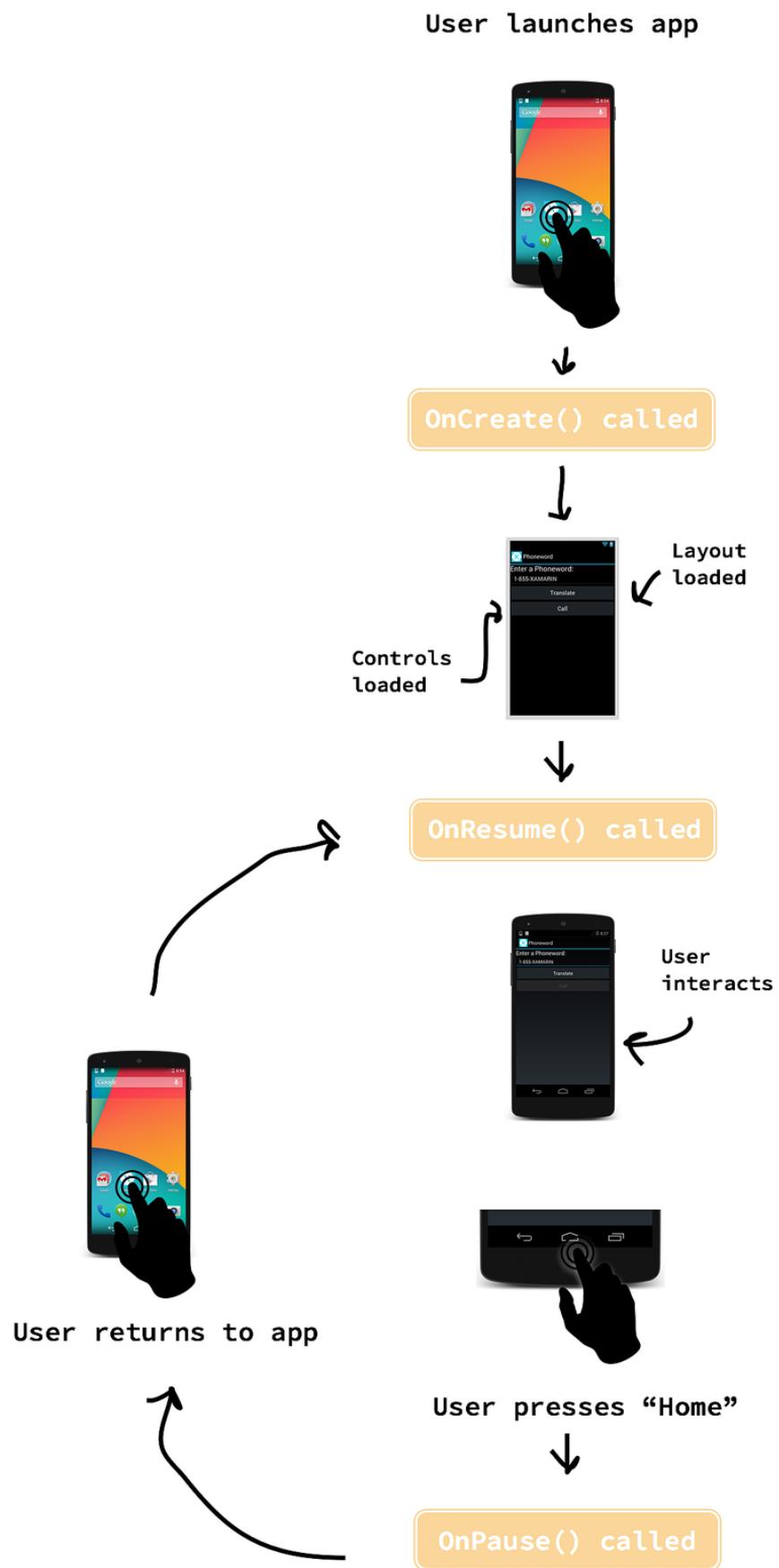
추적하고 이를 Android 운영체제에 공개하는 것인데 아래 그림과 같이 Android Manifest 파일로 묶인 하나의 Activity(화면)와 리소스 및 도우미 파일의 모음으로 생각할 수 있다.



- 새로운 액티비티가 실행되어 현재 액티비티가 정지하게 되면 시스템은 생명주기의 콜백 메소드를 호출함으로써 액티비티의 상태가 변경되었음을 알려주는데 이러한 콜백 메소드들은 시스템이 액티비티를 생성하고, 보여주고, 멈추고, 제거하는 등의 상황에 호출되며, 적절히 오버라이드(override)함으로써 각각의 상황에 맞는 동작을 구현할 수 있다. 예를 들어 액티비티는 정지되었을 때 네트워크나 데이터베이스 관련 객체와 같이 둉치가 큰 객체들을 해제하는 것이 좋고 액티비티가 다시 화면에 보여질 때 필요한 리소스들을 다시 가져와서 중지되었던 작업들을 다시 실행할 수 있다. 이렇게 액티비티가 생성되고 보여지고 멈추고 제거되고 하는 등의 상태변화는 모두 액티비티 생명주기의 일부다.
- Android에서는 액티비티들은 사용자와 상호작용에 의존하는 라이프사이클의 다른 단계를 통하여 작동된다. 액티비티들은 생성(OnCreate), 시작(OnStart), 일시정지(OnPause), 재실행(OnResume), 파괴(OnDestroy)의 라이프 사이클을 가지며 화면 라이프 사이클의 특정 지점에서 시스템이 호출하는 콜백 메소드가 들어 있다.
- Activity가 상태를 변경하면 적절한 라이프 사이클 이벤트 메소드가 호출되어 변경을 알리고 해당 변경 사항에 적응하기 위해 코드를 실행할 수 있다.
- 즉 액티비티 라이프 사이클 메소드를 재정 의하여 Activity가 로드되는 방식, 사용자가 반응하는 방식, 장치 화면에서 사라지는 상황까지 제어 할 수 있다.
- 액티비티 클래스는 스크린 라이프사이클에서 어떤 지점에 시스템이 호출하는 메소드들이 포

함되어 있는데 아래 그림이 전형적인 액티비티의 생명을 보여준다.





OnCreate : 반드시 구현되어야 하는 콜백 메소드로서 사용자에게 화면을 보여주기 전, 즉 액티비티 생성시 호출된다. 액티비티에 필요한 요소들을 초기화 및 멤버변수로 저장하고, setContentView()를 호출하여 UI 컴포넌트등을 로딩하고, Listener를 바인딩하고, 초기 데이터를 로딩하는 등의 초기화 작업을 한다.(OnCreate() 다음으로는 항상 onStart()가 호출된다. Hello 안드로이드에서 UI를 로딩하기 위해 SetContentView를 호출하고 resource 레이아웃 이름(레이아웃 파일 : activity_main) 을 전달해야 하는데 레이아웃은 Resorce.layout.activity_main에 위치한다.

```
// Set our view from the "main" layout resource  
SetContentView (Resource.Layout.activity_main);
```

MainActivity를 시작할 때 activity_main 파일의 내용을 바탕으로 한 뷰가 만들어진다. 레이아웃이름이 Activity 이름과 매칭되는데 **activity_main은 MainActivity의 레이아웃이다**. 이것은 Android 관점에는 필요한 것은 아니나 더 많은 Activity를 붙이기 시작할때 코드 파일에 매칭되는 레이아웃 파일을 좀더 쉽게 찾기 위한 명명 규칙이다.

일단 레이아웃 파일이 세팅되면 추가된 컨트롤들을 찾을수 있는데 컨트롤 찾기 위해서 먼저 findViewById 함수에 컨트롤의 Resource ID를 넘겨주면 된다.

```
EditText phoneNumberText = FindViewById<EditText>(Resource.Id.PhoneNumberText);  
Button translateButton = FindViewById<Button>(Resource.Id.TranslateButton);  
Button callButton = FindViewById<Button>(Resource.Id.CallButton)
```

레이아웃파일에서 컨트롤에 대한 참조를 가지게 되었으므로 사용자 상호작용에 대한 응답을 프로그래밍 할 수 있는 것이다.

- **OnRestart** : 액티비티가 정지되었다가 다시 시작되기 직전에 호출되며 OnRestart() 다음으로는 항상 OnStart()가 호출된다.
- **OnStart** : UI가 화면에 보이기 직전에 호출되며 이 시점에 UI가 로딩 된다고 해도 사용자의 입력을 받을 수는 없다. 이어서 액티비티가 전면에 나서며 포커스를 받을때 OnResume()이 호출되며, 그렇지 않고 바로 가려지면 OnStop()이 호출된다. 즉 다음으로 OnResume() 또는 OnStop()이 호출된다.
- **OnResume** : Activity가 장치 화면으로 돌아올 때마다 발생해야하는 모든 작업을 수행 UI 로딩이 끝난후, 사용자 Input (Interaction)이 시작되기 전에 호출된다. 이 함수들이 다 호출되고 나면 애플리케이션은 실행 가능 상태인 "**Activity Running**" 상태가 되고 UI도 모두 로딩되어 사용자로부터 입력을 받을 준비도 끝난다. 액티비티가 사용자와 상호작용을 할 수 있는 상태가 되기 직전에 호출된다. 이때 액티비티는 백스택(BackStack)의 최상단에 있고, 사용자는 액티비티에 뭔가를 할 수 있는 상태가 된다. OnResume() 다음으로는 항상 OnPause()가 호출됩니다.

- **OnPause** : 정지화면에서 액티비티가 사라질 때 실행된다. 즉 Activity가 장치 화면을 떠날 때마다 발생해야 하는 모든 작업을 수행한다. 사용자와 액티비티의 거리가 한단계 멀어지는 경우에 호출되는 콜백 메소드, 액티비티가 화면에 보여지고 있는 상태와 완전히 정지되어 화면에서 사라진 상태의 중간 단계라고 볼 수 있다. 이 상태 이후에 액티비티가 종료될 수도 있기 때문에 여기서는 액티비티가 다시 시작되었을 때 유지되어야 하는 값들이 있을 경우 저장하는 일을 한다. 정확한 상태 정의는 “**보이기는 하지만 사용자와 Interaction을 할 수 없는 상태**” 정도로 정의할 수 있다. 이런 상태가 어떤 상태인가 하면 다이얼로그등과 같은 다른 액티비티가 앞에 떠서 사용자 Interaction을 수행하는 상태이다 그러나 기존의 Activity는 그대로 뒤에 떠 있지만 뒤에 떠 있는 activity 는 사용자 Interaction을 받지 못하는 상태이다. 이 때 사용중인 쓰레드 정리, 데이터 저장등을 수행하거나 또는 포커스를 잃은 화면이기 때문에 애니메이션등을 정지해야 한다. 다시 해당 Activity로 돌아가게 되면 OnResume으로 다시 돌아가서 화면을 다시 호출하게 된다. 화면이 보이지 않는 상태에서 메모리가 부족하게 되면 안드로이드 시스템에 의해서 이 단계에서 자동으로 정지(Killed) 될 수 있다. Killed 된 상태에서 다시 그 화면으로 돌아가게 되면 다시 onCreate로 돌아가서 앱을 처음부터 다시 시작하게 된다.
- **OnStop** : 액티비티가 더이상 사용자에게 보여지지 않을때 호출된다. 보통 액티비티가 종료되는 상황이거나, 다른 액티비티가 resume 상태가 되면서 이전 액티비티를 완전히 가려버리는 상황이다. 이어서 액티비티가 다시 사용자에게 보여지는 상태가 되는 경우 OnRestart()가 호출되고 액티비티가 종료되고, 종료되어 백스택에서도 빠지는 경우 OnDestroy()가 호출된다.
- **.onDestroy** : 액티비티가 메모리에서 소멸될때 호출된다. 생명주기에서 마지막으로 호출되는 콜백 메소드로 finish()가 호출되었거나 시스템이 다른 작업을 위한 메모리를 확보하기 위해 임의로 종료시키는 경우에 호출된다.

2.2.4 Hello Xamarin Android의 기타 요소들

- 버튼의 속성변경 : callButton.Text = "호출하기";
- Enable and Disable Buttons – 버튼을 Enabled 또는 Disabled 상태로 바꿀수 있다. disabled 상태인 버튼은 사용자에게 반응하지 않는다. 아래 코드는 callButton을 비활성화한다.

```
callButton.Enabled = false;
```

- Display an Alert Dialog - 사용자가 Call 버튼을 누르면 경고 창을 보여주는데 경고 창을 만들기 위해 AlertDialog.Builder 를 사용한다. 다이얼로그창은 Call을 위치할 Neutral 버튼과 Call 을 취소할 Negative 버튼으로 구성된다.

```
callButton.Click += (object sender, EventArgs e) =>
{
    //Make a Call 버튼 클릭시 전화를 건다.
    var callDialog = new AlertDialog.Builder(this);
    callDialog.SetMessage("Call " + phoneNumberText + "?");
```

```

    // "Call"을 클릭하는 경우
    // 전화걸기 위한 인텐트 생성
    callIDialog.SetNeutralButton("Call", delegate
    {
        // 인텐트는 액티비티의 전환이 일어날 때 호출하거나 메시지를
        // 전달하는 매개체
        // 암시적 인텐트 : 전환될 곳을 직접 지정하지 않고 액션을
        // 적어서 사용한다.
        // 명시적 인텐트 : 전환될 액티비티를 직접 적어서 표현하는
        // 방법을 사용한다.
        var callIntent = new Intent(Intent.ActionCall);
        callIntent.SetData(Android.Net.Uri.Parse("tel:" +
        phoneNumberText));
        StartActivity(callIntent);
    });

    // "Cancel"을 클릭하는 경우
    callIDialog.SetNegativeButton("Cancel", delegate { });
};

```

- Launch Phone App with Intent - CallButton을 클릭했을때 전화번호를 바탕으로 시스템 폰 앱을 실행하기 위해 인텐트를 만든다.

```

var callIntent = new Intent(Intent.ActionCall);
callIntent.SetData(Android.Net.Uri.Parse("tel:" + translatedNumber));
StartActivity(callIntent);

```

- 해상도에 따른 다른 아이콘의 설정

안드로이드 장치들은 다른 화면 크기와 해상도를 가지고 있다. 모든 이미지가 모든 화면에서 보기 좋을 수는 없다. 안드로이드는 낮은 밀집도의 ldpi 버전, 중간의 mdpi, 높음의 hdpi, 매우 높은 밀집도 스크린의 xhdpi, xxhdpi 등을 포함해서 여러가지 다른 밀집도의 이미지들을 drawable 폴더의 버전들로 제공한다.

2.2.5 Xamarin Android Intent(인텐트)

- 안드로이드를 이루는 4가지 기본 요소는 Activity, Service, Broadcast Receiver, Content Provider가 있는데 인텐트(Intent)란 이러한 어플리케이션 구성요소 사이의 작업 수행을 위한 정보를 전달하는 기능을 하는데 **액티비티의 화면전환이 일어날 때 호출하거나 메시지를 전달하는 매개체로서의 역할을 한다.**

- 안드로이드 OS에서 수행 되어야하는 작업을 위한 추상적인 개념인데 전화를 걸거나, 웹 페이지를 표시하거나, 주소를 매핑하는 등의 의도로 외부 프로그램을 시작하는 데 사용된다.
- 인텐트는 명시적 인텐트와 암시적 인텐트 두가지 방법이 사용 되는데 명시적 인텐트는 전환될 액티비티를 직접 적어서 표현하는 방법이다. 아래는 CallHistoryActivity화면으로 이동시키는 예문인데 다음화면으로 넘기면서 "phone_numbers"라는 이름으로 ArrayList에 담긴 데이터를 넘긴다.

```
var intent = new Intent(this, typeof(CallHistoryActivity));
intent.PutStringArrayListExtra("phone_numbers", phoneNumbers);
StartActivity(intent);
```

- 암시적 인텐트는 전환될 곳을 직접 지정하지 않고 액션을 적어서 사용하며 전환될 곳에도 액션을 적어 인텐트를 받는다.

```
var callIntent = new Intent(Intent.ActionCall);
callIntent.SetData(Android.Net.Uri.Parse("tel:" + translatedNumber));
StartActivity(callIntent);
```

2.2.6 Simple Intent Example(웹페이지 오픈하기)

- “**OpenWebpage**”라는 이름의 Xamarin.Android 프로젝트 생성
- Resource\layout\activity_main

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <Button
        android:id="@+id/myButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    />
</LinearLayout>
```

■ Resource\values\Strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Go! OracleJava Community</string>
    <string name="app_name">OpenWebpage</string>
</resources>
```

■ MainActivity.cs 파일에 Button의 클릭 이벤트 코드 작성

```
using System;

using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;

namespace OpenWebpage
{
    [Activity (Label = "OpenWebpage", MainLauncher = true)]
    public class MainActivity : Activity
    {

        protected override void OnCreate (Bundle bundle)
        {
            base.OnCreate (bundle);

            // Set our view from the "main" layout resource
            SetContentView (Resource.Layout.Main);

            // Get our button from the layout resource,
            // and attach an event to it
            Button button = FindViewById<Button> (Resource.Id.myButton);

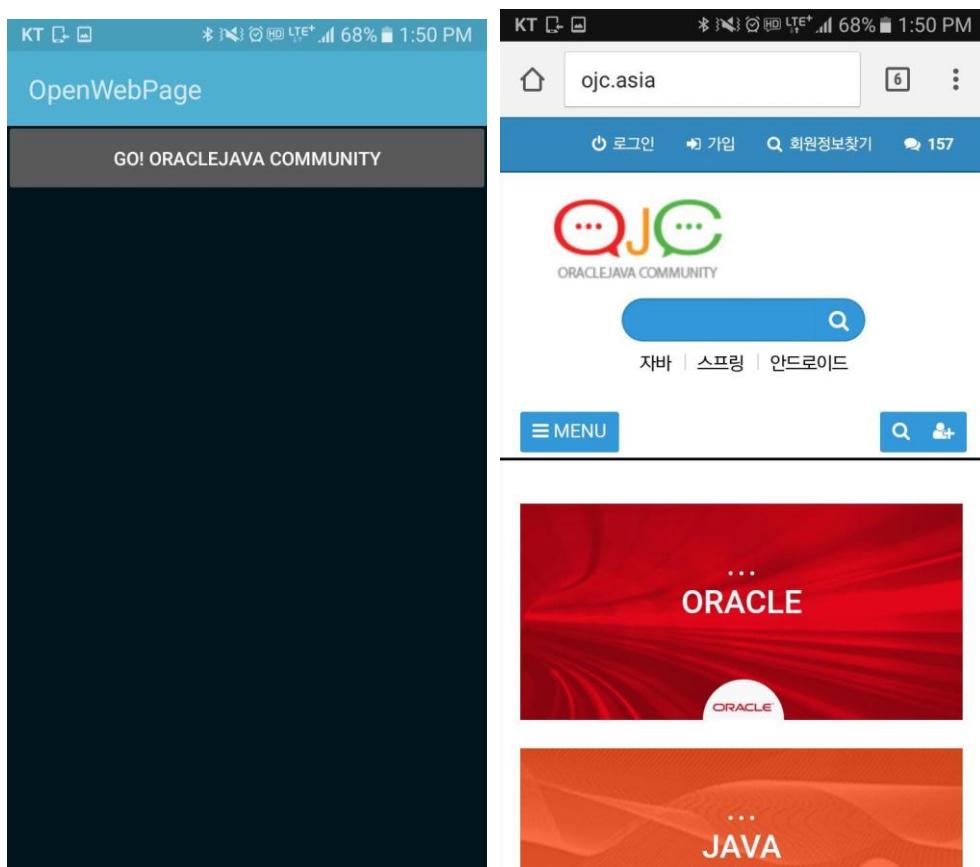
            button.Click += (sender, e) => {

                var uri = Android.Net.Uri.Parse ("http://ojc.asia");
            };
        }
    }
}
```

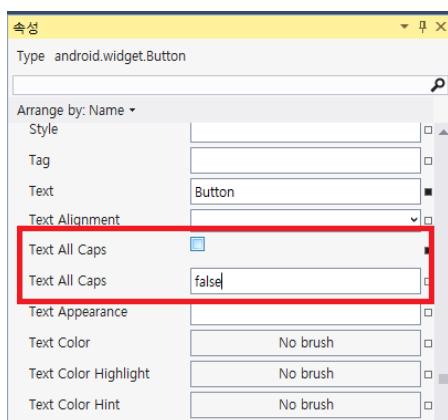
```

        var intent = new Intent (Intent.ActionView, uri);
        StartActivity (intent);
    };
}
}
}
}

```



버튼의 글씨를 원래 입력한대로 소문자로 출력하려면 Button의 Text All Caps 속성의 체크를 풀거나 false로 입력하면 된다.



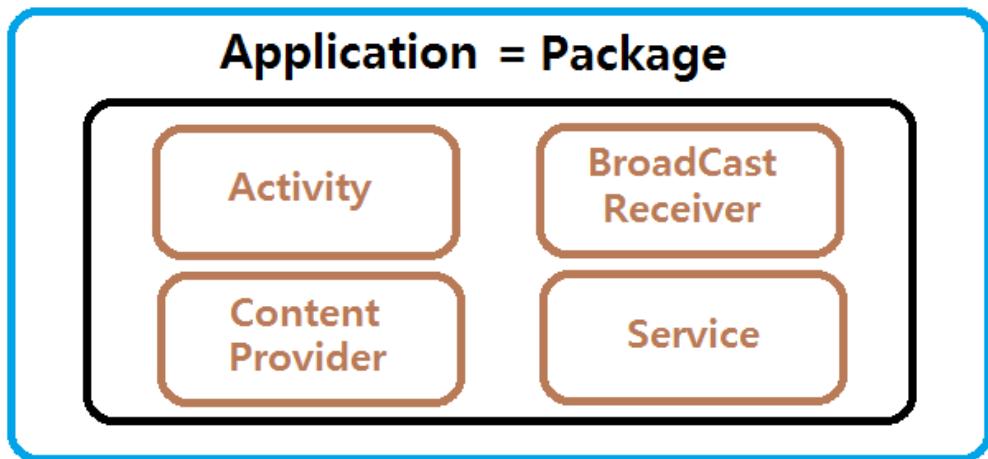
2.2.7 Android Service 개요

- 안드로이드 Application을 구성하는 4가지 컴포넌트 중에 하나인 Service는 Activity처럼 사용자와 상호작용 하는 컴포넌트가 아니고, 사용자 인터페이스(UI)도 제공하지 않고 백그라운드에서 주로 시간이 오래 걸리는 작업들을 처리하는 앱 컴포넌트이다.
- 모바일 앱은 데스크톱 앱과 다르다. 데스크탑은 화면 공간, 메모리, 저장 공간 및 연결된 전원 공급 장치와 같은 많은 자원을 가지고 있지만 모바일 장치는 그렇지 않다. 이러한 제약으로 인해 모바일 앱이 다르게 작동하는데, 예를 들어 휴대 기기의 작은 화면은 일반적으로 한번에 하나의 앱 (즉, Activity) 만 표시된다는 의미이며 기타 Activity는 백그라운드로 이동하고 작업을 수행 할 수없는 일시 중지 상태로 푸시된다. 그러나 Android 애플리케이션이 백그라운드에 있다고 해서 앱이 계속 작동하는 것이 불가능 하지않다.
- Activity는 사용자가 응용 프로그램과 상호 작용할 수있는 UI를 제공하기 때문에 Android 응용 프로그램의 시작이다. 그러나 동시 또는 배경 작업을 수행 할 때 Activity가 항상 최고의 선택이되는 것은 아니다.
- 안드로이드에서 백그라운드 작업의 기본 메커니즘은 서비스로 Android 서비스는 사용자 인터페이스없이 일부 작업을 수행하도록 설계된 구성 요소로 서비스는 안드로이드 어플리케이션 간의 프로세스 간 통신 (IPC)에도 사용될 수 있다. 예를 들어 하나의 Android 앱이 다른 앱의 뮤직 플레이어 서비스를 사용하거나 앱이 데이터 (예 : 사람의 연락처 정보)를 서비스를 통해 다른 앱에 노출 할 수 있다.
- 서비스와 백그라운드 작업의 처리능력은 유연한 사용자 인터페이스를 제공하는 데 중요하다. 모든 Android 애플리케이션에는 Activity가 실행되는 기본 스레드 (UI 스레드라고도 함)가 있는데 기기가 응답 하도록 하려면 Android는 초당 60 프레임의 속도로 사용자 인터페이스를 업데이트 할 수 있어야 한다. Android 앱이 주 스레드에서 많은 작업을 수행하면 Android가 프레임을 삭제하고 이로 인해 UI가 번쩍이는 것처럼 보입니다 (때로는 janky라고도 함). 즉, UI 스레드에서 수행 된 작업은 두 프레임 사이의 시간 간격 (약 16 밀리 초 (60 프레임마다 1 초)) 동안 완료되어야 한다.
- 이러한 우려를 해소하기 위해 개발자는 Activity의 스레드를 사용하여 UI를 차단하는 작업을 수행 할 수 있는데 이로 인해 문제가 발생할 수 있다. Android가 여러 Activity 인스턴스를 종료시키고 다시 만들 가능성이 매우 높다. 그러나 Android는 스레드를 자동으로 삭제하지 않으므로 메모리 누수가 발생할 수도 있는데 가장 대표적인 예는 모바일 기기가 회전 한 경우로 Android는 Activity의 인스턴스를 종료시킨 다음 새 인스턴스를 다시 만들려고 하는데 Activity의 첫 번째 인스턴스에 의해 생성 된 스레드는 계속 실행되고 스레드가 Activity의 첫 번째 인스턴스에 대한 참조를 가지고 있다면 Android가 객체를 가비지 컬렉션하지 못하게 된다. 그러나 Activity의 두 번째 인스턴스는 여전히 생성되며 (이는 다시 새 스레드를 생성 할

수 있음) 빠른 속도로 연속해서 여러 번 기기를 돌리면 모든 RAM이 소모되어 안드로이드가 메모리를 회수하기 위해 전체 애플리케이션을 강제 종료 할 수 있다.

- 일반적으로 수행해야 할 작업이 Activity보다 오래 지속되어야 한다면, 그 작업을 수행하기 위한 서비스가 만들어 져야 하는데, 그러나 작업이 Activity의 컨텍스트에서만 적용 가능하면 작업을 수행 할 스레드를 만드는 것이 더 적절할 수 있다. 예를 들어 사진 갤러리 앱에 방금 추가 된 사진의 미리보기를 만들려면 서비스로 구현을 하는 것이 좋고, 액티비티가 Foreground 에 있을 때만 들려야하는 음악을 재생하려면 스레드가 더 적합 할 수 있다.
- 다른 컴포넌트가 서비스를 실행할 수 있고 사용자가 내 앱에서 벗어나 다른 앱으로 이동하더라도 서비스는 백그라운드에서 하던 일을 계속하도록 지원하며 컴포넌트는 서비스와 상호작용을 하기 위해 바인드할 수 있고, 심지어 프로세스간 통신(IPC)도 할 수 있다.
- 시간이 걸리는 계산작업, 파일 다운로드, 네트워크를 통한 데이터 전송 및 수신, 음악 재생, 주기적으로 특정 웹사이트에서 데이터를 읽어 온다든지, 파일 I/O, 컨텐트 프로바이더와의 상호작용 등을 백그라운드에서 처리할 수 있다. 즉 Activity 화면에서 동작뿐만 아니라 Activity 가 종료되어 있는 상태에서도 동작하기 위해서 만들어진 컴포넌트 이다.
- 백그라운드 작업은 크게 두 가지로 분류 할 수 있다.
 - ✓ 장기 실행 태스크 - 명시적으로 중지 될 때까지 진행중인 작업으로 장기 실행 작업의 예로 음악을 스트리밍하거나 센서에서 수집 한 데이터를 모니터링 해야 하는 앱등으로 응용 프로그램에 표시되는 사용자 인터페이스가 없는 경우에도 이러한 작업이 실행되어야 한다.
 - ✓ 주기적 작업 - 주기적 작업이란 비교적 짧은 기간 (수 초)으로 일정에 따라 실행됩니다 (즉, 하루에 한 번 또는 일주일에 한 번 또는 다음 60 초 내에 한 번만 실행 됨). 예를 들면 인터넷에서 파일을 다운로드하거나 이미지의 축소판을 생성하는 것등이 있다.

Process



■ 안드로이드 서비스는 실행되는 방식에 따라 4가지 유형으로 구분할 수 있다

1. **바운드 서비스(Bound Service)** - 바운드 서비스는 다른 구성 요소 (일반적으로 Activity)가 바인딩 된 서비스입니다. 바운드 서비스는 바인딩 된 구성 요소와 서비스가 서로 상호 작용할 수 있도록하는 인터페이스를 제공하며 서비스에 바인딩 된 클라이언트가 더 이상 없으면 Android가 서비스를 종료합니다.
2. **인텐트 서비스(IntentService)** - IntentService는 서비스 생성 및 사용을 단순화하는 Service 클래스의 특수 하위 클래스입니다. IntentService는 개별적인 호출을 처리하기 위한 것으로 여러 개의 호출을 동시에 처리 할 수 있는 서비스와 달리 IntentService는 작업 대기열 프로세서와 비슷하다. 작업은 대기열에 있으며 IntentService는 단일 작업자 스레드에서 한 번에 하나씩 각 작업을 처리합니다. 일반적으로 IntentService는 예 바인딩되지 않는다.
3. **Started Service(시작 서비스)** - Started Service는 다른 Android 구성 요소 (예 : Activity)에 의해 시작된 서비스로, 서비스가 중지 되도록 명시 적으로 알릴 때까지 백그라운드에서 계속 실행된다. 바운드 서비스와 달리 시작된 서비스에는 직접 바인딩 된 클라이언트가 없고 이러한 이유로 시작된 서비스가 필요에 따라 정상적으로 다시 시작되도록 설계하는 것이 중요하다.
4. **Hybrid Service(하이브리드 서비스)** - 하이브리드 서비스는 Started Service와 Bound Service의 특성을 가진 서비스입니다. 하이브리드 서비스는 구성 요소가 구성 요소에 바인딩되거나 특정 이벤트에 의해 시작될 때 시작될 수 있고 클라이언트 구성 요소는 하이브리드 서비스에 바인딩되거나 바인딩되지 않을 수 있다. 하이브리드 서비스는 명시 적으로 중지 할 때까지 또는 더 이상 클라이언트가 바인딩되지 않을 때까지 계속 실행된다.

- 일반적으로 사용할 서비스 유형은 응용 프로그램 요구 사항에 따라 다른데 일반적으로 Android 애플리케이션이 수행해야 하는 대부분의 작업에는 IntentService 또는 바운드 서비스로 충분하기 때문에 두 가지 유형의 서비스 중 하나를 선호한다. IntentService는 파일 다운로드와 같은 "원 샷"작업에 적합하지만 액티비티와의 상호 작용이 빈번 할 때 바운드 서비스가 적합하다.

2.2.8 Xamarin.Android Service 생성 및 시작하기

- Xamarin.Android 서비스 구현을 위한 규칙

Android.App.Service를 확장해야 하며,
Android.App.ServiceAttribute로 꾸며져 있어야 한다.

- 안드로이드 서비스의 또 다른 요구 사항은 안드로이드 서비스가 AndroidManifest.xml에 등록되고 고유 한 이름이 주어져야 한다는 것인데 Xamarin.Android는 필요한 XML 속성을 사용하여 빌드시 매니페스트에 서비스를 자동으로 등록한다.
- Xamarin.Android에서 서비스 만드는 방법

```
[Service]
public class DemoService : Service
{
    // Magical code that makes the service do wonderful things.
}
```

- 컴파일 할 때 Xamarin.Android는 AndroidManifest.xml에 다음 XML 요소를 삽입하여 서비스를 등록한다. (Xamarin.Android가 서비스의 임의의 이름을 생성했음을 알 수 있다).

```
<service android : name = "md5a0cbbf8da641ae5a4c781aaf35e00a86.DemoService"/>
```

- 다른 안드로이드 응용 프로그램과 함께 서비스를 내보냄으로써 서비스를 공유 할 수 있는데 이 작업은 ServiceAttribute에서 Exported 속성을 설정하여 수행 할 수 있다. 서비스를 내보낼 때 ServiceAttribute.Name 속성도 서비스에 의미있는 이름을 제공하도록 설정해야 한다. 아래 예문을 참조하자.

```
[Service(Exported=true, Name="com.xamarin.example.DemoService")]
public class DemoService : Service
{
    // Magical code that makes the service do wonderful things.
```

```
}
```

[AndroidManifest.xml]

```
<service android : exported = "true" android : name = "com.xamarin.example.DemoService"/>
```

- 기본적으로 서비스는 Android 애플리케이션과 동일한 프로세스에서 시작되며 ServiceAttribute.IsolatedProcess 속성을 true로 설정하여 자체 프로세스에서 서비스를 시작할 수 있다.

[Service(IsolatedProcess=true)]

```
public class DemoService : Service
{
    // Magical code that makes the service do wonderful things, in it's own process!
}
```

- 안드로이드 서비스의 시작 : 안드로이드에서 서비스를 시작하는 가장 기본적인 방법은 시작 해야 하는 서비스를 식별하는 데 도움이되는 메타 데이터가 포함 된 Intent를 전달하는 것인데 서비스를 시작하는 데 사용할 수 있는 두 가지 스타일의 인텐트(Intent)가 있다.
 - ✓ **명시적 인텐트(Explicit Intent)** : 명시적 인텐트는 주어진 작업을 완료하기 위해 사용 되어야 하는 서비스를 정확히 기술한다. Android는 Intent를 명시적으로 식별 된 서비스로 라우팅하는데 아래는 명시적 인 텐트를 사용하여 DownloadService라는 서비스를 시작하는 한 예이다.

```
Intent downloadIntent = new Intent(this, typeof(DownloadService));

downloadIntent.data = Uri.Parse(fileToDownload);
```

- ✓ **묵시적 인텐트(Implicit Intent)** : 작업을 느슨하게 식별하지만 해당 작업을 완료하는 정확한 서비스는 알 수 없다. 안드로이드는 묵시적 인텐트의 내용을 검사하고, 인텐트와 일치하는 기존 서비스가 있는지 여부를 결정한다. 인텐트 필터는 묵시적인 의도와 등록 된 서비스를 일치시키는 데 사용되며 AndroidManifest.xml에 추가되는 XML 요소로, 서비스를 암시 적 의도와 일치시키는 데 필요한 메타 데이터가 포함되어 있다.

```
Intent sendIntent = new Intent("common.xamarin.DemoService");

sendIntent.Data = Uri.Parse(fileToDownload);
```

- 가능하다면 애플리케이션은 명시적인 인 텐트를 사용하여 서비스를 시작해야 한다. 묵시적 인텐트는 특정 서비스 시작을 요구하지 않고 요청을 처리하기 위해 장치에 설치된 일부 서비

스에 대한 요청으로 이러한 모호한 요청은 다른 서비스 또는 앱을 시작할 수 있다.

- Creating an Intent Filter for Implicit Intents

서비스를 암시적 인텐트와 연결하려면 Android 앱이 서비스의 기능을 식별 할 수 있는 메타 데이터를 제공해야 하는데 이 메타 데이터는 인텐트 필터에 의해 제공된다. 인텐트 필터는 서비스를 시작하기 위해 인텐트에 있어야 하는 일부 정보 (예 : 데이터 유형)를 포함한다. Xamarin.Android에서 인텐트 필터는 IntentFilterAttribute로 서비스를 만들면 AndroidManifest.xml에 등록된다. 예를 들어 다음 코드는 com.xamarin.DemoService와 연결된 동작이 있는 인텐트 필터를 추가한다.

[Service]

```
[IntentFilter(new String[]{"com.xamarin.DemoService"})]  
public class DemoService : Service  
{ }
```

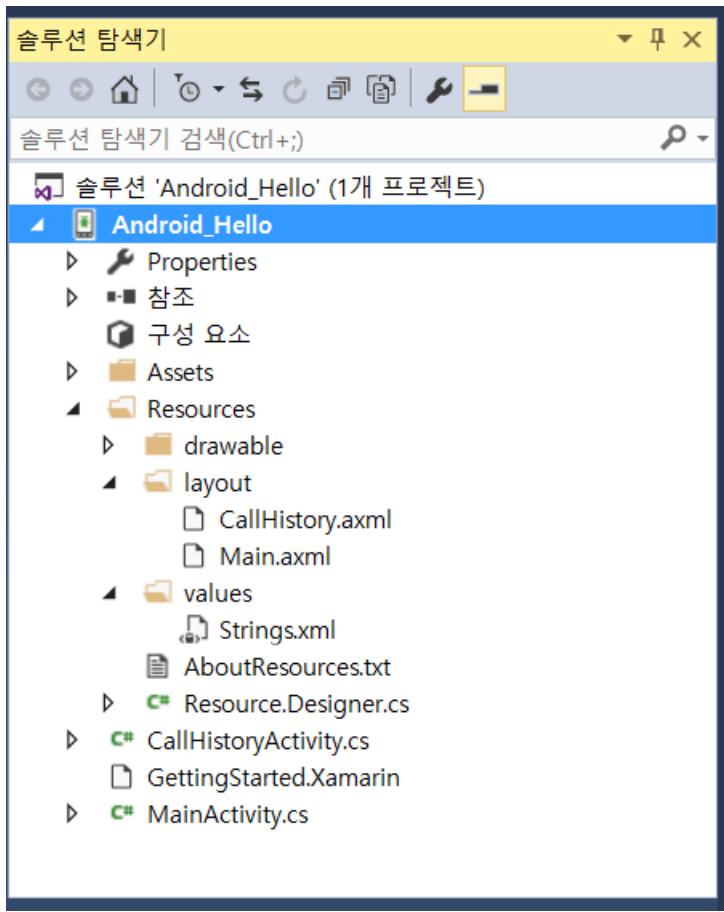
[AndroidManifest.xml]

```
<service android:name="demoservice.DemoService">  
    <intent-filter>  
        <action android:name="com.xamarin.DemoService" />  
    </intent-filter>  
</service>
```

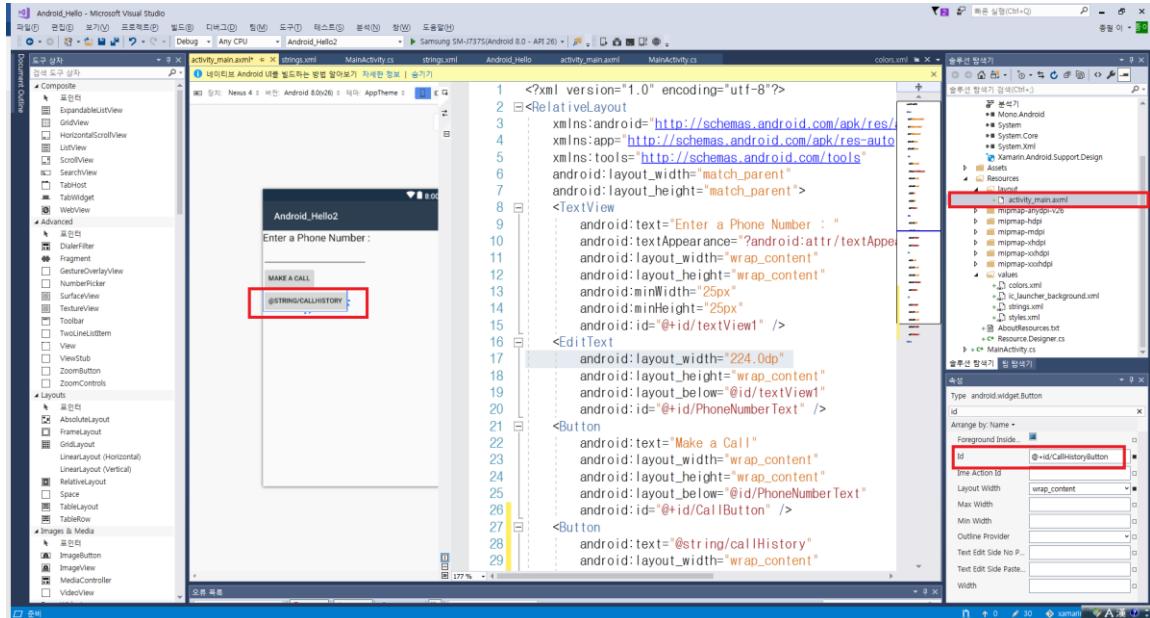
2.3 Hello Android MultiScreen Example

- 이전 Android_Hello 프로젝트에 전화건 내역을 저장하는 화면과 로직을 추가해 보자.
- 비주얼 스튜디오에서 **이전에 작성한 Android_Hello 응용프로그램을 오픈하자.**

[완성된 프로젝트 구조]



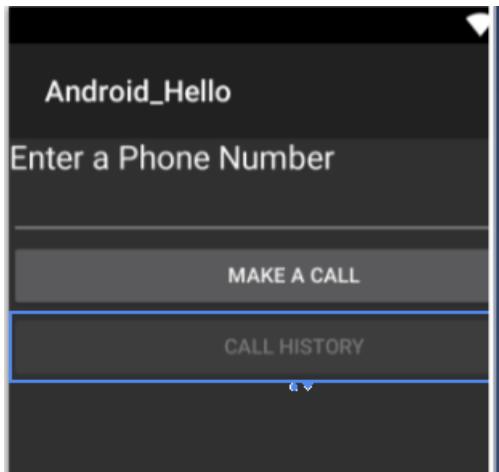
- Resource/Layout아래의 **activity_main**을 오픈 후 도구상자(ToolBox)에서 Button을 드래그하여 Call Button 아래에 위치시킨 후 속성창에서 **Id속성을 "@+id/CallHistoryButton"**, **Text 속성을 "@string/callHistory"**로 설정한다.



- Resources/value 아래의 Strings.xml파일을 오픈 후 Resources 안쪽에 아래코드를 삽입하자.

```
<string name="callHistory">CallHistory</string>
```

- Activity_main 화면에서 CallHistoryButton 선택후 enabled 속성을 false로(체크를 푼다.) 설정하면 디자인 화면에서 버튼이 비활성화 된다.



- 전화건 내역을 저장하는 두번째 화면을 위한 Activity를 생성하는데 프로젝트에 추가 -> 새항목에서 **Visual C# -> Activity(동작)를 선택, 이름을 CallHistoryActivity.cs로** 하고 아래 내용으로 코드를 채우자. ([ListActivity](#)를 상속받았음에 주의하자!, 별도의 UI인 *.axml 파일이 필요없다.)

```

using Android.App;
using Android.Content;
using Android.OS;
using Android.Widget;

namespace Android_Hello
{
    [Activity(Label = "@string/callHistory")]
    public class CallHistoryActivity : ListActivity
    {
        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            var phoneNumbers = Intent.Extras.GetStringArrayList("phone_numbers") ?? new string[0];

            // ListView는 스크롤하는 행 목록을 표시하는 간단한 방법을 제공하는 UI 구성 요소이다.
            // ListView에 데이터 출력을 위해 아래 코드처럼 어댑터를 제공해야 한다.
            //ListAdapter 속성은 Activity의 ListView에 연동될 ListAdapter를 지정
            // 현재 액티비티의 ListView1이라는 리스브류에 phoneNumbers라는 ArrayList를 연동시킴
            // Android.Resource.Layout.SimpleListItem1은 내장된 간단한 리스트아이템이다.
            thisListAdapter = new ArrayAdapter<string>(
                this,           // Context, 일반적으로 Activity
                Android.Resource.Layout.SimpleListItem1, // Layout, 데이터의 표시방법
                phoneNumbers); // SimpleListItem1에 바인딩 될 열거형 데이터
        }
    }
}

```

- Android의 ListView를 사용하면 열거 가능한 컬렉션을 반복하고 각 데이터를 목록 항목에 표시 할 수 있다. 목록보기는 어댑터와 함께 작동하여 데이터를 반복하고 레이아웃에 데이터를 표시하는 구조다.
- 단순한 데이터를 화면에 표시하고자하는 경우 Android.Resource.Layout.SimpleListItem1과 같은 built-in list item layouts을 사용하면 된다.
- **MainActivity.cs** 파일을 오픈해서 전화를 걸 때 전화번호를 저장하기 위한 List<string>를 위한 "System.Collections.Generic"을 import하고 phoneNumbers라는 컬렉션 변수를 정의하자.

```
using System;
using Android.App;
using Android.Content;
using Android.Widget;
using Android.OS;
using Android.Text;
using System.Collections.Generic;

namespace Android_Hello
{
    [Activity(Label = "Xamarin Android",
        MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        static readonly List<string> phoneNumbers = new List<string>();
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.Main);
        }
    }
}
```

- MainActivity의 OnCreate 메소드앞 SetContentView 뒤부분에 CallHistoryButton의 이벤트 핸들러를 작성하자. 혹시 Resource.Id.CallHistoryButton을 못찾으면 솔루션 탐색기에서 F5를 누른 후 다시해보자.

```

namespace Android_Hello
{
    [Activity(Label = "Xamarin Android",
        MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        static readonly List<string> phoneNumbers = new List<string>();
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            // Set our view from the "main" layout resource
            SetContentView (Resource.Layout.Main);

            Button callHistoryButton = FindViewById<Button>(Resource.Id.CallHistoryButton);
            callHistoryButton.Click += (sender, e) =>
            {
                // 인텐트는 액티비티의 전환이 일어날 때 호출하거나 메시지를 전달하는 매개체
                // 암시적 인텐트 : 전환될 곳을 직접 지정하지 않고 액션을 적어서 사용한다.
                // 명시적 인텐트 : 전환될 액티비티를 직접 적어서 표현하는 방법을 사용한다.
                var intent = new Intent(this, typeof(CallHistoryActivity));

                // PutStringArrayListExtra는 Intent에 전화 번호 목록을 첨부한다.
                intent.PutStringArrayListExtra("phone_numbers", phoneNumbers);
                StartActivity(intent);
            };
        }

        EditText phoneNumberText =
            FindViewById<EditText>(Resource.Id.PhoneNumberText);
    }
}

```

- 사용자가 “Make a Call”을 클릭하고 “Call”을 클릭할 때 사용자 전화번호를 저장해 두기 위해 위에서 만든 케이션 변수 phoneNumbers에 저장하고 CallHistoryButton을 활성화하는 기능 추가를 위해 callDialog.SetNeutralButton를 수정하자.

[아래는 MainActivity.cs의 전체코드이다.]

```

using System;
using Android.App;
using Android.Content;
using Android.Widget;
using Android.OS;
using Android.Text;
using System.Collections.Generic;

namespace Android_Hello
{
    [Activity(Label = "Xamarin Android",
        MainLauncher = true, Icon = "@drawable/icon")]

```

```

public class MainActivity : Activity
{
    static readonly List<string> phoneNumbers = new List<string>();
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);

        // Set our view from the "main" layout resource
        SetContentView (Resource.Layout.Main);

        Button callHistoryButton =
FindViewById<Button>(Resource.Id.CallHistoryButton);
        callHistoryButton.Click += (sender, e) =>
        {
            // 인텐트는 액티비티의 전환이 일어날 때 호출하거나
            메시지를 전달하는 매개체
            // 암시적 인텐트 : 전환될 곳을 직접 지정하지 않고
            액션을 적어서 사용한다.
            // 명시적 인텐트 : 전환될 액티비티를 직접 적어서
            표현하는 방법을 사용한다.
            var intent = new Intent(this,
typeof(CallHistoryActivity));

            // PutStringArrayListExtra는 Intent에 전화 번호 목록을
            첨부한다.
            intent.PutStringArrayListExtra("phone_numbers",
phoneNumbers);
            StartActivity(intent);
        };

        EditText phoneNumberText =
FindViewById<EditText>(Resource.Id.PhoneNumberText);

        //전화걸기 버튼
        Button callButton =
FindViewById<Button>(Resource.Id.CallButton);

        callButton.Enabled = false;

        phoneNumberText.TextChanged +=
            (object sender, TextChangedEventArgs e) =>

```

```

    {
        if (!string.IsNullOrEmpty(phoneNumberText.Text))
            callButton.Enabled = true;
        else
            callButton.Enabled = false;
    };

    callButton.Click += (object sender, EventArgs e) =>
    {
        //Make a Call 버튼 클릭시 전화를 건다.
        var callDialog = new AlertDialog.Builder(this);
        callDialog.SetMessage("Call " + phoneNumberText.Text +
"?");

        // "Call"을 클릭하는 경우
        // 전화걸기 위한 인텐트 생성
        callDialog.SetNeutralButton("Call", delegate
        {
            phoneNumbers.Add(phoneNumberText.Text);
            callHistoryButton.Enabled = true;

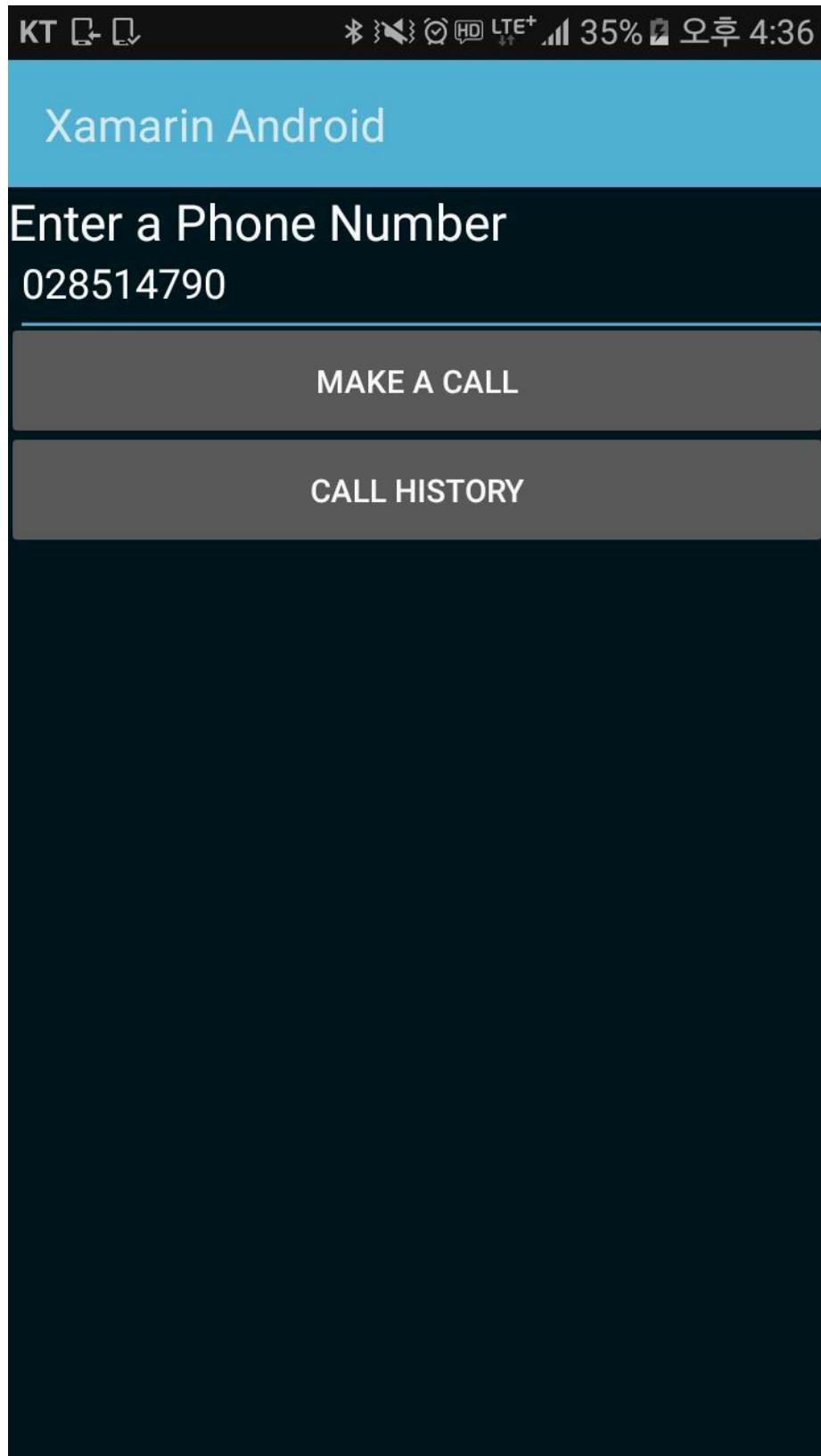
            // 인텐트는 액티비티의 전환이 일어날 때 호출하거나
            메시지를 전달하는 매개체
            // 암시적 인텐트 : 전환될 곳을 직접 지정하지 않고
            액션을 적어서 사용한다.
            // 명시적 인텐트 : 전환될 액티비티를 직접 적어서
            표현하는 방법을 사용한다.
            var callIntent = new Intent(Intent.ActionCall);
            callIntent.SetData(Android.Net.Uri.Parse("tel:" +
phoneNumberText.Text));
            StartActivity(callIntent);
        });

        //Cancel을 클릭하는 경우
        callDialog.SetNegativeButton("Cancel", delegate { });

        callDialog.Show();
    };
}
}

```

■ 실행 화면



KT

* 3G Q HD LTE+ 35% 4:36

Call History

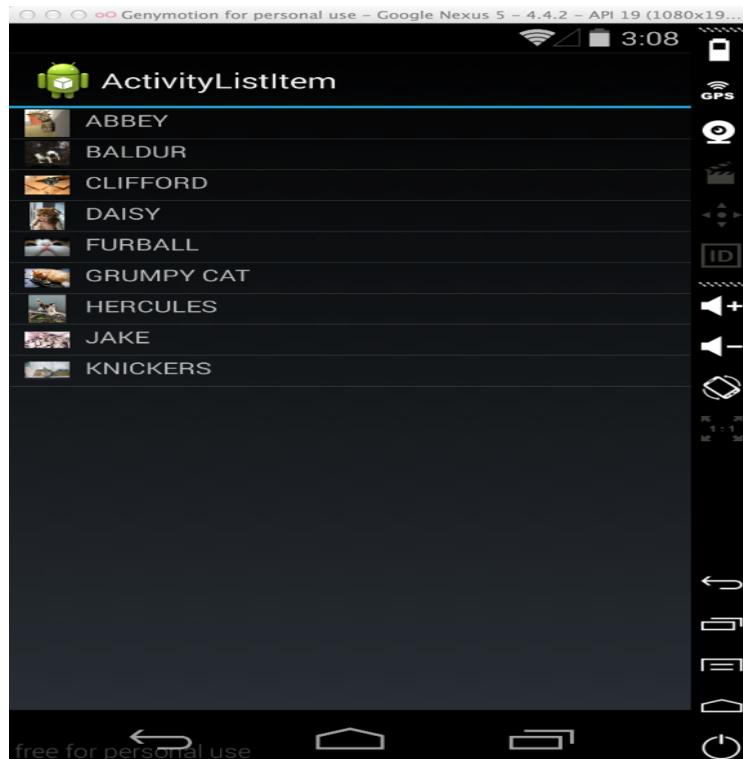
028514790

2.4 Built-In List Item Layouts(내장 리스트아이템 레이아웃)

- 내장된 리스트 아이템들을 위한 Layouts은 다음과 같은 것들이 있다.
- `Android.Resource.Layout.ActivityListItem`

1 `ImageView` (`Android.Resource.Id.Icon`)

1 `TextView`(`Android.Resource.Id.Text1`)



- `Android.Resource.Layout.SimpleListItem1`

1 `TextView`(`Android.Resource.Id.Text1`)

```
using System;
using Android.App;
using Android.OS;
using Android.Widget;

namespace HelloWorld
{
    [Activity (Label = " MainActivity")]
    public class MainActivity : ListActivity
    {
```

```

protected override void OnCreate (Bundle savedInstanceState)
{
    base.OnCreate (savedInstanceState);

    var names = new [] { "Fluffy", "Muffy", "Tuffy" };

    thisListAdapter = new ArrayAdapter (
        this, //Context, typically the Activity
        Android.Resource.Layout.SimpleListItem1, //The
        layout. How the data will be presented
        names //The enumerable data
    );
}
}
}

```

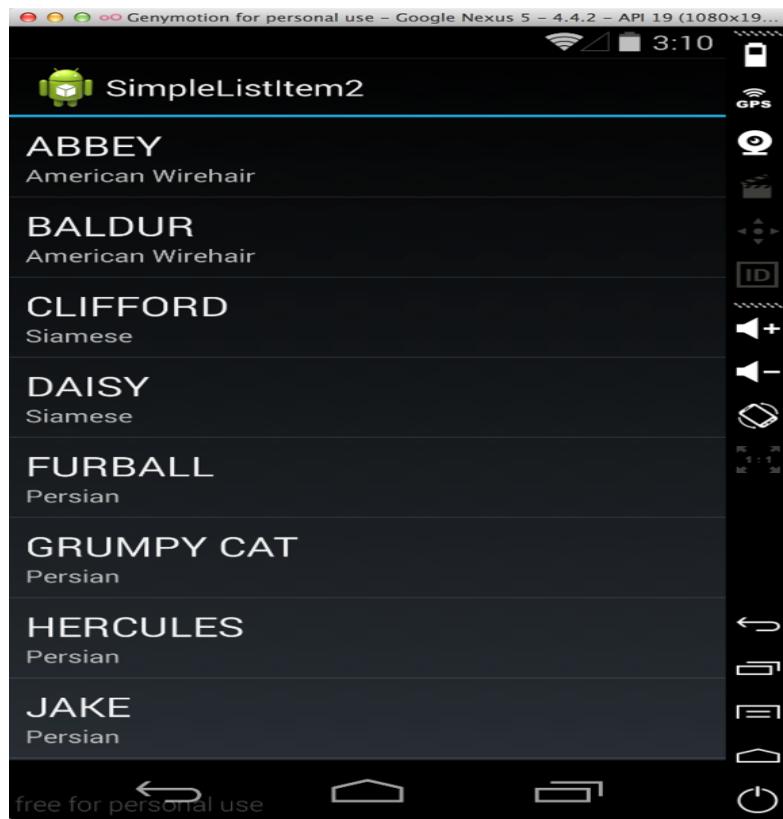


■ `Android.Resource.Layout.SimpleListItem2`

```

1TextView/Title (Android.Resource.Id.Text1)
1TextView/Subtitle(Android.Resource.Id.Text2)

```

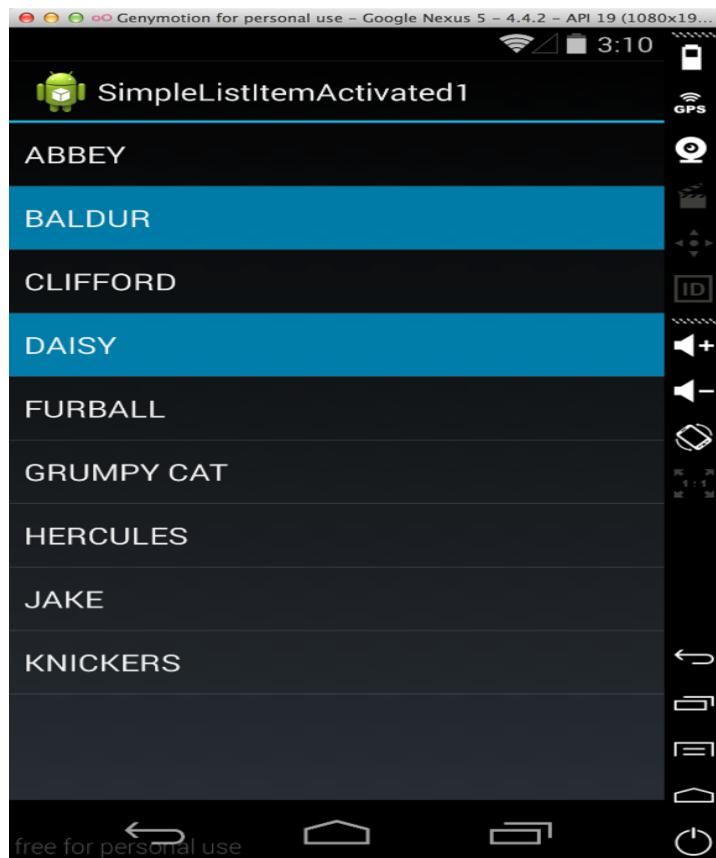


■ `Android.Resource.Layout.SimpleListItemActivated1`

1 `TextView` (`Android.Resource.Id.Text1`)

* multiple or single로 ChoiceMode를 설정해야 한다.

```
this.ListView.ChoiceMode=Multiple
```

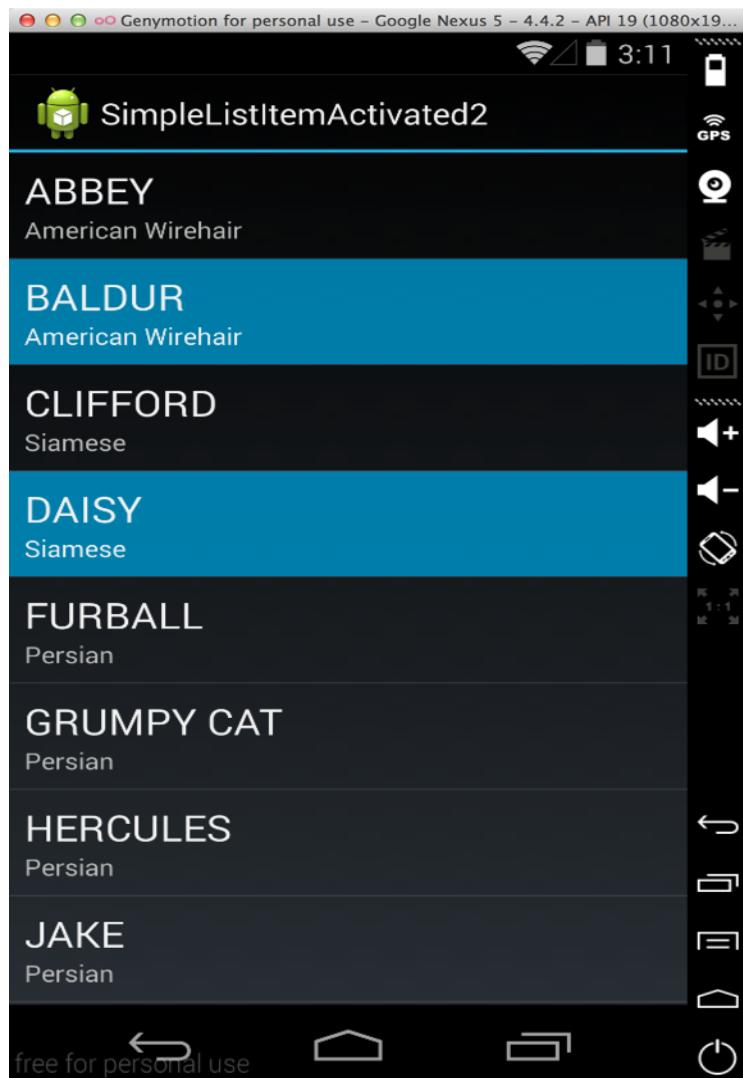


■ Android.Resource.Layout.SimpleListItemActivated2

- 1 TextView (Android.Resource.Id.Text1)
- 1 TextView/Subtitle (Android.Resource.Id.Text2)

* multiple or single로 ChoiceMode를 설정해야 한다.

```
this.ListView.ChoiceMode = ChoiceMode.Multiple;
```

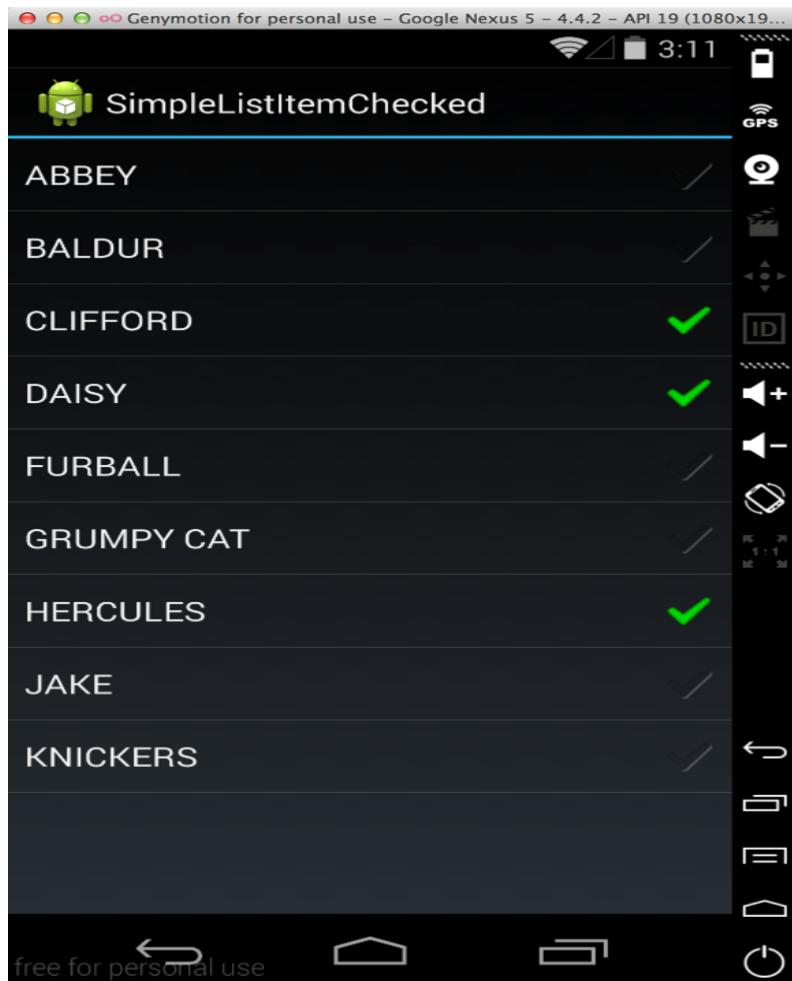


■ `Android.Resource.Layout.SimpleListItemChecked`

1 `TextView (Android.Resource.Id.Text1)`

* Set choice mode to multiple or single

```
this.ListView.ChoiceMode = ChoiceMode.Multiple;
```

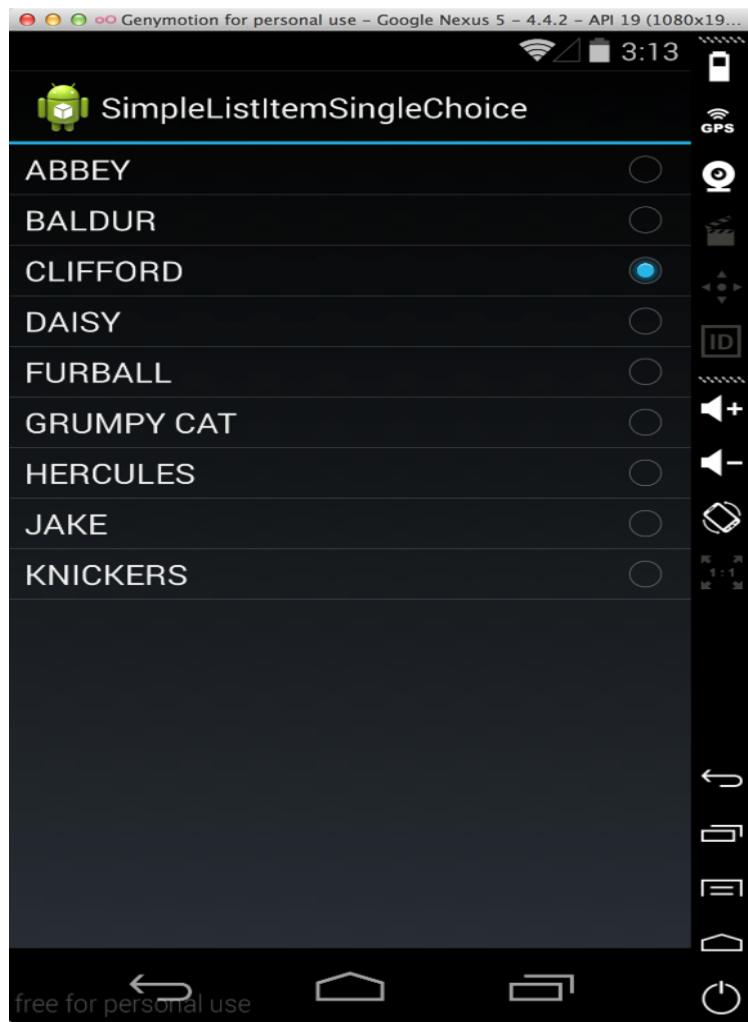


- `Android.Resource.Layout.SimpleListItemSingleChoice`

- 1 `TextView` (`Android.Resource.Id.Text1`)

- * Set choice mode to single

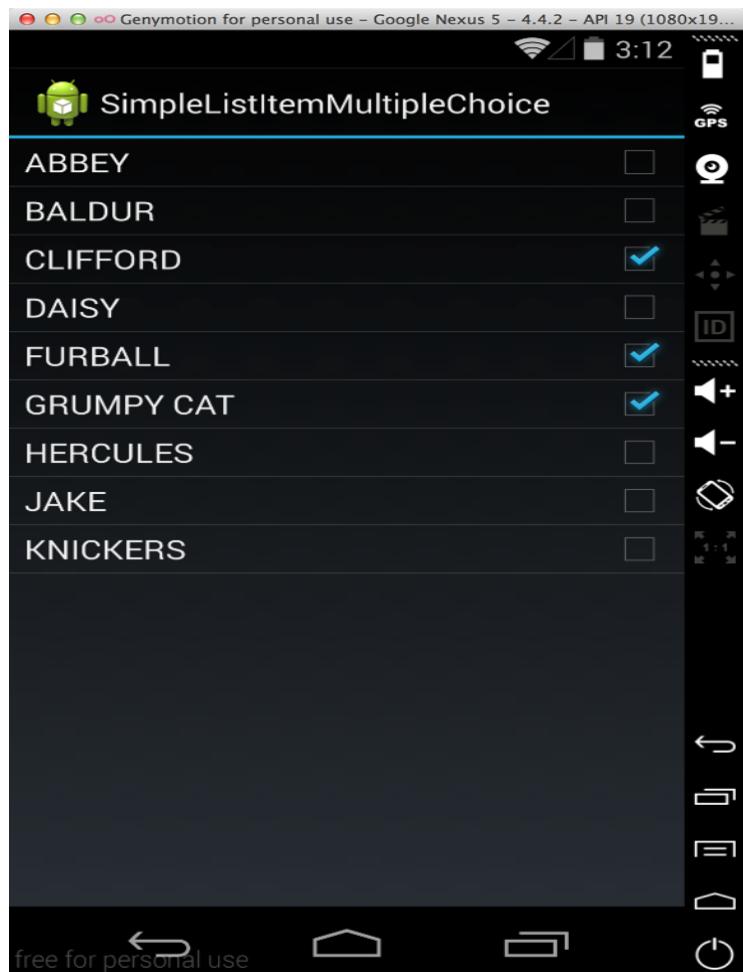
```
this.ListView.ChoiceMode = ChoiceMode.Single;
```



- `Android.Resource.Layout.SimpleListItemMultipleChoice`

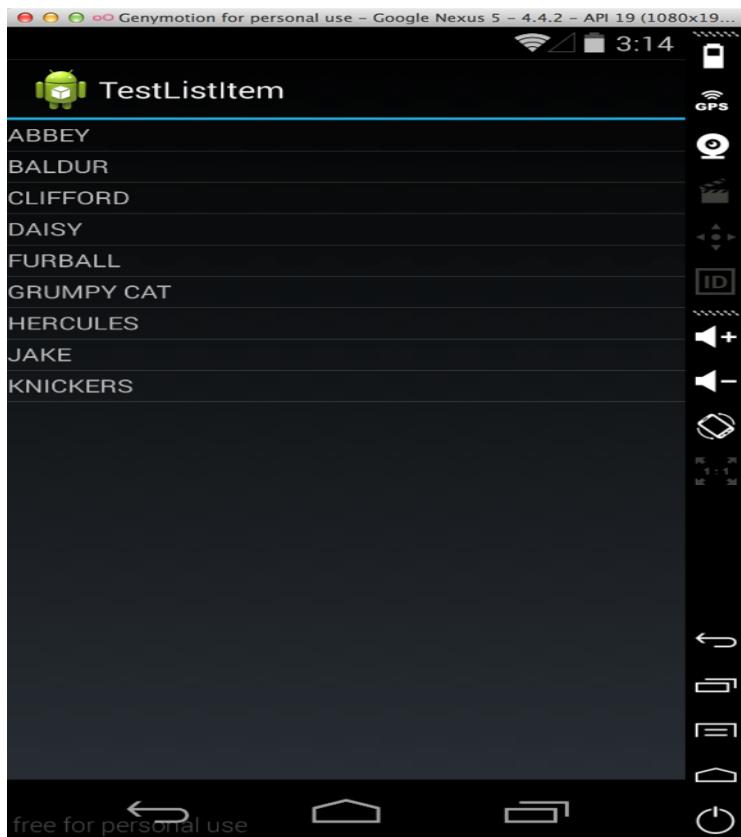
- 1 `TextView` (`Android.Resource.Id.Text1`)

- * Set choice mode to multiple or single



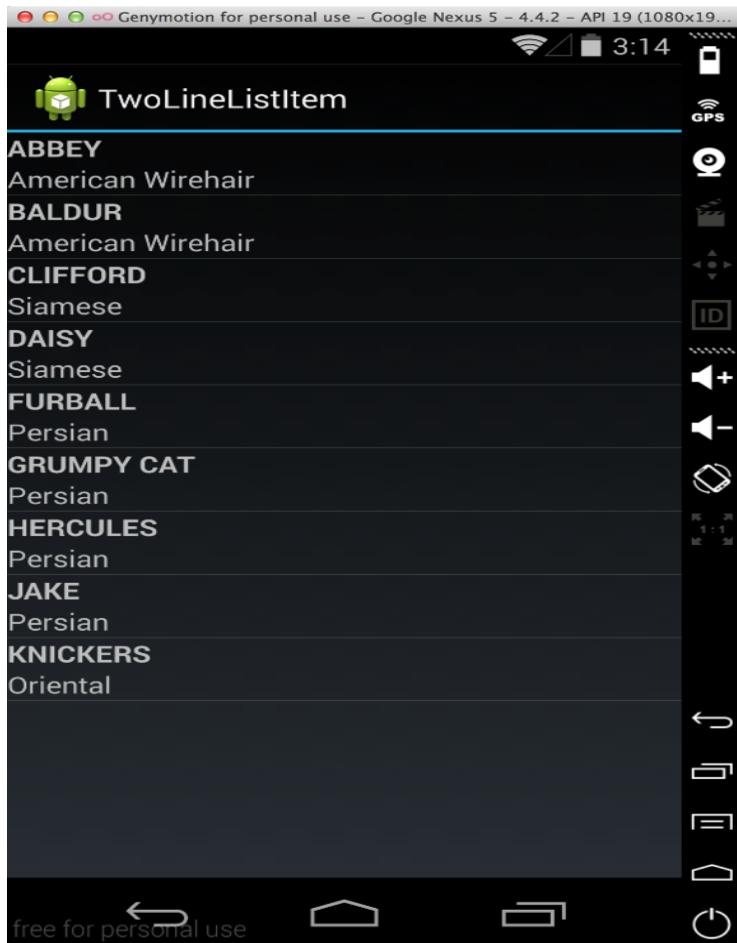
- `Android.Resource.Layout.TestListItem`

`1 TextView @Android.Resource.Id.Text1)`



■ `Android.Resource.Layout.TwoLineListItem`

- 1 `TextView/Title (Android.Resource.Id.Text1)`
- 1 `TextView/Subtitle (Android.Resource.Id.Text2)`



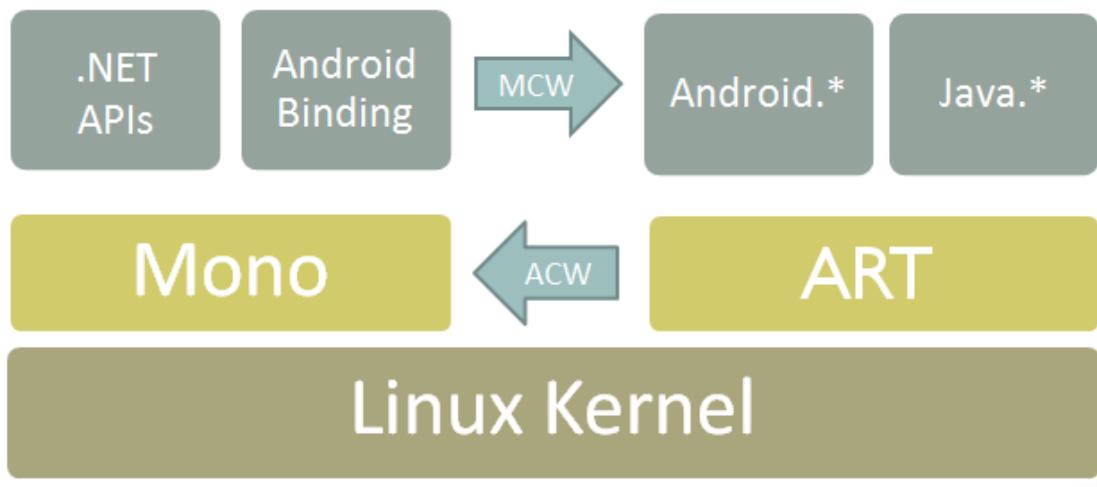
2.5 Xamarin.Android(With .JAR, .AAR, Native Android Library)

2.5.1 Binding Java Library(Consuming Java libraries from C#)

- Android 커뮤니티에는 앱에서 사용할 수 있는 많은 자바 라이브러리가 있다. 바인딩 라이브러리를 만들어 Java 라이브러리를 Xamarin.Android 응용 프로그램에 통합하는 방법에 대해 알아보자
- Android용 라이브러리는 방대하기 때문에 새로운 Android 라이브러리를 만드는 것보다 기존 라이브러리를 사용하는 것이 좋다. Xamarin.Android는 이러한 라이브러리를 사용하는 두 가지 방법을 제공한다.
- 자마린(C#)에서 Java 코드를 호출 할 수 있도록 C# 래퍼로 라이브러리를 자동으로 래핑하는 바인딩 라이브러리를 만드는 방법과 Java Native Interface (JNI)를 사용하여 Java 라이브러리를 직접 호출을 호출하는 방법이 있다. JNI는 Java 코드가 네이티브 응용 프로그램 또는 라이브러리에 의해 호출 될 수 있게 해주는 프레임 워크이다.
- 아래는 첫 번째 방법인 하나 이상의 기존 Java 라이브러리를 응용 프로그램에 연결할 수 있는 어셈블리로 래핑하는 Bindings 라이브러리를 만드는 방법에 대한 설명이다.
- Xamarin.Android는 Managed Callable Wrappers (MCW)를 사용하여 바인딩을 구현하는데

MCW는 Java 코드를 호출해야 할 때 사용되는 JNI 브리지로 호출 가능 래퍼는 Java 유형의 서브 클래스 작성 및 Java 유형의 가상 메소드 대체를 지원한다.

- 마찬가지로 Android 런타임 (ART) 코드가 관리 코드(C#)를 호출하기를 원한다면 Android Callable Wrappers (ACW)라고하는 다른 JNI 브리지를 통해 코드를 호출한다. 아래 다이어그램을 참조하자.



- Bindings 라이브러리는 Managed Callable Wrappers(MCW) for Java 유형을 포함하는 어셈블리이다. 예를 들어, 다음은 Bindings 라이브러리에서 랩핑 하려는 Java 유형 MyClass다.

```
package com.xamarin.mycode;  
  
public class MyClass  
{  
    public String myMethod (int i) { ... }  
}
```

- C#쪽에서 MyClass가 포함 된 .jar 용 Bindings Library를 생성 한 후에 객체를 만들고 메소드를 호출 할 수 있다.

```
var instance = new MyClass ();  
string result = instance.myMethod (42);
```

- Bindings 라이브러리를 만들려면 Xamarin.Android Java Bindings Library 템플릿을 사용하는데 결과로 생성 된 바인딩 프로젝트는 MCW 클래스, jar 파일 및 Android 라이브러리 프로젝트에 포함 된 리소스가 포함 된 .NET 어셈블리를 만든다.

- **JAR, AAR, DEX, APK 차이점**

JAR (Java Archive) : JAR는 해당 플랫폼에서 JAVA 응용 프로그램을 배포하기 위해 고안된 패키지 파일 형식. 컴파일 된 자바 클래스 파일과 MANIFEST와 같은 파일들이 포함되며 기본적으로 ZIP 아카이브 형태이다.

AAR (Android Archive) : Android 라이브러리 프로젝트의 바이너리 배포판입니다. 주로 Java 클래스 파일들만 포함하는 Jar와 달리 리소스 파일들도 포함하고 있으며 이클립스 + 안드로이드 플러그인 형태의 개발에서는 사용불가하며 안드로이드 스튜디오 개발에서에서 사용 가능하다.

DEX (Dalvik Executable) : Java 코드로 작성되어 컴파일된 클래스 파일을 DX(Android dx tool) 도구를 사용해 변환한 파일로 DVM(Dalvik Virtual Machine)을 위한 실행 파일이다. Java 바이트 코드는 달vik 바이트 코드로 변환되며 여러 클래스 파일에 들어있는 중복된 코드들을 재사용하기 때문에 JAR(Java archive) 파일에 비해 필요한 공간이 절반 정도로 크게 줄어든다. Android SDK의 Dex 컴파일러에 의해 JVM 바이트코드를 DVM 바이트코드로 변환하고 모든 클래스파일들을 Dex 파일에 넣는다. DEX는 바이너리 파일 형식으로 컴파일 된다.

안드로이드는 실행 파일 수행에 있어 달vik 가상머신(Dalvik VM)을 사용한다. 달vik 가상머신 상에서 수행되는 애플리케이션들은 각각 분리된 프로세스와 ID를 갖고 있으며 분리된 VM 영역에서 실행된다. 윈도우 환경에서의 실행 파일인 PE(Portable Executable) 파일이 있듯이, 안드로이드 환경하에서도 실행 파일인 DEX(Dalvik Executable) 파일이 있다. 안드로이드는 이 DEX 파일을 달vik 가상머신에서 구동시킨다. 달vik 가상머신은 모바일 기기에서 용량이 적은 메모리를 사용해 실행 파일을 구동할 수 있게 최적화한 가상머신이다. 동시에 여러 가상머신을 실행할 수 있도록 설계돼 있다. 다시 말해 안드로이드 OS는 실행되는 애플리케이션마다 가상머신을 생성해 구동시티는 것이다.

APK (Android Application Package) : 안드로이드 플랫폼에 배포할 수 있도록 설계된 파일 형식으로 확장자는 .apk 이고 ZIP파일 기반인 JAR를 기반으로 한다. 그래서 이 때문에 7-Zip, Alzip 등 압축 프로그램으로 내용을 확인할 수 있다. AndroidManifest.xml 등 리소스 파일들도 포함하는데 이 파일은 APK 내 최상위 폴더에 위치해 있으며 애플리케이션에 필요한 퍼미션(Permosson) 설정 정보를 담고 있다. 주요 컴포넌트(Component) 즉 Activity, Service, Broadcast Receiver, Content Provider에 대한 동작 설정 정보도 포함하고 있다.

- Android 보관 파일 (.AAR) 파일과 Eclipse Android 라이브러리 프로젝트 용 Bindings Libraries 를 만들 수도 있다. 결과로 생성 된 Bindings Library DLL 어셈블리를 참조하면 Xamarin.Android 프로젝트에서 기존 Java 라이브러리를 다시 사용할 수 있다.
- 바인딩 라이브러리에서 형식을 참조 할 때는 바인딩 라이브러리의 네임 스페이스를 사용해야 하고 일반적으로 Java 패키지 이름의 .NET 네임 스페이스 버전 인 C# 소스 파일의 맨 위에 using 지시문을 추가한다. 예를 들어, 바인드 된 .jar의 Java 패키지 이름이 "a.b.c" 라면 "using a.b.c"를 추가하면 된다.
- Android 라이브러리를 바인딩 할 때 다음 사항에 유의해야 한다.
- **라이브러리에 대한 외부 종속성이 있는가?** - Android 라이브러리에 필요한 모든 Java 종속성은 Xamarin.Android 프로젝트에 ReferenceJar 또는 EmbeddedReferenceJar로 포함되어야 하며 모든 네이티브 어셈블리는 EmbeddedNativeLibrary로 바인딩 프로젝트에 추가 되어야 한

다.

- **Android 라이브러리는 어떤 Android API 버전을 타겟팅 하는가?** - Android API 레벨을 "다운 그레이드"할 수는 없다. Xamarin.Android 바인딩 프로젝트가 Android 라이브러리와 동일한 API 수준 (또는 그 이상)을 타겟팅하는지 확인해야 한다.
- **라이브러리를 컴파일하는 데 사용 된 JDK의 버전은 무엇인가?** - Android 라이브러리가 Xamarin.Android에서 사용중인 JDK와 다른 버전으로 작성된 경우 바인딩 오류가 발생할 수 있다. 가능한 경우 Xamarin.Android를 설치하는 데 사용되는 JDK와 동일한 버전의 JDK를 사용하여 Android 라이브러리를 다시 컴파일 하는 것이 좋다.
- Bindings Library를 생성 할 때 Bindings Library 프로젝트에 통합하는 .jar 또는 .AAR 파일에 대한 빌드 작업(Build Action)을 설정하는데 각 빌드 작업은 .jar 또는 .AAR 파일이 사용자의 Bindings 라이브러리에 포함되거나 참조되는 방식을 결정한다.
 - ✓ **EmbeddedJar** - 포함 된 자원으로 .jar를 결과 Bindings Library DLL에 삽입, 가장 간단하고 가장 일반적으로 사용되는 빌드 작업이다. .jar를 자동으로 바이트 코드로 컴파일하고 Bindings 라이브러리에 패키지화 하려면 이 옵션을 사용한다.
 - ✓ **InputJar** - 결과 Bindings 라이브러리 .DLL에 .jar를 내장하지 않으며 Bindings 라이브러리 .DLL은 런타임에 .jar에 종속되는 형태이다. Bindings Library에 .jar를 포함하지 않으려는 경우 (예 : 라이센스상의 이유로)이 옵션을 사용하면 되는데 이 옵션을 사용하는 경우 입력 .jar이 앱을 실행하는 기기에서 사용 가능한지 확인해야 한다.
 - ✓ **LibraryProjectZip** - .AAR 파일을 결과 Bindings 라이브러리 .DLL에 포함한다. 이는 바인딩 된 .AAR 파일의 리소스 (코드는 물론)에 액세스 할 수 있다는 점을 제외하고는 EmbeddedJar와 유사하다. .AAR을 Bindings Library에 임베디드 하려면이 옵션을 사용한다.
 - ✓ **ReferenceJar** - .jar에 대한 참조를 지정한다. 바인딩 된 .jar 또는 .AAR 파일 중 하나가 의존하는 .jar를 가리킨다. 이 참조 .jar은 컴파일 타임 의존성을 만족시키기 위해서만 사용하며 이 빌드 동작을 사용하면 참조하는 .jar에 대한 C# 바인딩이 만들어지지 않고 결과 Bindings 라이브러리 .DLL에 포함되지 않는다. 참조하는 .jar에 대한 바인딩 라이브러리를 만들지만 아직 수행하지 않은 경우 이 옵션을 사용하며 여러 .jar (및 / 또는 .AAR)를 여러 상호 의존 Bindings 라이브러리로 패키징하는 데 유용하다.
 - ✓ **EmbeddedReferenceJar** - 참조하는 .jar를 결과 Bindings 라이브러리 .DLL에 포함한다. 입력 .jar (또는 .AAR)과 Bindings 라이브러리의 모든 참조 .jar (둘 다)에 대해 C# 바인딩을 만들려는 경우에 사용한다.
 - ✓ **EmbeddedNativeLibrary** - 네이티브 .so를 바인딩에 포함한다. 바인딩되는 .jar 파일에 필요한 .so 파일에 사용하며 Java 라이브러리에서 코드를 실행하기 전에 .so 라이브러리를 수동으로 로드해야 할 수도 있다. 아래 내용을 참조하자.
- **Including a Native Library in a Binding**
 - ✓ Java 라이브러리를 바인딩하는 일부로 .so 라이브러리를 Xamarin.Android 바인딩 프로젝트에 포함해야 할 수도 있는데 간혹 래핑 된 Java 코드가 실행되면 Xamarin.Android가 JNI 호출을 수행하지 못하고 java.lang.UnsatisfiedLinkError : Native method not found 오류 메시지가 응용 프로그램의 logcat 출력에 나타날 수 있다.

- ✓ 이 경우 Java.Lang.JavaSystem.LoadLibrary를 호출하여 .so 라이브러리를 수동으로 로드해야 한다. 예를 들어, Xamarin.Android 프로젝트가 EmbeddedNativeLibrary 빌드 작업으로 바인딩 프로젝트에 포함 된 라이브러리 libpocketsphinx_jni.so를 공유한다고 가정하면 공유 라이브러리를 사용하기 전에 다음 방법으로 .so 라이브러리를 로드한다.

```
Java.Lang.JavaSystem.LoadLibrary("pocketsphinx_jni");
```

■ Adapting Java APIs to C#

- ✓ Xamarin.Android Binding Generator는 .NET 패턴과 일치하도록 일부 Java 관용구 및 패턴을 변경하는데 아래는 Java가 C# / .NET에 매핑되는 방법을 설명한다.
- ✓ Java의 Setter / Getter 메소드는 .NET의 속성이다.
- ✓ Java의 필드는 .NET의 속성이다.
- ✓ Java의 Listener / Listener 인터페이스는 .NET의 이벤트이며 콜백 인터페이스의 메서드 매개 변수는 EventArgs 하위 클래스로 표시된다.
- ✓ Java의 Static Nested 클래스는 .NET의 Nested 클래스이다.
- ✓ Java의 Inner 클래스는 C#에서 인스턴스 생성자가 있는 Nested 클래스이다.

■ Binding Example

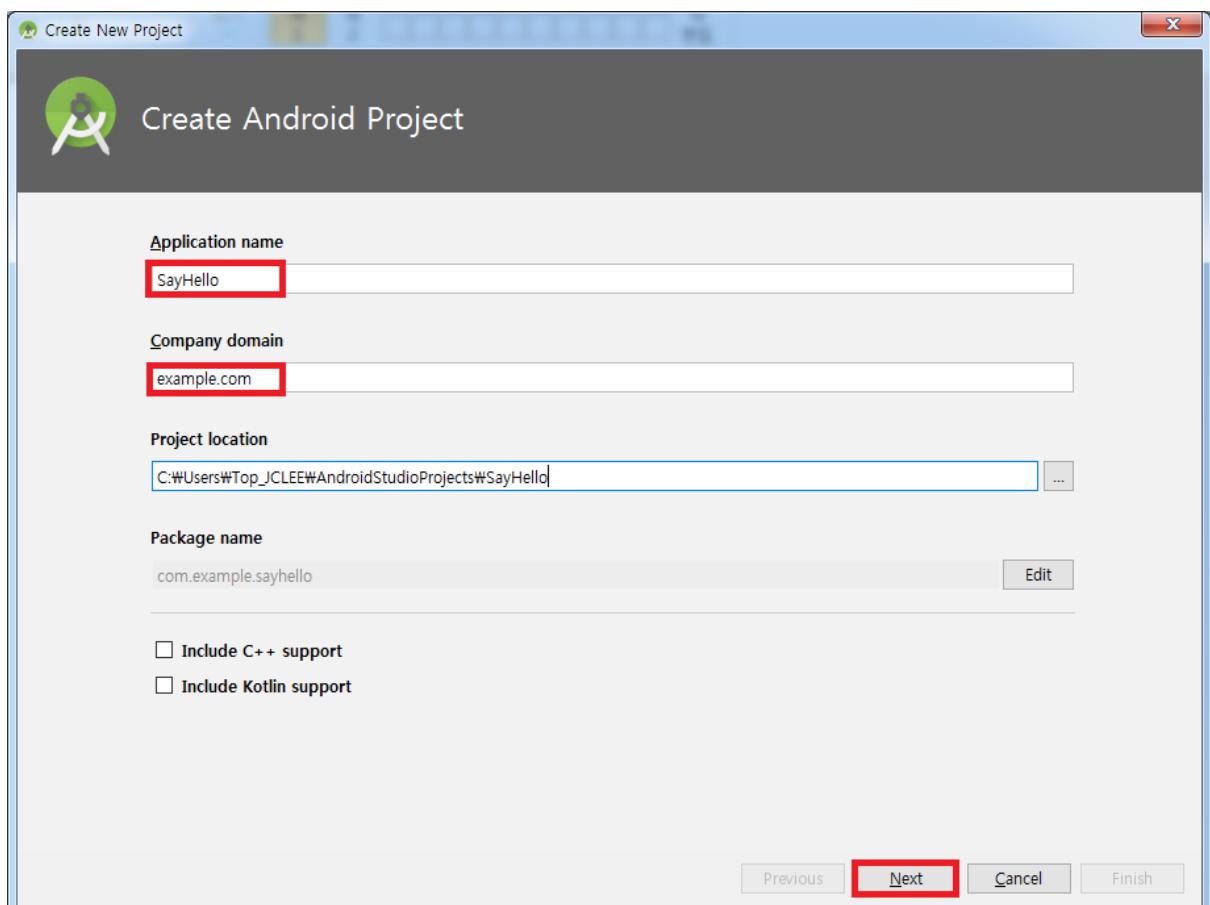
- ✓ **.JAR 바인딩은** jar 파일에 대한 바인딩이다.(https://developer.xamarin.com/guides/android/advanced_topics/binding-a-java-library/binding-a-jar/)
- ✓ **.AAR 바인딩은** .AAR 파일에 대한 바인딩 라이브러리를 만들기위한 방법으로 Android Studio 라이브러리를 바인딩 하려면 방법을 사용한다.(https://developer.xamarin.com/guides/android/advanced_topics/binding-a-java-library/binding-an-aar/)
- ✓ **이클립스 라이브러리 프로젝트 바인딩은** Android 라이브러리 프로젝트에서 바인딩 라이브러리를 만들 때 사용하며 Eclipse Android Library Project를 바인딩 할 때 사용한다.(https://developer.xamarin.com/guides/android/advanced_topics/binding-a-java-library/binding-a-library-project/)
- ✓ **Bindings 커스터마이징에서는** 빌드 오류를 해결하고 결과로 나오는 API를 구조화 하기 위해 바인딩을 수동으로 수정하는 방법을 설명하여 "C#과 유사"합니다.(https://developer.xamarin.com/guides/android/advanced_topics/binding-a-java-library/customizing-bindings/)

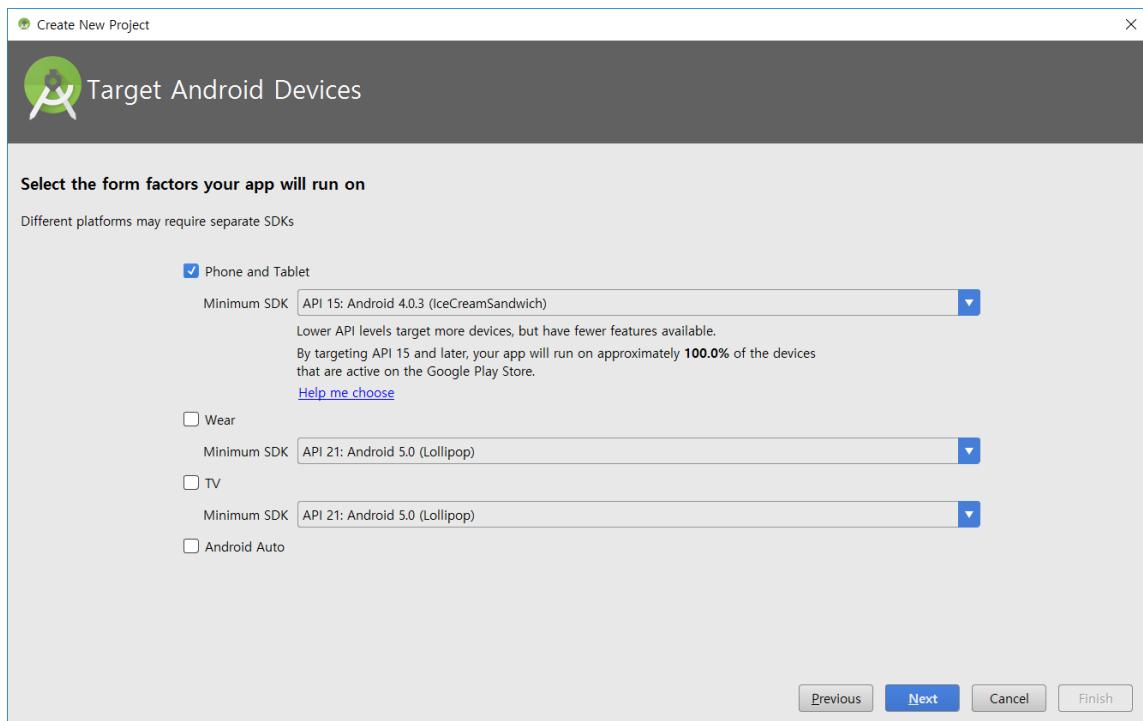
2.5.2 Xamarin.Android EmbeddedJar Binding(안드로이드 JAR 라이브러리 바인딩)

- Java 라이브러리는 대개 .JAR (Java Archive) 형식으로 패키지화 되지만 Java Bindings 라이브러리에서 .JAR 패키지를 패키지화하여 Xamarin.Android 응용 프로그램에서 해당 기능을 사용할 수 있다.
- Java Bindings 라이브러리의 목적은 .JAR 파일의 API를 자동 생성 코드 래퍼를 통해 C # 코드에서 사용할 수 있도록 하는 것이다.
- Xamarin Tooling(자마린 툴링)은 하나 이상의 입력 .JAR 파일에서 Bindings Library를 생성 할

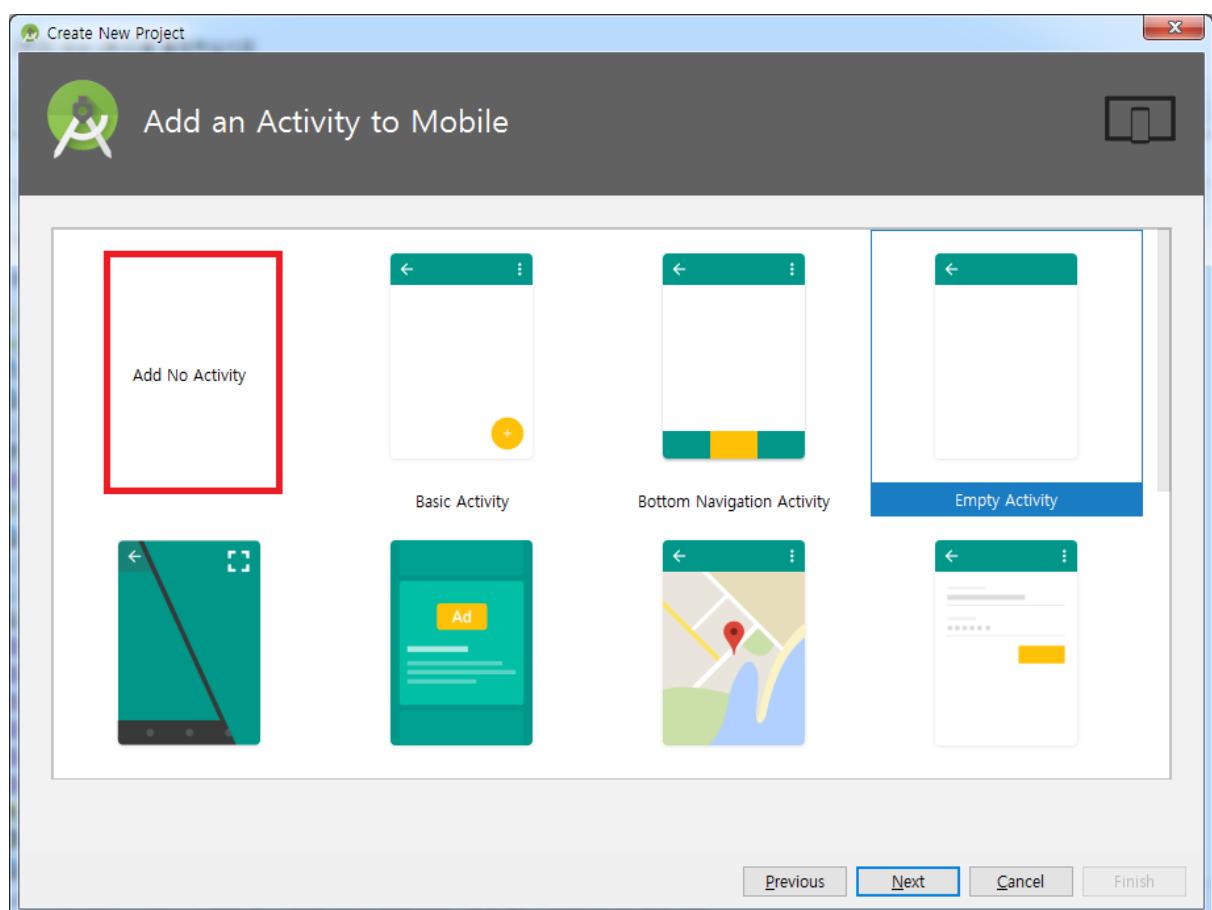
수 있으며 바인딩 라이브러리 (.DLL 어셈블리)에는 아래 내용이 포함되어 있다.

- 원본 JAR 파일의 내용
- JAR 파일 내에 Java Type을 래핑하는 C# Type인 Managed Callable Wrappers(MCW), 생성 된 MCW 코드는 JNI (Java Native Interface)를 사용하여 API 호출을 기본 .JAR 파일로 전달한다.
- 기본적으로 안드로이드와 함께 사용하도록 지정된 모든 JAR 파일에 대한 바인딩 라이브러리를 작성할 수 있지만 Xamarin Tooling(자마린 툴링)은 현재 Android가 아닌 Java 라이브러리 바인딩을 지원하지 않는다.
- DLL이 런타임시 JAR에 종속되도록 JAR 파일의 내용을 포함하지 않고 바인딩 라이브러리를 빌드하도록 선택할 수도 있다.
- 1. 안드로이드 스튜디오에서 새로운 프로젝트를 생성하고 location, application name, company domain을 기술하고 다음 페이지 Target Android Device에서 “Phone and Tablet” 선택
(안드로이드 최신버전 3.1.4 이상에서는 지우너하지 않으므로 SKIP, 실습하려면 하위버전에서하세요)





- Add an Activity to Mobile 화면에서 “Add No Activity”를 선택 후 “Finish” 클릭



- 자마린으로 바인딩 될 SayHello 인터페이스 및 SayHelloImpl 구현클래스를 작성하자.

```

SayHello [C:\Users\Top_CLEEE\AndroidStudioProjects\SayHello] - ...app\src\main\java\com\example\sayhello\SayHello [app] - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
SayHello app src main java com example sayhello SayHello
1 package com.example.sayhello;
2
3 public interface SayHello {
4     public String sayHello(String name);
5 }
6

```

The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it is the code editor with the file 'SayHello.java' open. The code defines a public interface named 'SayHello' with a single method 'sayHello(String name)'. The code editor has syntax highlighting and a status bar at the bottom indicating the build was successful.

```

SayHello [C:\Users\Top_CLEEE\AndroidStudioProjects\SayHello] - ...app\src\main\java\com\example\sayhello\SayHelloImpl [app] - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
SayHello app src main java com example sayhello SayHelloImpl
1 package com.example.sayhello;
2
3 public class SayHelloImpl implements SayHello {
4     public String sayHello(String name) {
5         return "Hello~ " + name;
6     }
7 }
8

```

The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it is the code editor with the file 'SayHelloImpl.java' open. The code implements the 'SayHello' interface from the previous screenshot, returning a string that includes the input name. The code editor has syntax highlighting and a status bar at the bottom indicating the build was successful.

- Build.gradle(Module : sayhello)

```
apply plugin: 'com.android.library'
```

```

//task to delete the old jar
task deleteJar(type:Delete){
    delete 'libs/sayhello.jar'
}

//task to export contents as jar
task exportJar(type:Copy){
    //from('build/intermediates/bundles/release/')           //안드로이드 3.1 하위버전
    from('build/intermediates/intermediate-jars/release/')  //안드로이드 3.1 이상버전
    into('libs/')
    include('classes.jar')
    rename('classes.jar', 'sayhello.jar')
}
exportJar.dependsOn(deleteJar, build)

android {
    compileSdkVersion 28
    defaultConfig {
        //응용프로그램이 아니라 라이브러리 이므로 생략
        //applicationId "com.example.sayhello"
        minSdkVersion 15
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0-rc02'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
}

```

- 안드로이드 스튜디오 우측의 Gradle을 클릭 후 :sayhello -> Tasks -> other -> exportJar 태스크를 더블 클릭하여 실행(exportJar Task가 보이지 않으면 Gradle Project 상단 좌측의 refresh 버튼을 클릭하자.)

```

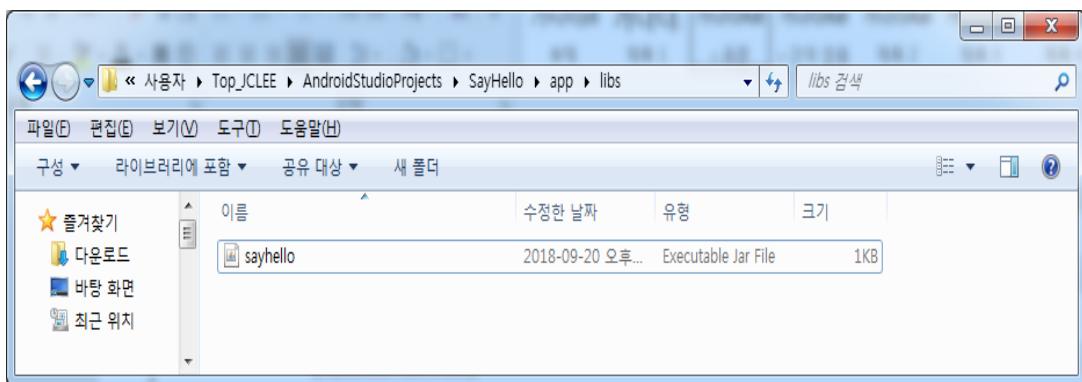
apply plugin: 'com.android.library'
//task to delete the old jar
task deleteJar(type:Delete){
    delete 'libs/sayhello.jar'
}
//task to export contents as jar
task exportJar(type:Copy){
    from('build/intermediates/bundles/release/')
    into('libs/')
    include('classes.jar')
    rename('classes.jar', 'sayhello.jar')
}
exportJar.dependsOn(deleteJar, build)

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.example.sayhello" //응용프로그램이 아니라 라이브러리 이므로 생략
        minSdkVersion 15
        targetSdkVersion 28
    }
}

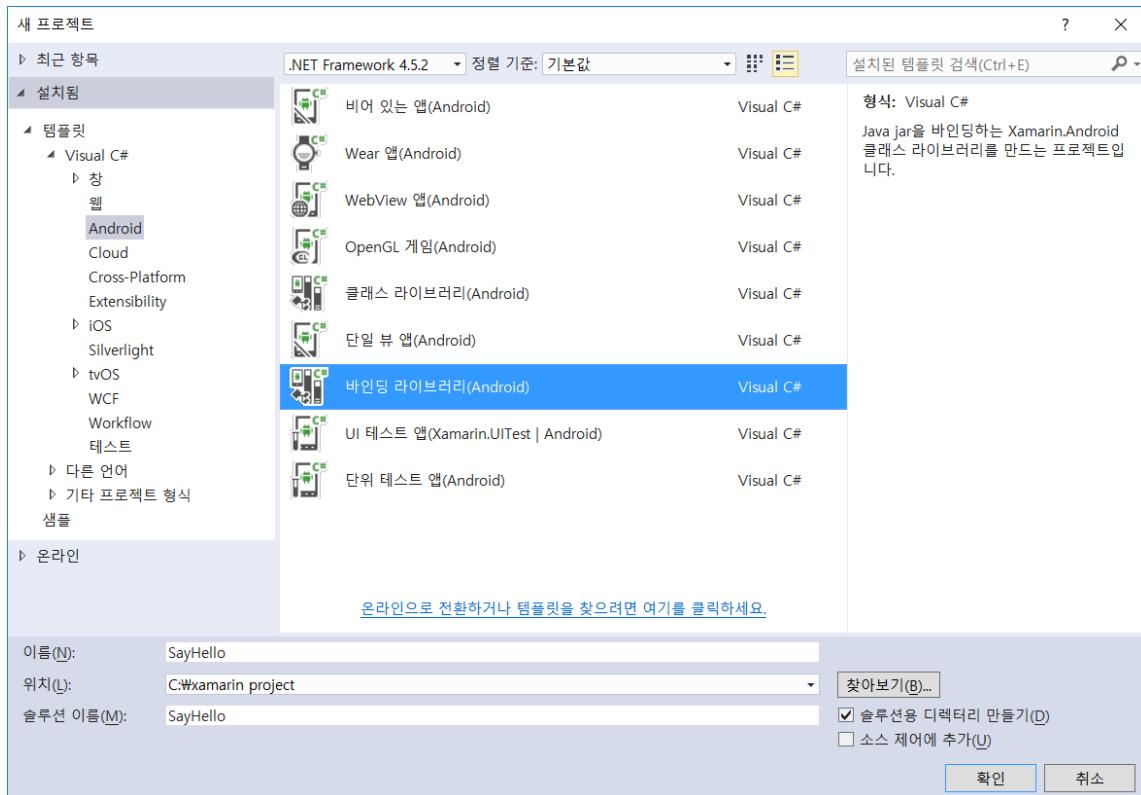
BUILD SUCCESSFUL in 7s

```

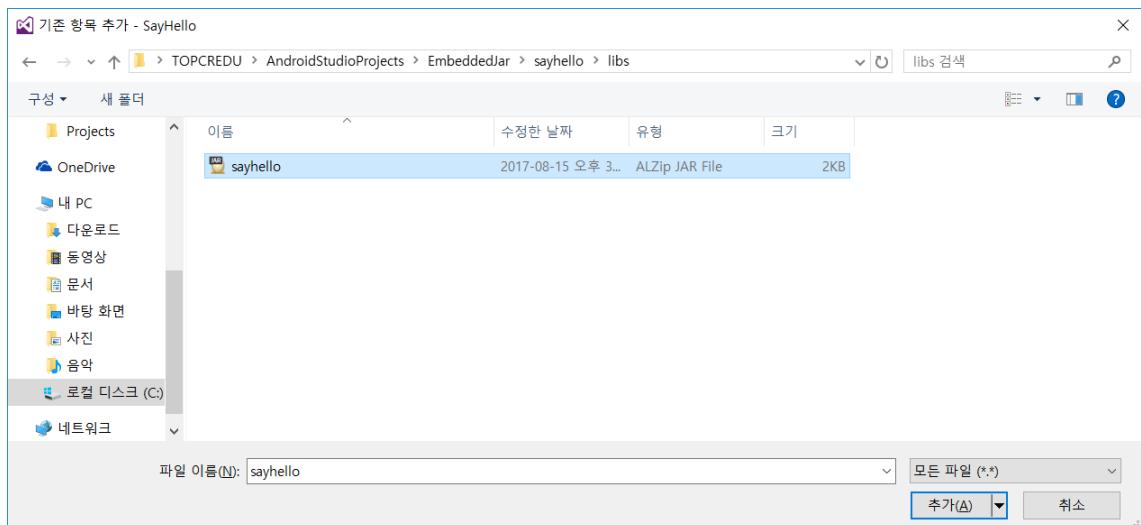
- 이제 sayhello\libs 폴더에 sayhello.jar 파일이 생성되었다.

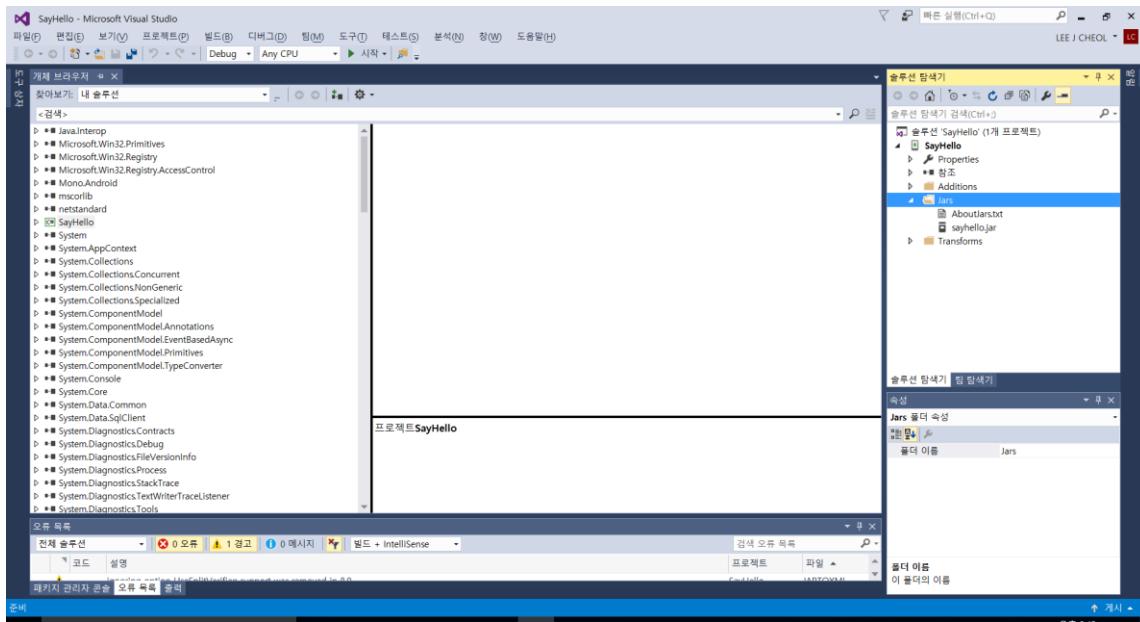


- 2. Visual Studio에서 Android → Binding Library(Android) 프로젝트 생성

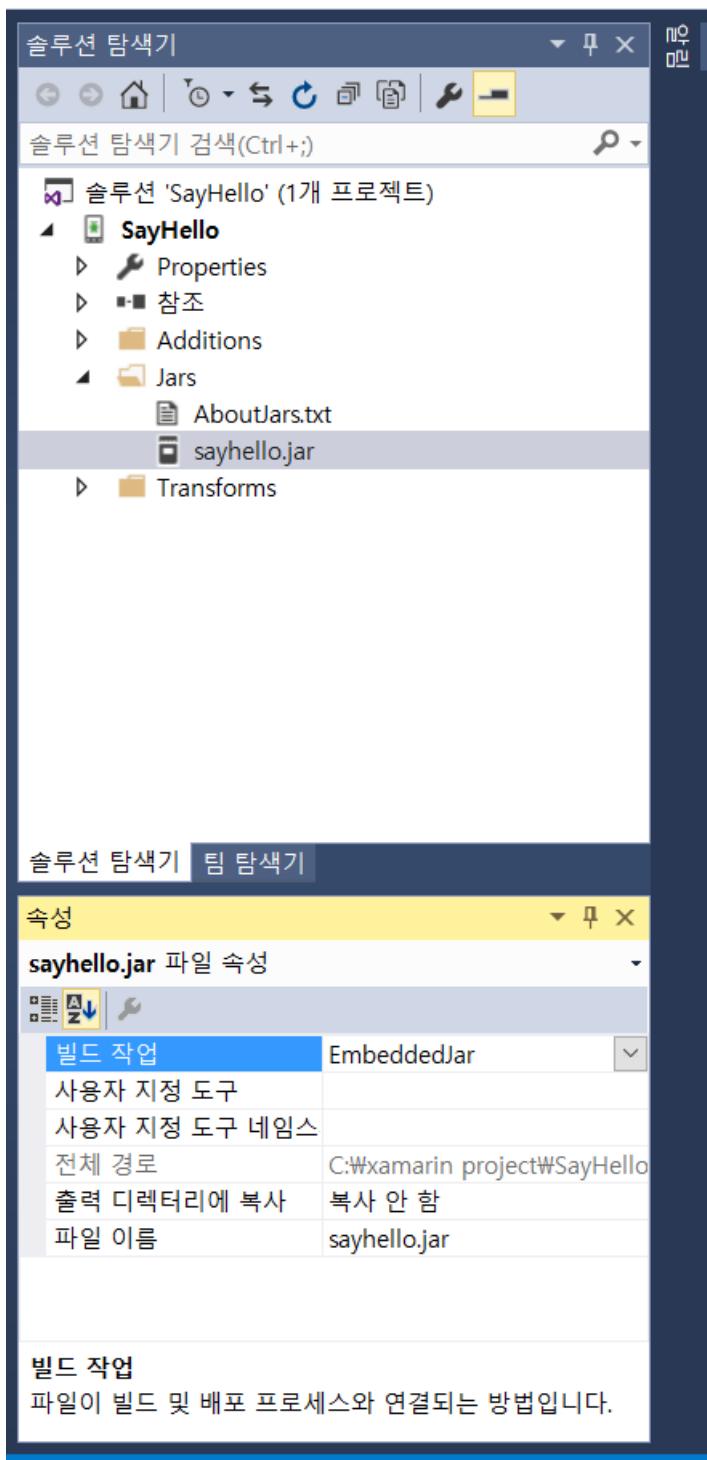


- Bindings Library (BL)는 Managed Callable Wrappers for Java 유형을 포함하는 어셈블리이므로 C# 호출을 통해 Java 코드를 호출 할 수 있다.
- 이전 프로젝트에서 생성한 sayhello.jar 파일을 Jars 디렉터리에 추가한다. **Jars -> 추가 -> 기존항목**



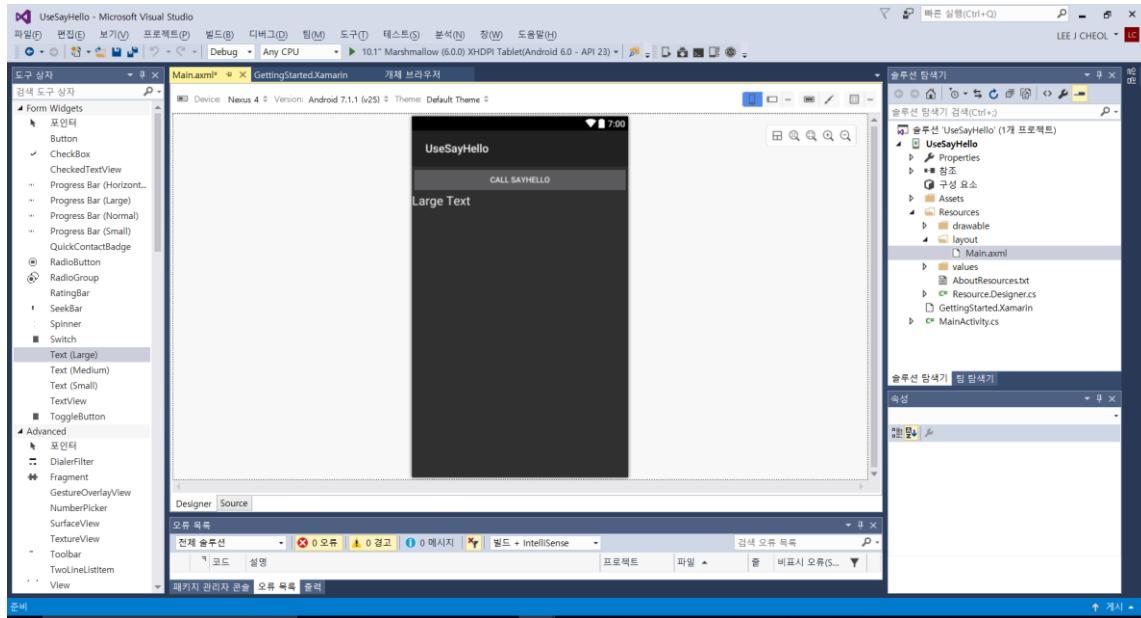


- Java Bindings 라이브러리 프로젝트를 작성할 때 .JAR 파일이 바인딩 라이브러리에 임베드되는지 또는 별도로 패키지되는지 여부를 “**빌드작업(Build Action)**” 속성에서 지정해야 한다.
 - ✓ **EmbeddedJar** - .JAR 파일이 라이브러리에 임베드 된다.
 - ✓ **InputJar** - .JAR 파일은 Bindings 라이브러리와 별도로 유지된다.
- EmbeddedJar 형식은 JAR의 Java 바이트 코드가 Dex 바이트 코드로 변환되고 APK에 (Managed Callable Wrappers와 함께) 포함된다.
- .JAR 파일을 바인딩 라이브러리와 별도로 유지하려면 InputJar 옵션을 사용할 수 있지만 응용 프로그램을 실행하는 장치에서 .JAR 파일을 사용할 수 있어야 한다.
- 일반적으로 .JAR이 자동으로 바인딩 라이브러리에 포함되어 패키징 되도록 **EmbeddedJar** 빌드 작업을 사용한다.

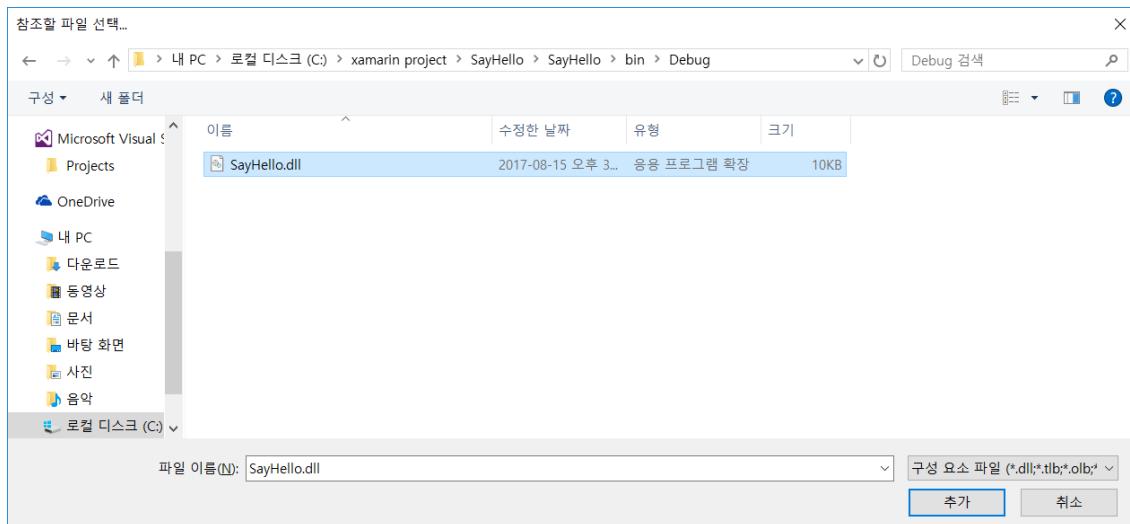


- 솔루션 빌드, 주의 : BL은 Android 라이브러리와 동일한 또는 그 이상의 Android API 레벨을 타겟팅 해야한다
- 3. 이제 C # 코드에서 Java 라이브러리를 사용할 수 있는데 **Xamarin.Android 프로젝트**를 생성하자. **Android -> 비어있는 앱(blank app)** 선택 후 프로젝트 이름을 **“UseSayHello”**라고 하자.
- Activity_main을 더블클릭 하여 아래와 같이 화면을 만들자. (button, Text(Large) 두개의 컨트롤

을 사용하여 컨트롤의 이름은 default로 두자.)



- BL(바인딩 라이브러리) 프로젝트를 참조추가 하자. **참조 -> 참조추가**, 찾아보기에서 C:\xamarin\project\SayHello\SayHello\bin\Debug의 SayHello.dll을 참조추가 하면 된다.



- 아래 MainActivity의 코드에서 자바쪽에서 만든 SayHello를 사용했다.

```
using Android.App;
using Android.Widget;
using Android.OS;

// 원래 자바에서 소문자로 패키지를 만들었지만 바인딩 라이브러리에
// 포함될 때 단어의 첫글자가 대문자로 변환된다.
using Com.Example.SayHello;
```

```

namespace UseSayHello
{
    [Activity(Label = "UseSayHello",
        MainLauncher = true,
        Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        // 자바의 SayHello 인터페이스는 이름 앞에
        // I가 붙어서 바인딩 된다.
        ISayHello hello;

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            // Set our view from the "main" layout resource
            SetContentView (Resource.Layout.Main);

            hello = new SayHelloImpl();

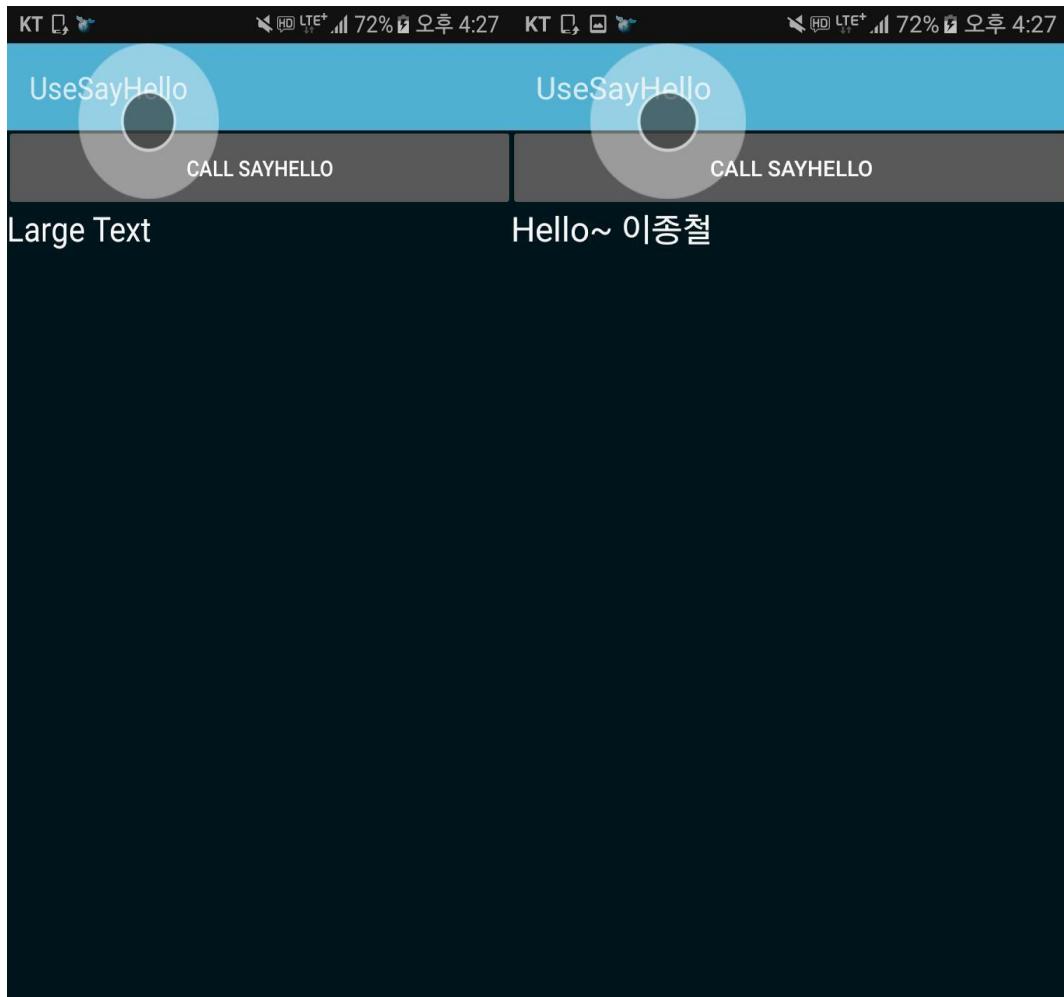
            Button button1 = FindViewById<Button>(Resource.Id.button1);
            TextView textView1 = FindViewById<TextView>(Resource.Id.textView1);

            // 자바에서 sayHello라는 메소드 명으로 만들었지만
            // 바인딩 라이브러리에 포함될 때 SayHello로 변경됨.
            button1.Click += (sender, args) =>
                textView1.Text = hello.SayHello("이종철");

        }
    }
}

```

■ 실행결과

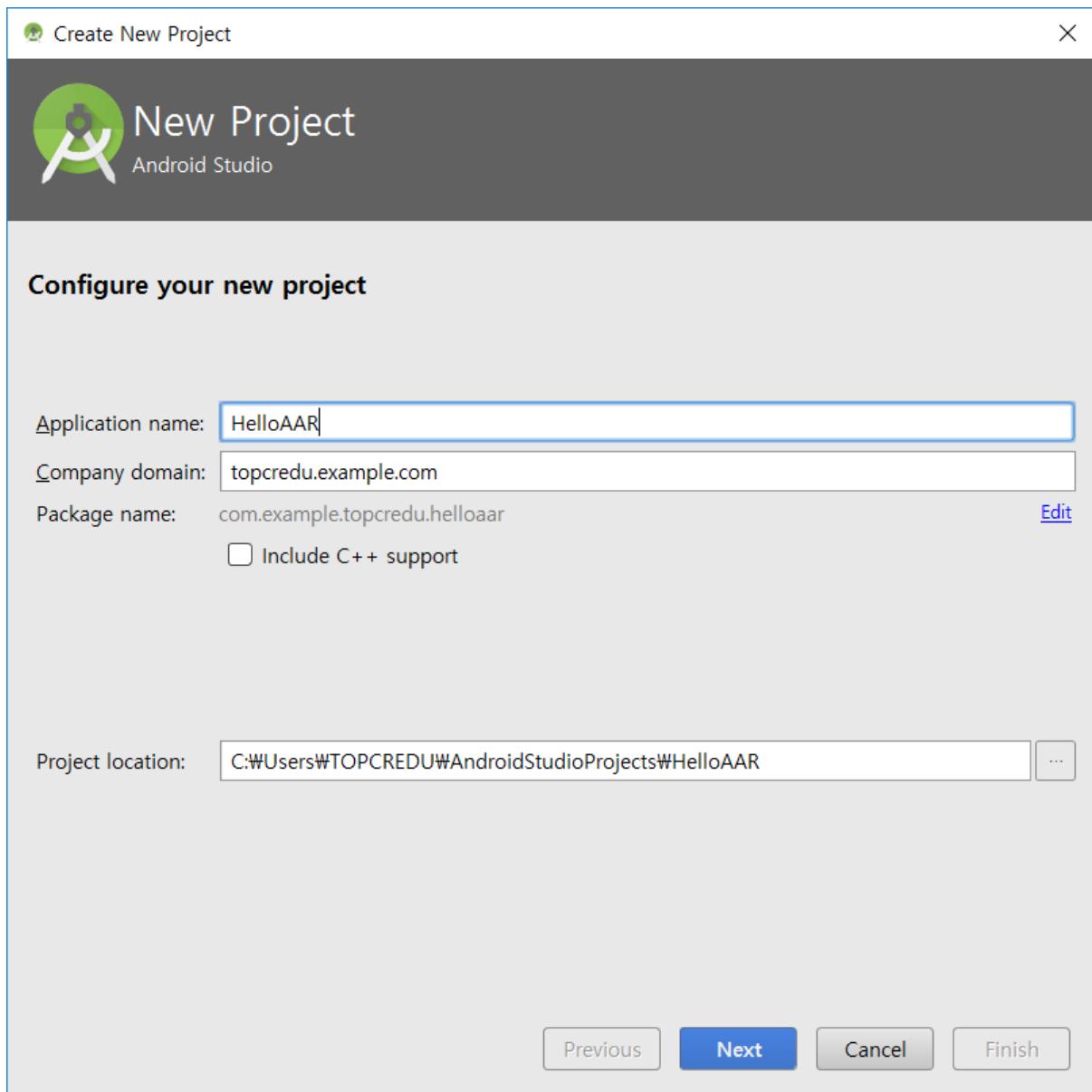


2.5.3 Xamarin.Android .AAR Binding(안드로이드 .AAR File을 자마린 바인딩 자바 라이브러리로 구현 후 Xamarin.Android 프로젝트에서 호출하기)

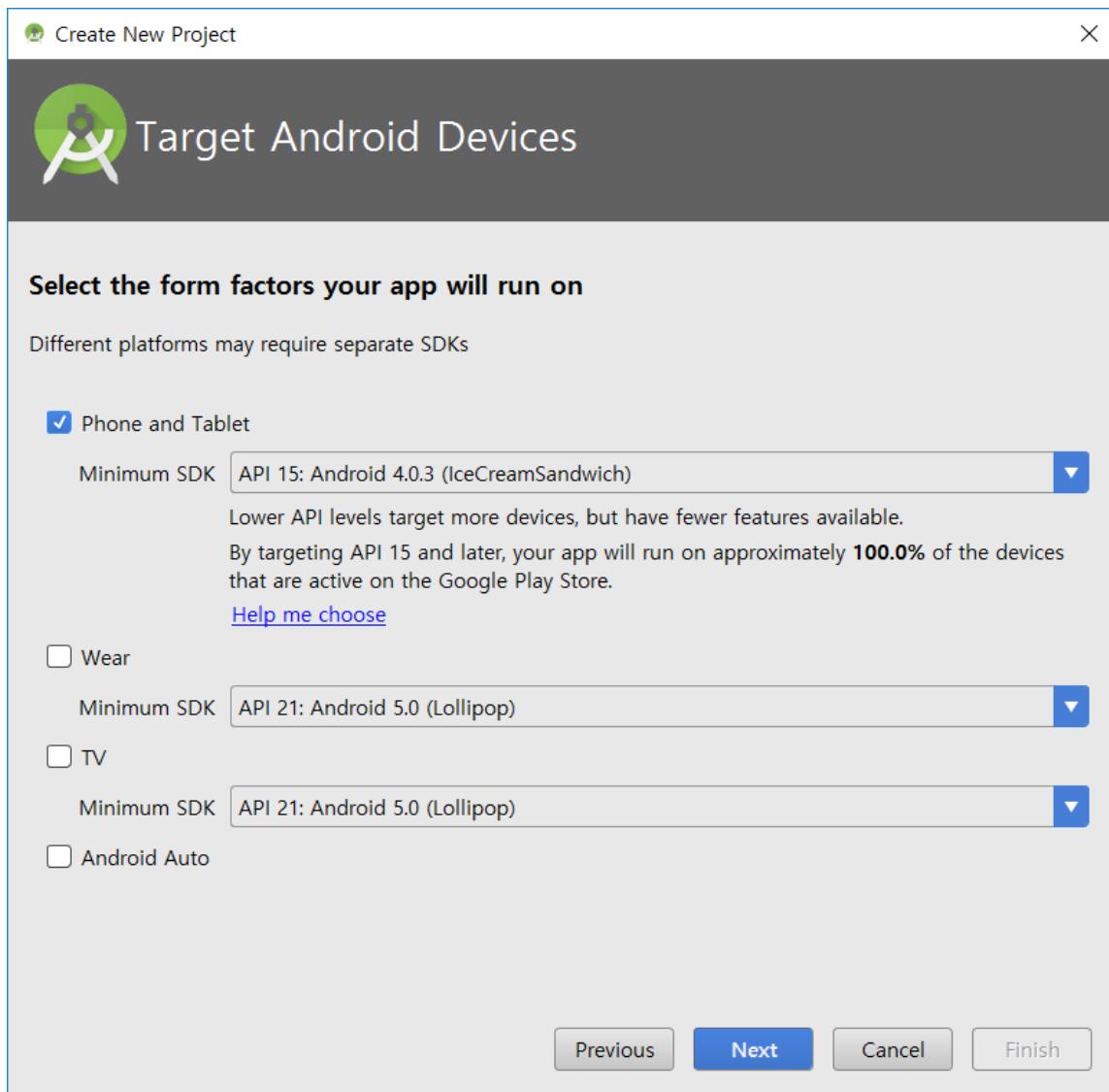
- 안드로이드 아카이브(AAR) 파일은 안드로이드쪽의 라이브러리 형식의 파일이며 아래 내용을 포함하는 ZIP 아카이브 형식의 파일이다.
 - 컴파일된 자바 코드
 - 리소스 ID
 - 자원
 - 메타 데이터(Activity 선언, 사용권한등)
- 자마린쪽의 바인딩 프로젝트에는 하나의 .AAR 파일만 포함될 수 있으므로 다른 .AAR에 대한 .AAR 종속성이 있는 경우, 바인딩 프로젝트에 포함된 다음 참조되어야 한다.

1. 안드로이드 스튜디오에서 .AAR 파일 생성

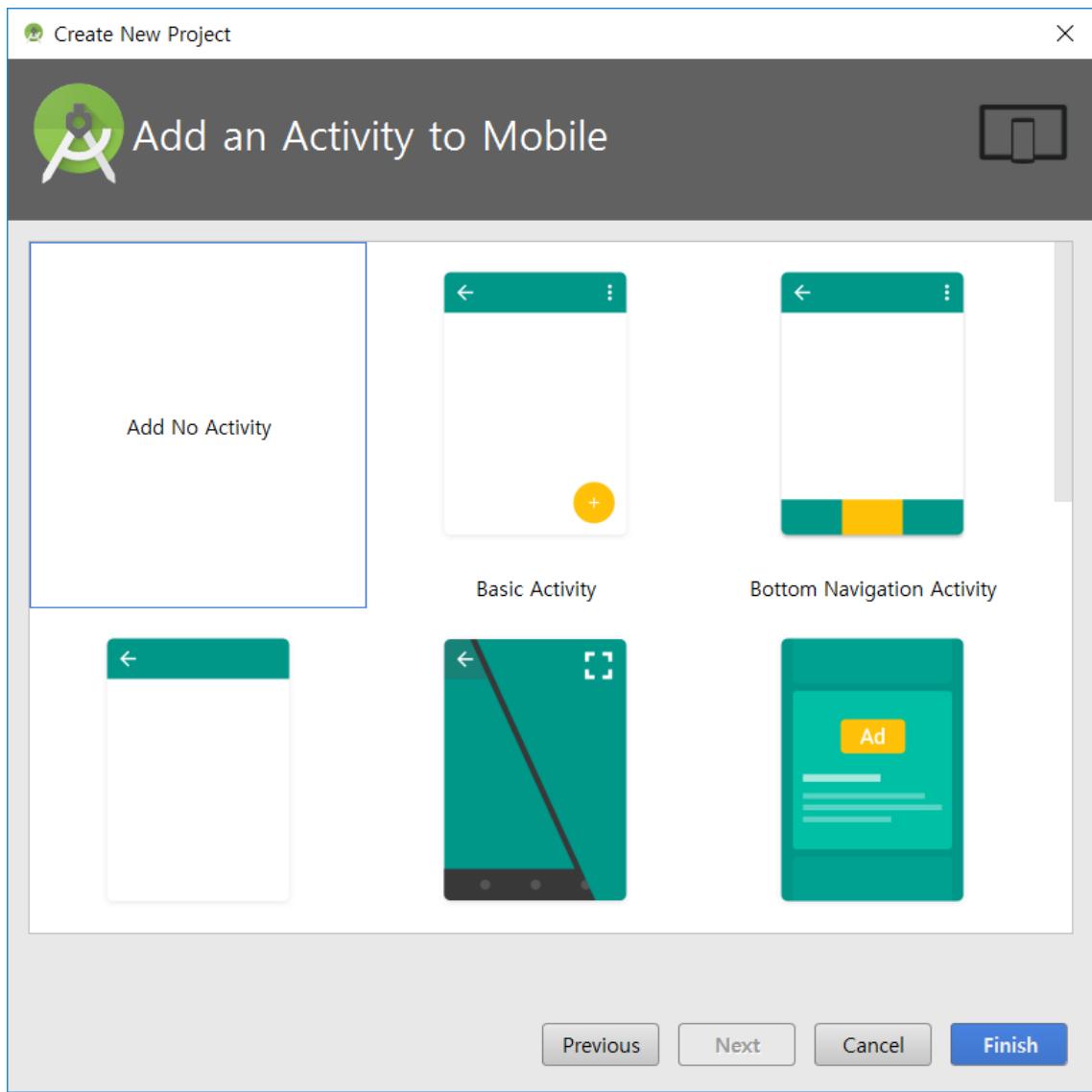
- "HelloAAR"이라는 이름으로 새프로젝트 생성



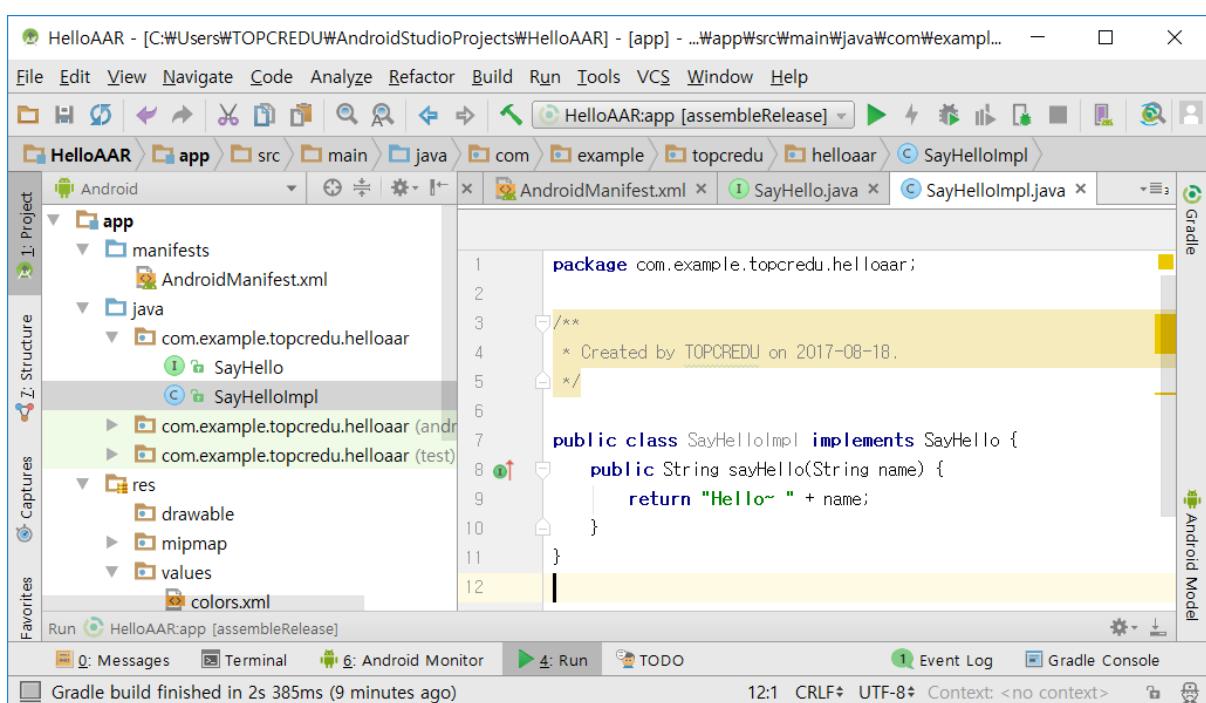
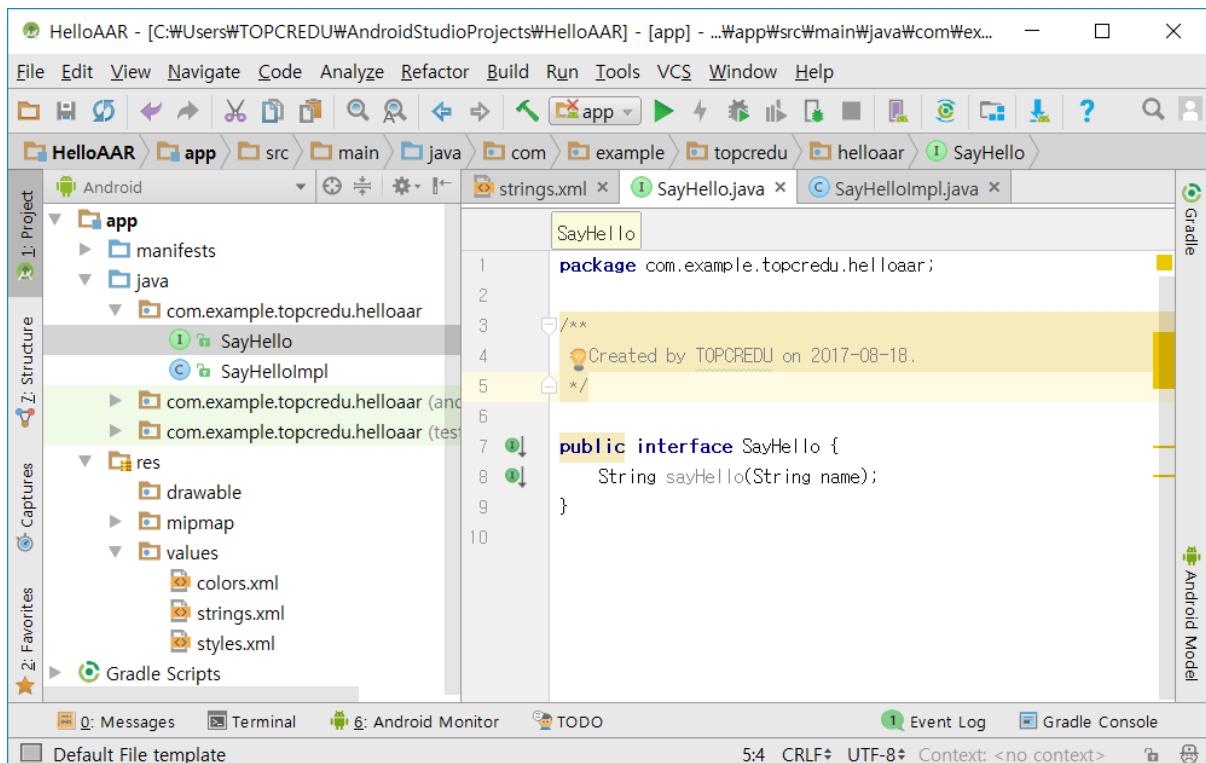
- 다음 화면에서 “Phone And Tablet” 선택



- "Add No Activity" 선택



■ 인터페이스(SayHello.java), 구현클래스(SayHelloImpl.java) 작성



- **AndroidManifest.xml**에서 `android:theme="@style/AppTheme"`를 삭제, `values` >> `styles.xml`을 삭제한다. (AppTheme를 참조하므로 Xamarin.Android App에서 AppTheme을 찾을 수 없다는 오류가 발생한다.)
- **build.gradle**(Module:app)에서 플러그인을 `com.android.library`로 변경, defaultConfig의 `applicationId`를 삭제한다.

```
apply plugin: 'com.android.library'

android {
    compileSdkVersion 26
    buildToolsVersion "26.0.1"
    defaultConfig {
        minSdkVersion 15
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:26.+'
    testCompile 'junit:junit:4.12'
}
```

- 우상단 “Gradle” 버튼을 클릭하여 **assembleRelease** 태스크를 더블 클릭하여 실행하자.

```

HelloAAR - [C:\Users\TOPCREDU\AndroidStudioProjects\HelloAAR] - build.gradle - Android Studio 2.3.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HelloAAR app build.gradle
1 apply plugin: 'com.android.library'
2
3 android {
4     compileSdkVersion 26
5     buildToolsVersion "26.0.1"
6     defaultConfig {
7         minSdkVersion 15
8         targetSdkVersion 26
9         versionCode 1
10        versionName "1.0"
11        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
12    }
13    buildTypes {
14        release {
15            minifyEnabled false
16            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
17        }
18    }
}

```

Gradle projects

- HelloAAR
 - HelloAAR (root)
 - app
 - Tasks
 - android
 - build
 - assemble
 - assembleAndroidTest
 - assembleDebug
 - assembleRelease**
 - build
 - buildDepends
 - buildNeeded
 - clean
 - cleanBuildCache
 - compileDebugAndroidTest
 - compileDebugSources
 - compileDebugUnitTest

Run HelloAAR.app [assembleRelease]

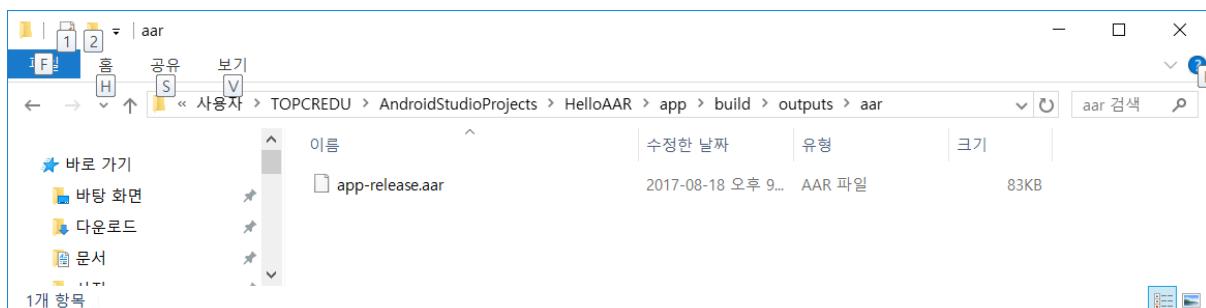
Build Variants

Messages Terminal Android Monitor Run TODO

Gradle build finished in 11s 376ms (a minute ago)

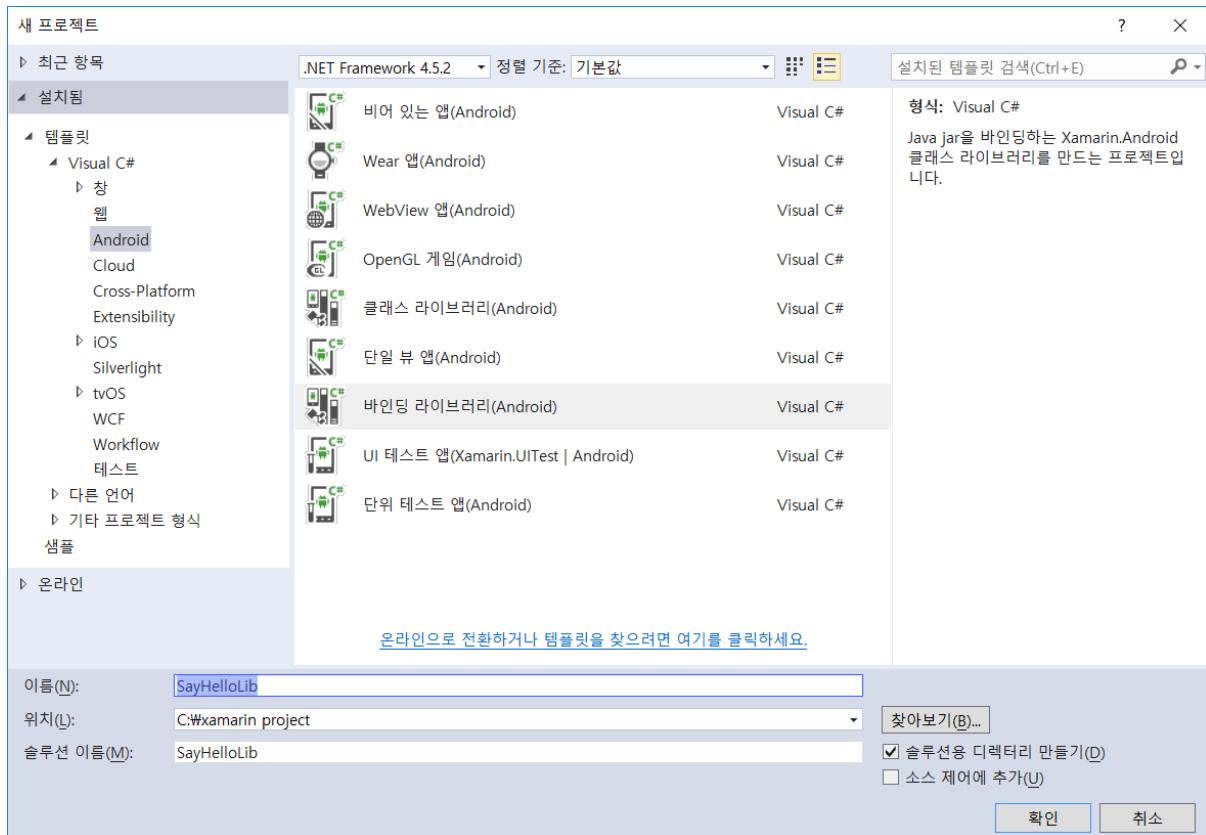
14:18 CRLF+ UTF-8+ Context: <no context>

- 프로젝트 >> app >> build >> outputs >> aar >> app-release.aar 파일이 생성됨을 확인.

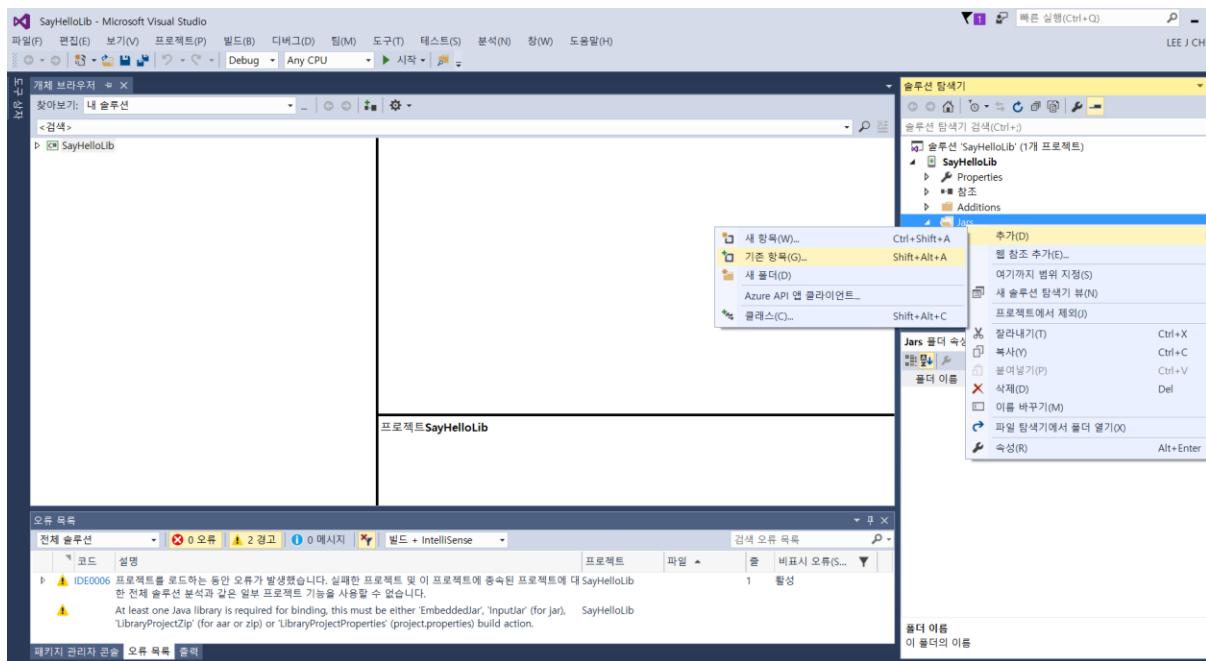


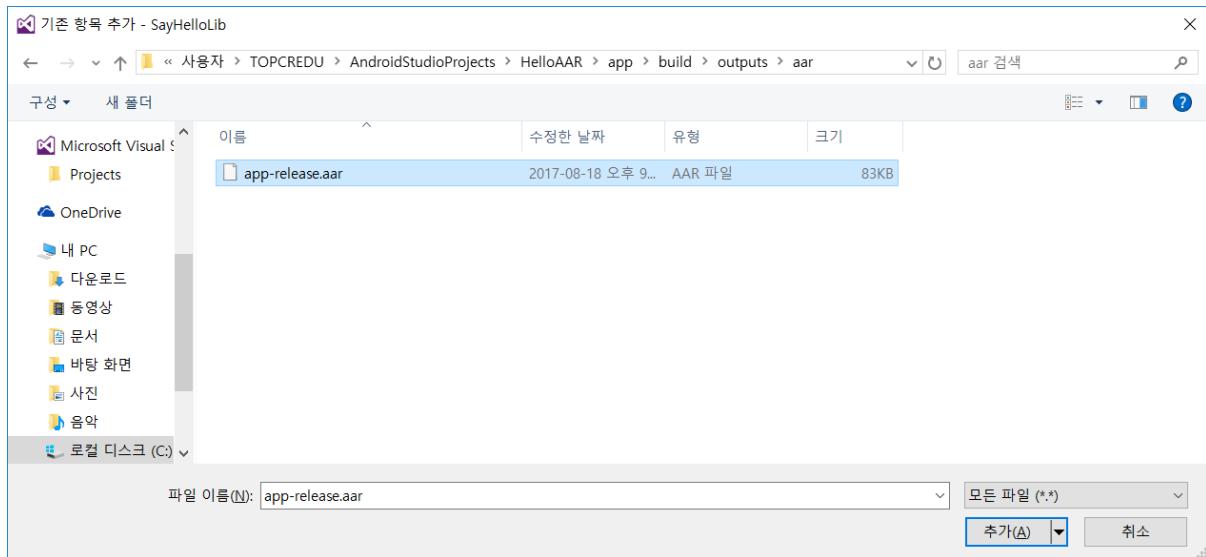
2. 자마린쪽 바인딩 라이브러리 만들기

- Visual Studio에서 Android >> 바인딩 라이브러리(Android)를 선택하여 "SayHelloLib" 이라는 이름의 바인딩 라이브러리 프로젝트를 생성하자.

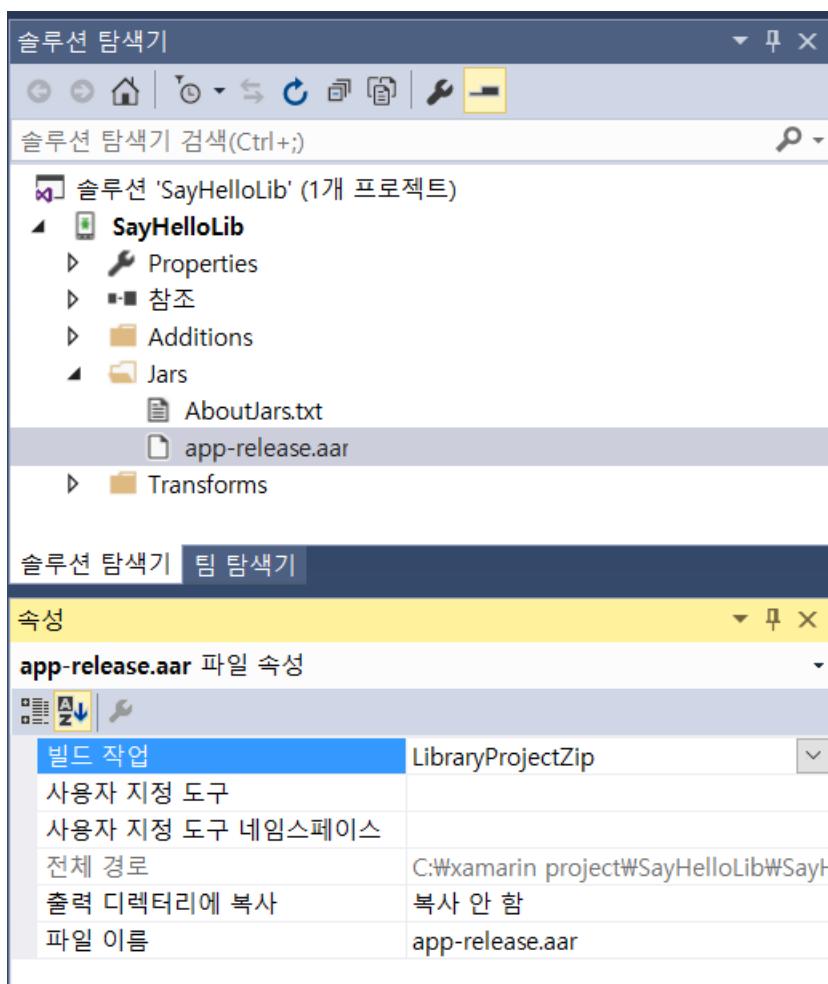


- 생성된 프로젝트에서 “Jars” 폴더에서 우측마우스 클릭 >> 추가 >> 기존항목 추가 선택 후 이전에 만든 .AAR 파일을 추가한다.



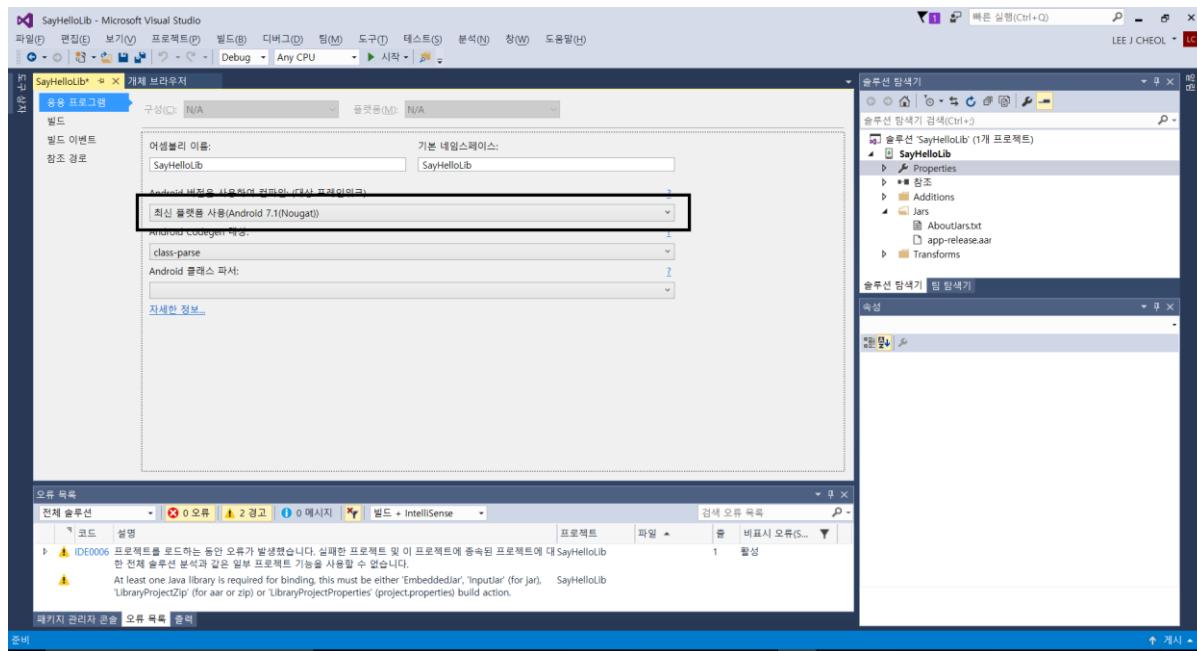


- 추가된 app-release.aar 파일을 선택 후 하단의 속성창의 빌드 작업(Build Action) 속성에서 "LibraryProjectZip"을 선택. (.AAR파일용)

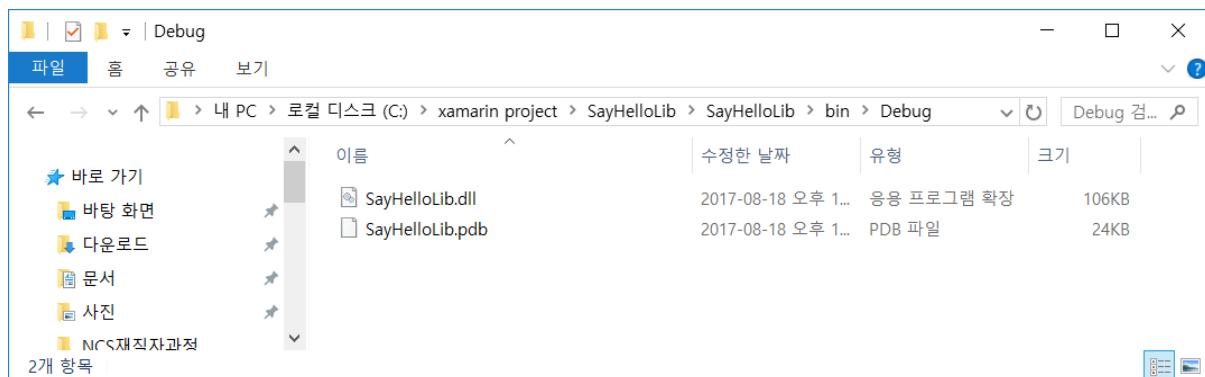


- 만약 .AAR 파일이 안드로이드 API를 사용하는 경우 적합한 Android API 레벨을 선택해야 하

며 본 예제에서는 Android API를 사용하지 않았으므로 마음대로 최신 API 레벨을 사용할 수 있다.

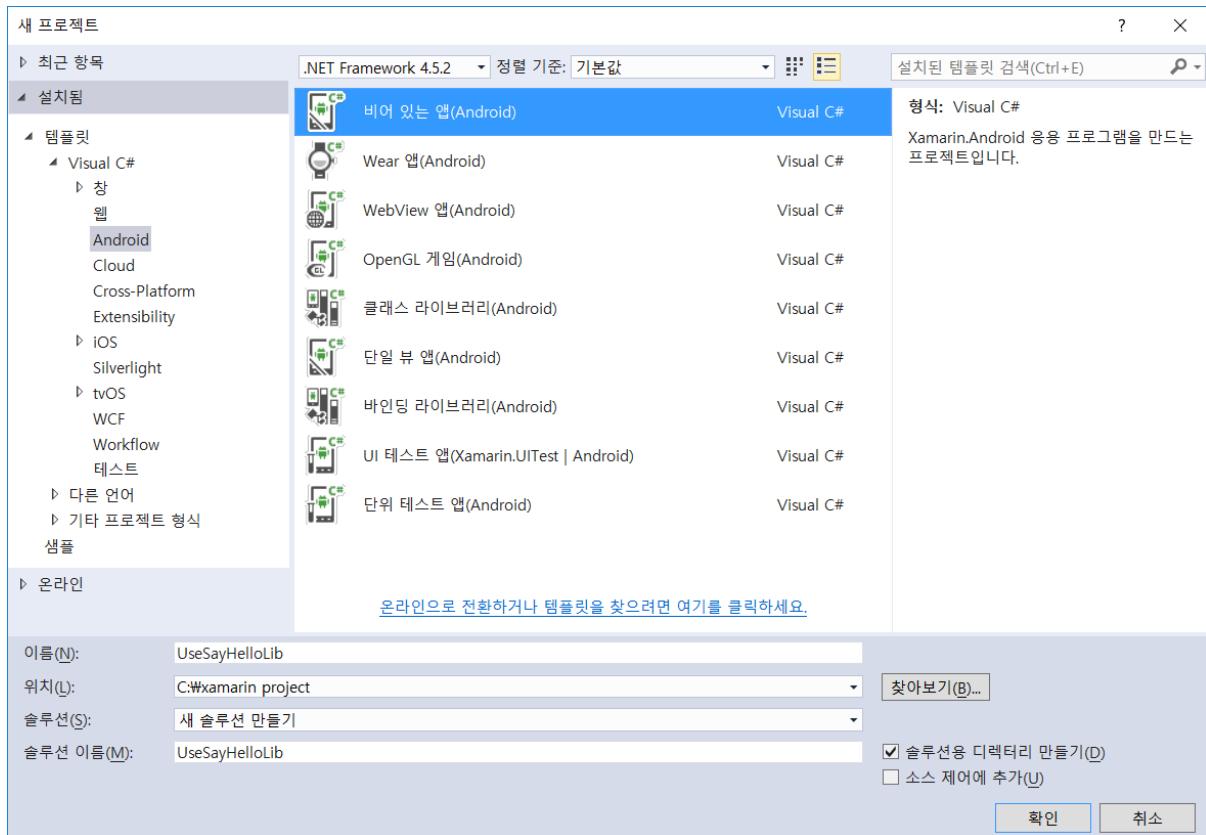


- 프로젝트를 빌드하면 DLL이 생성된다. (이 DLL을 Xamarin.Android 프로젝트에서 참조 추가해 SayHelloImpl의 sayHello() 메소드를 호출한다.)

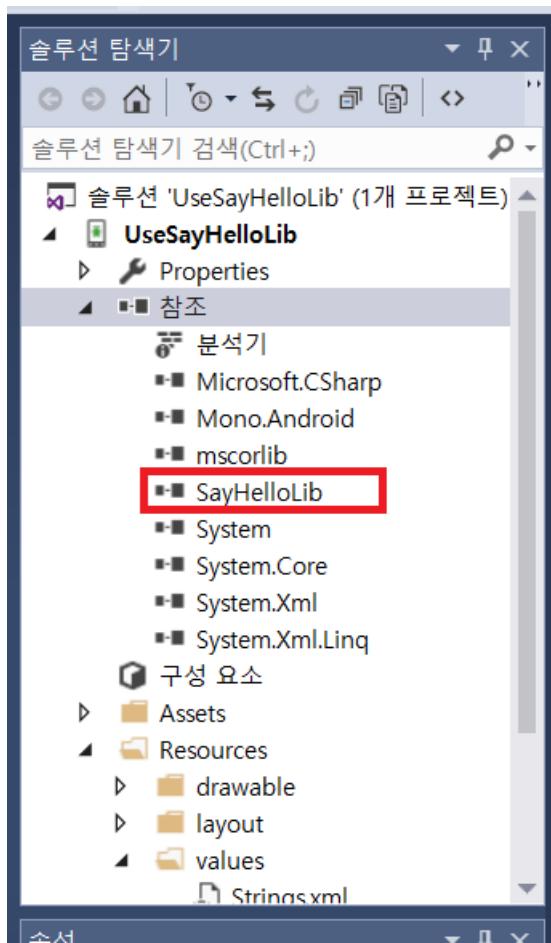


3. Xamarin.Android 응용프로젝트를 생성 후 DLL에 포함된 .AAR파일의 sayHello() 메소드를 호출하자.

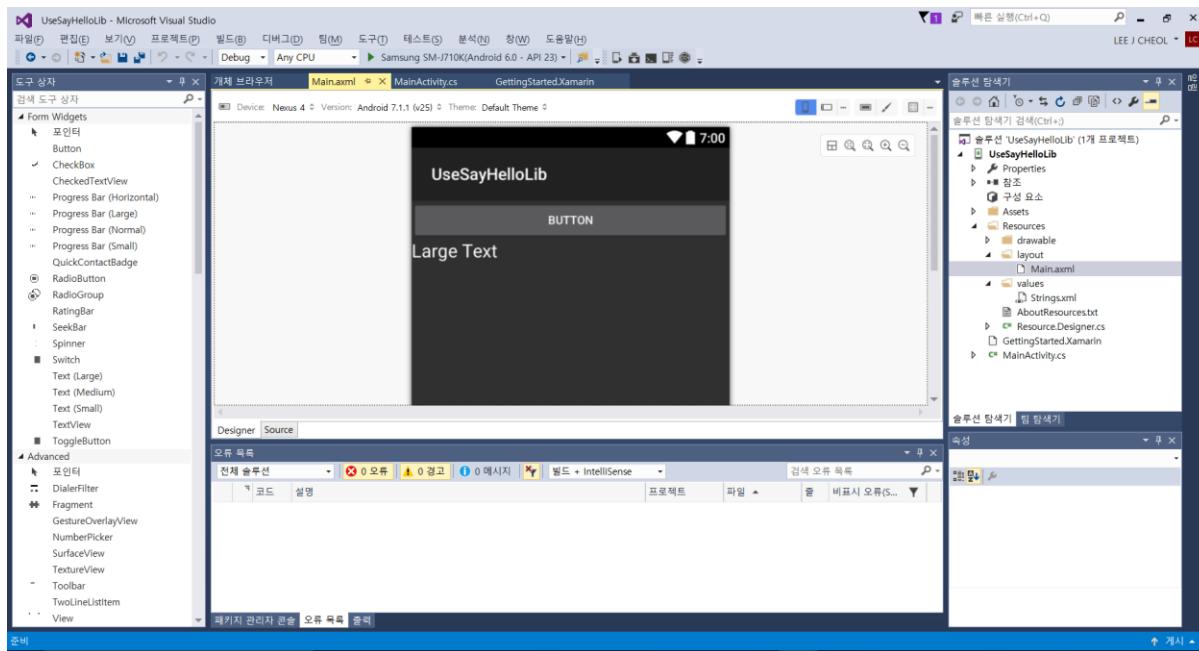
- “**UseSayHelloLib**” 이라는 이름의 Xamarin Android, 비어있는 앱(Android) 응용 프로젝트를 생성하자.



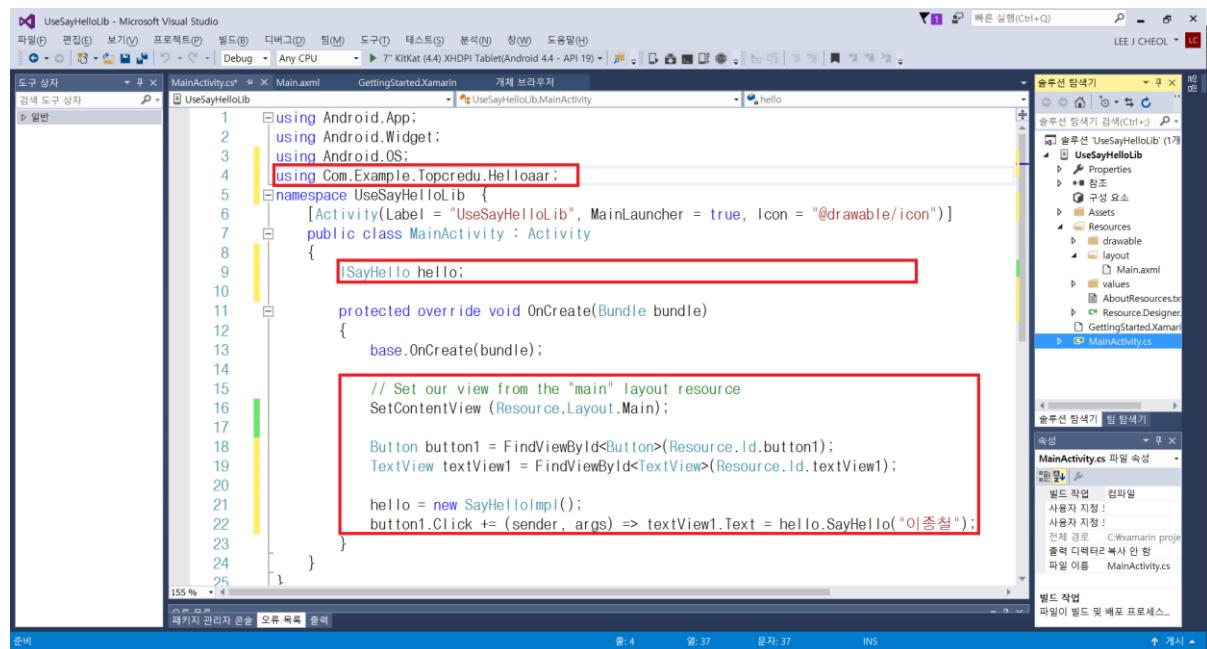
- 참조 >> 참조추가에서 2번 단계에서 작성한 “**SayHelloLib.dll**” 파일을 참조추가 하자.
(SayHelloLib >> bin >> debug)



- **Resources >> layout >> activity_main** 파일을 열어 간단히 UI를 구성하자. Button, Text(Large) 두 컨트롤을 드래그 해서 올리고 이름은 기본값인 button1, textView1로 그대로 두자.



- **MainActivity**에서 Button의 클릭 이벤트를 작성하고 sayHello() 메소드를 호출해 보자.
- 2번 단계의 바인딩 라이브러리인 DLL이 만들어 질 때 원래 .AAR 파일에서 만든 자바 인터페이스 이름의 앞에는 "I"가 붙어 ISayHello로 변하고, 자바의 패키지명의 첫글자가, sayHello() 메소드명도 SayHello()로 첫글자가 대문자로 래핑된다.



```

using Android.App;
using Android.Widget;
using Android.OS;
using Com.Example.Topcredu.Helloaar;

namespace UseSayHelloLib
{
    [Activity(Label = "UseSayHelloLib", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        ISayHello hello;

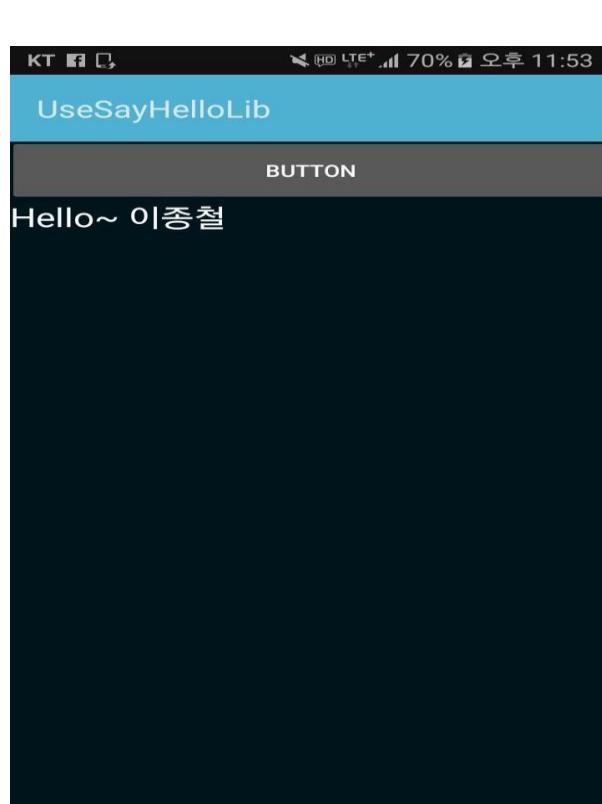
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.Main);

            Button button1 = FindViewById<Button>(Resource.Id.button1);
            TextView textView1 = FindViewById<TextView>(Resource.Id.textView1);

            hello = new SayHelloImpl();
            button1.Click += (sender, args) => textView1.Text = hello.SayHello("이종철");
        }
    }
}

```



■ 실행화면

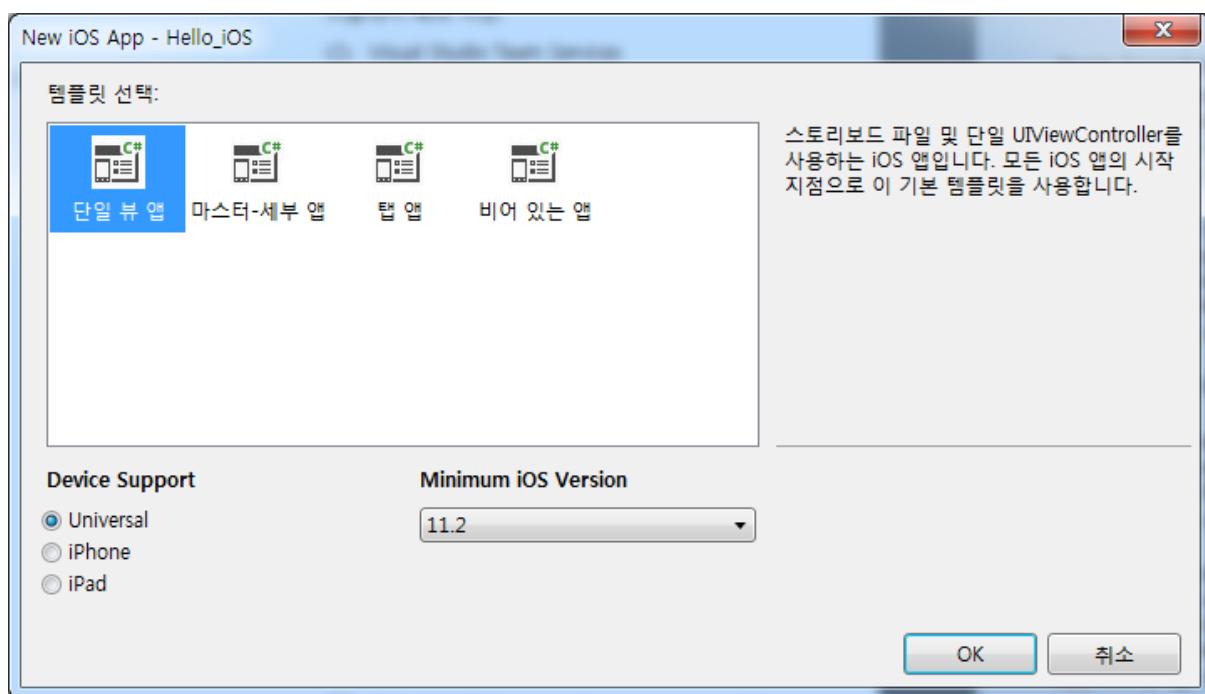
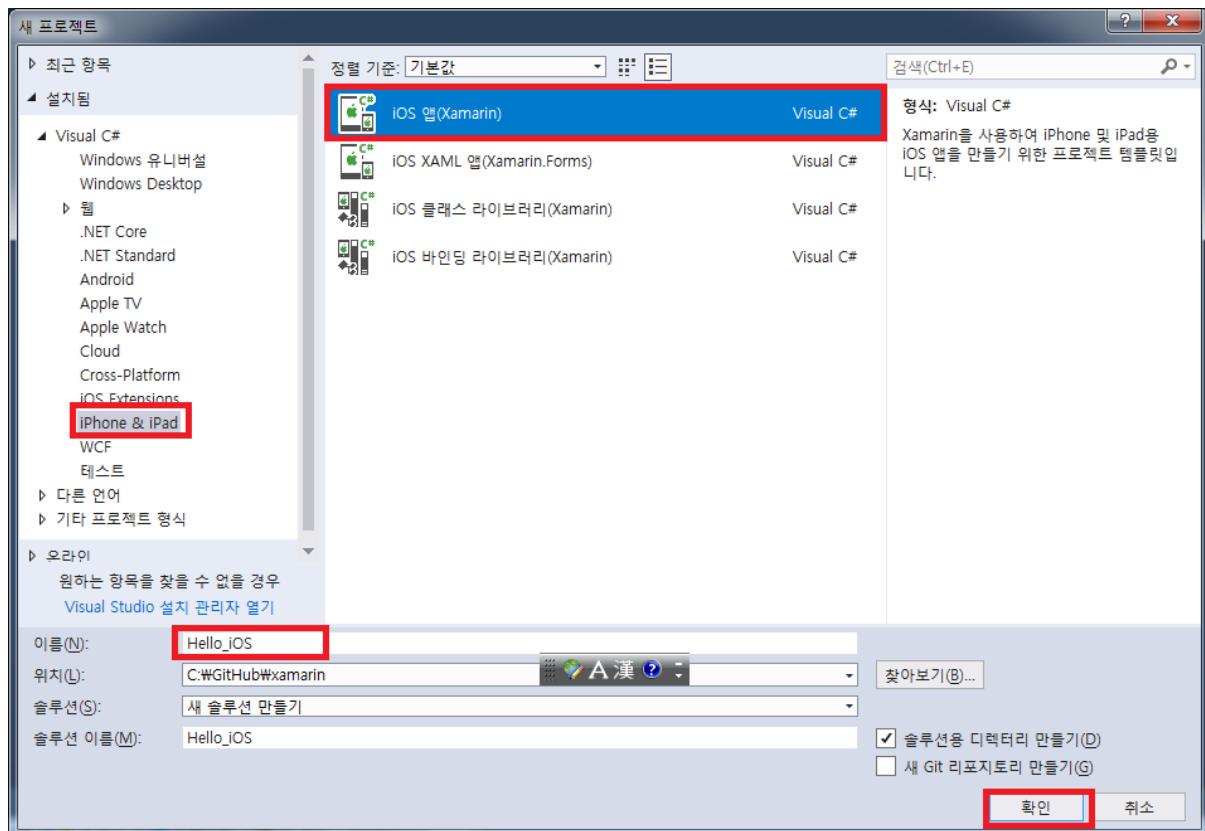
3. Xamarin.iOS

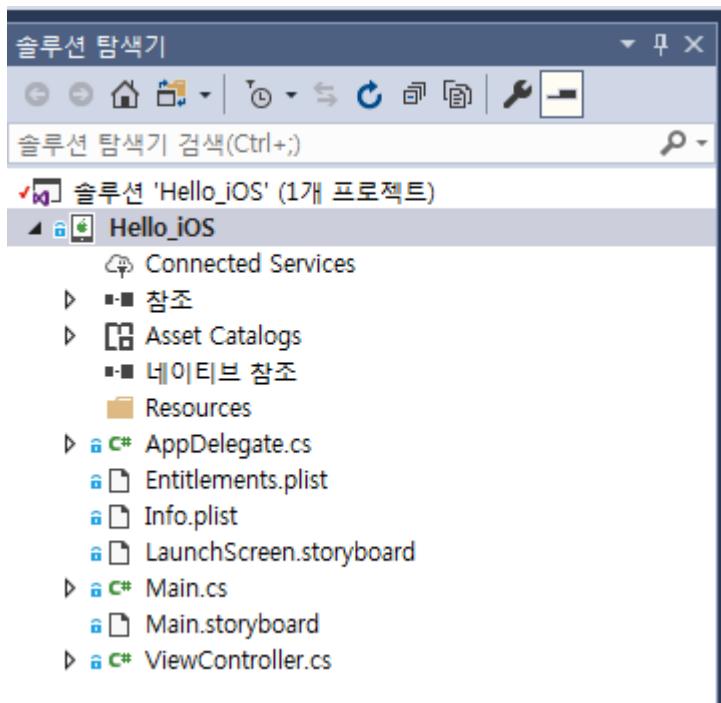
3.1 Xamarin.iOS 설치, 개발환경

- Visual Studio Community 2015 이상에서 Xamarin을 사용한 iOS 개발에는 다음이 필요하다.
 - Mac OS X Sierra(10.12) 이상을 실행하는 Mac 장비
 - App Store에서 설치된 Xcode 및 iOS SDK의 최신 버전.
- 위 개발환경과 Mac의 Xamarin Studio 최신버전 및 윈도우의 Visual Studio에서 동작하며 Windows의 Xamarin Studio는 Xamarin.iOS를 사용할 수 없다.
- Xamarin.iOS 응용 프로그램 개발은 시뮬레이터 외에도 실제 장치에 응용 프로그램을 배포하여 테스트하는 것이 필수적이다. 장치 전용 버그 및 성능 문제는 메모리 또는 네트워크 연결과 같은 하드웨어 제한으로 인해 장치에서 실행될 때 발생할 수 있다.
- 물리적 장치에서 테스트하려면 장치를 준비해야하며 테스트를 위해 장치를 사용할 것이라는 사실을 Apple에 알려야 하는데 응용 프로그램을 장치에 배포하기 전에 Apple의 개발자 프로그램에 가입하거나 무료 제공을 사용해야 하며 Apple은 두 가지 프로그램 옵션을 제공한다.
- Apple 개발자 프로그램 - 개인이든 단체이든 상관없이 Apple 개발자 프로그램을 사용하면 응용 프로그램을 개발, 테스트 및 배포 할 수 있다.
- Apple Developer Enterprise Program - Enterprise 프로그램은 사내에서만 앱을 개발하고 배포하려는 조직에 가장 적합하다. 엔터프라이즈 프로그램의 회원은 iTunes Connect에 액세스 할 수 없으며 생성 된 응용 프로그램을 App Store에 게시 할 수 없다.
- Apple Developer Portal을 방문하여 등록해야 하며 Apple 개발자로 등록하려면 Apple ID가 있어야 한다.

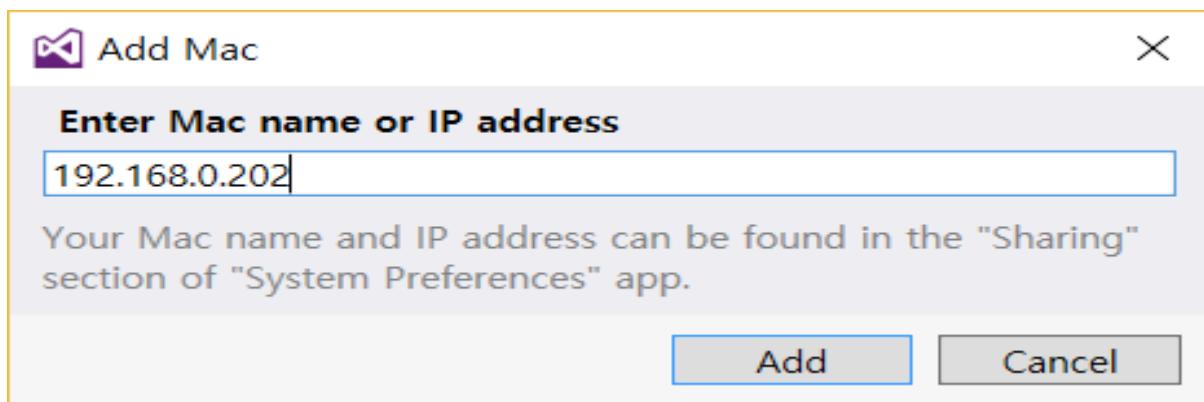
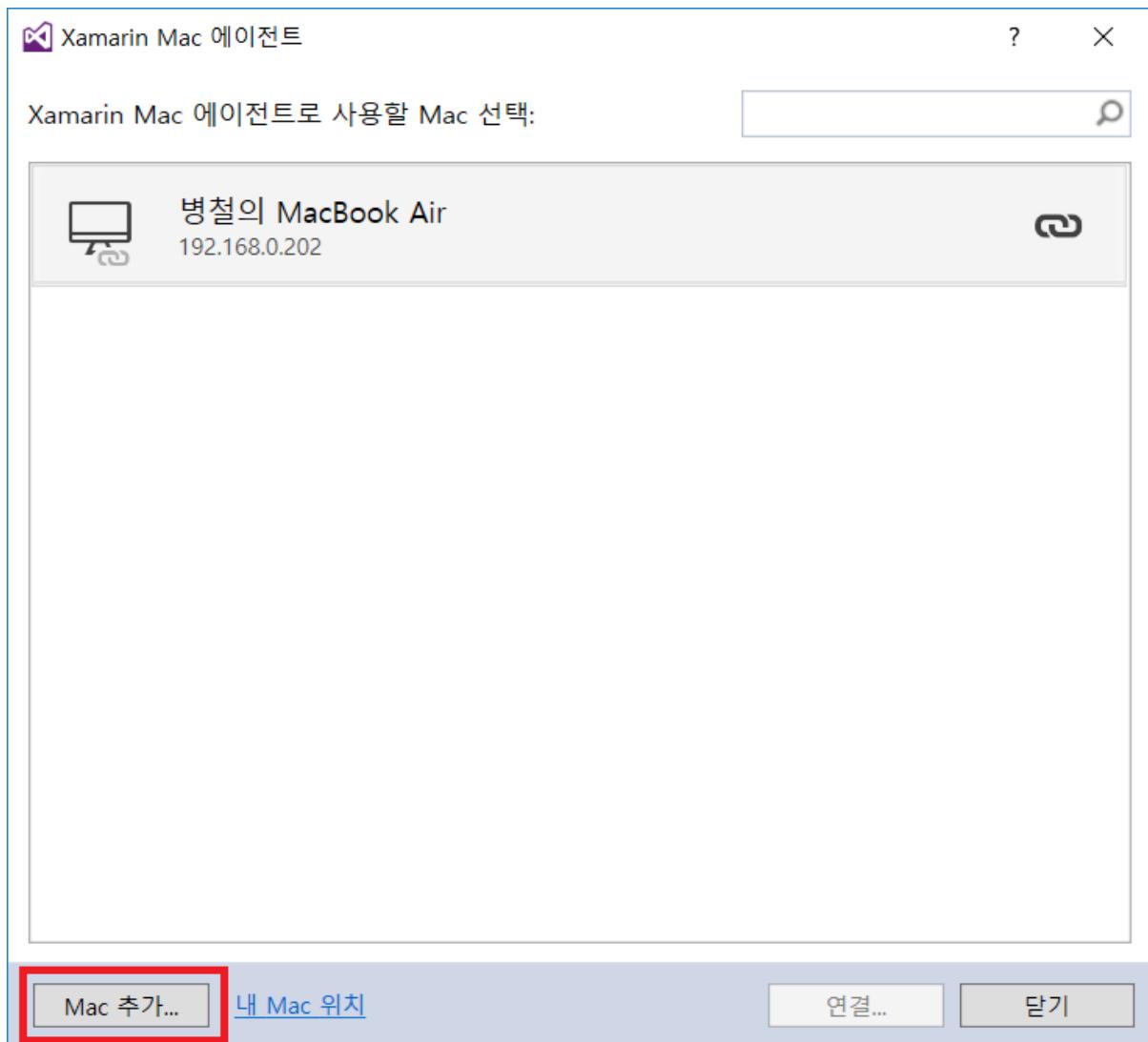
3.2 Xamarin.iOS HelloWorld(단일 뷰) 실습

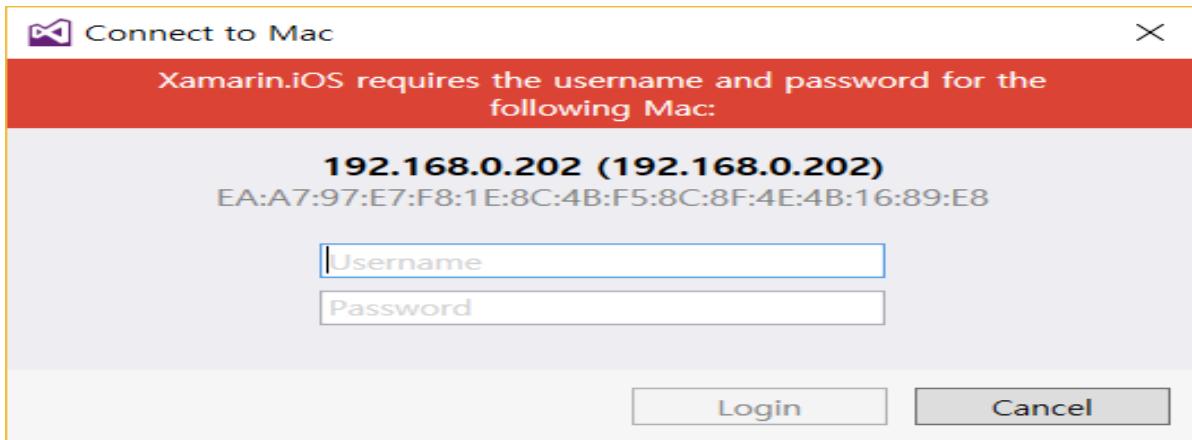
- Xamarin.Android HelloWorld 예제를 Xamarin.iOS 기반으로 재작성 해보자.
- **비주얼 스튜디오를 실행하여 File -> New Project 후 좌측 Template에서 iphone&ipad -> iOs앱(xamarin)을 선택 후 프로젝트명 기술하고 확인, 다음 화면에서 “단일뷰앱”을 선택 후 OK 버튼을 클릭한다.**





- 비주얼스튜디오 -> 도구 -> iOS -> Xamarin Mac Agent(M)를 클릭하여 Mac 장비를 등록해야 한다. Mac 장비의 IP주소, 로그인 계정, 비밀번호를 입력해서 Mac에 로그인 해야 한다. (Mac 쪽에 사용자 생성되어 있어야 함)

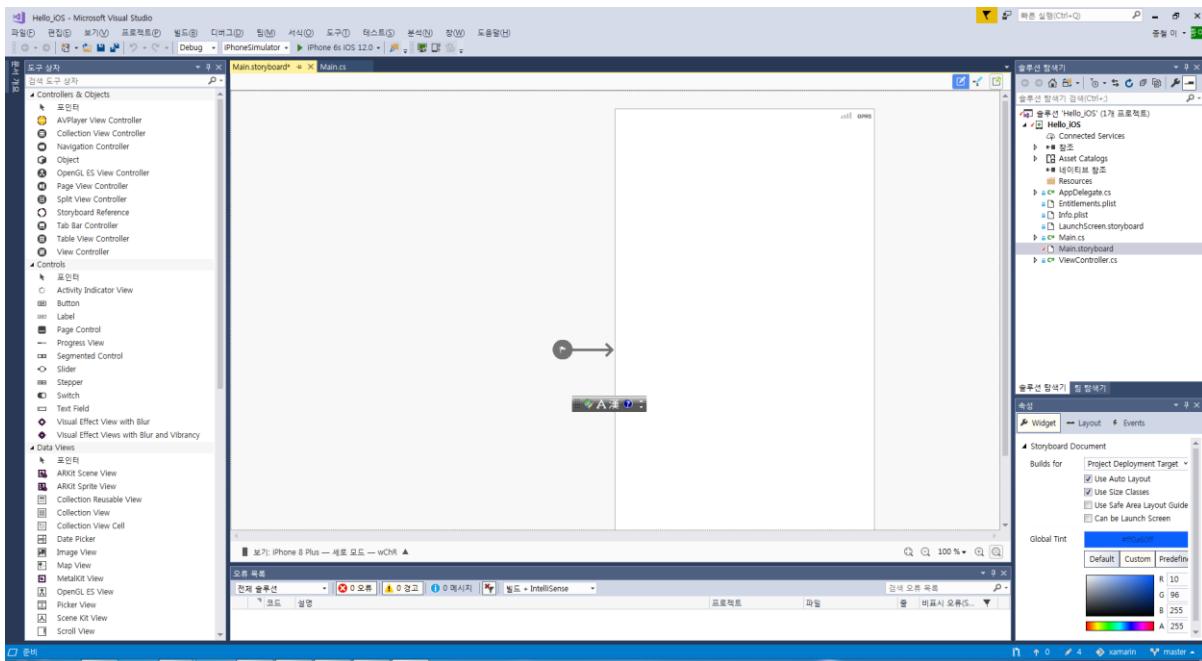




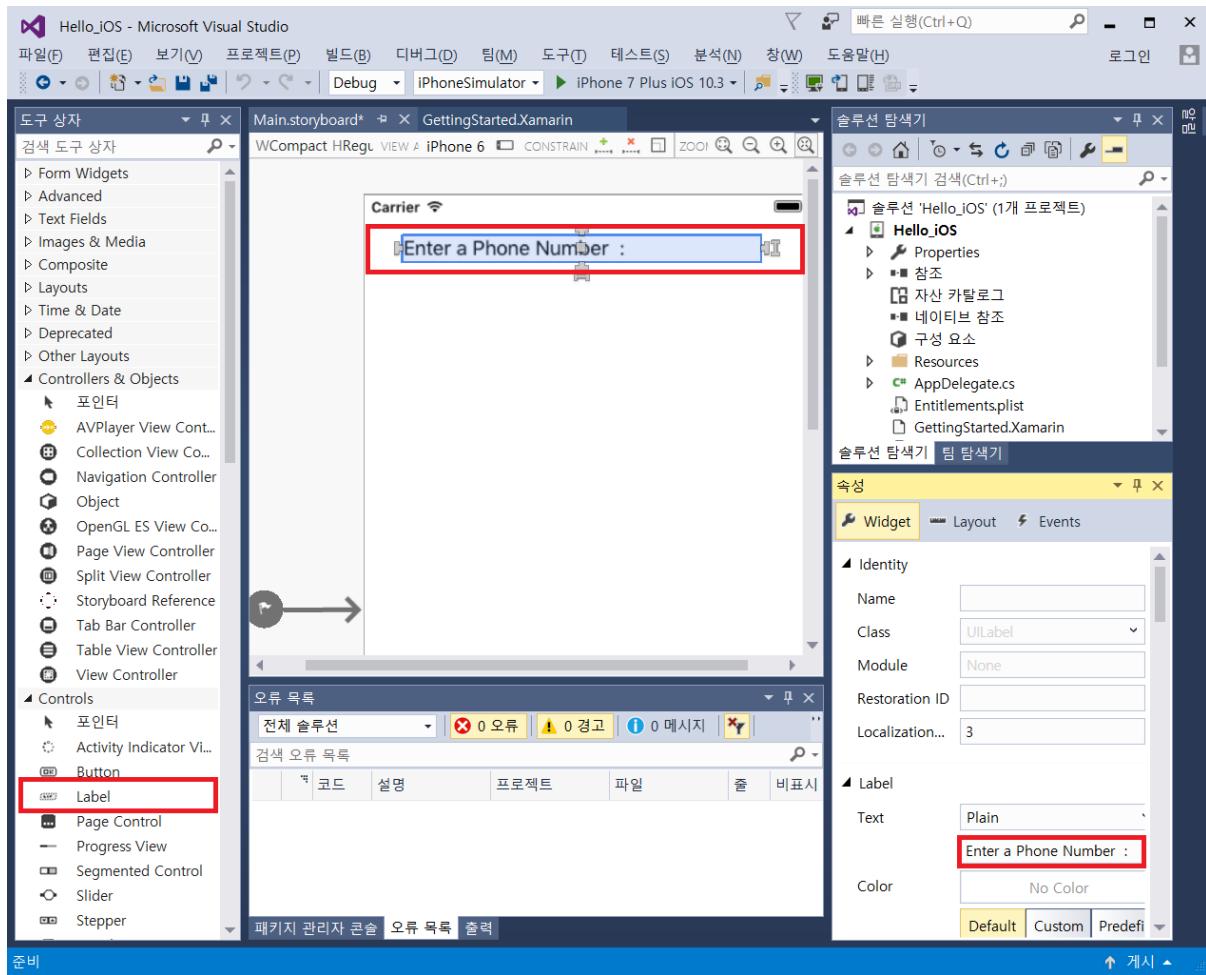
- Mac에 연결이 되면 상단 “Xamain Mac 에이전트” 버튼이 아래처럼 Green 색으로 표시된다.



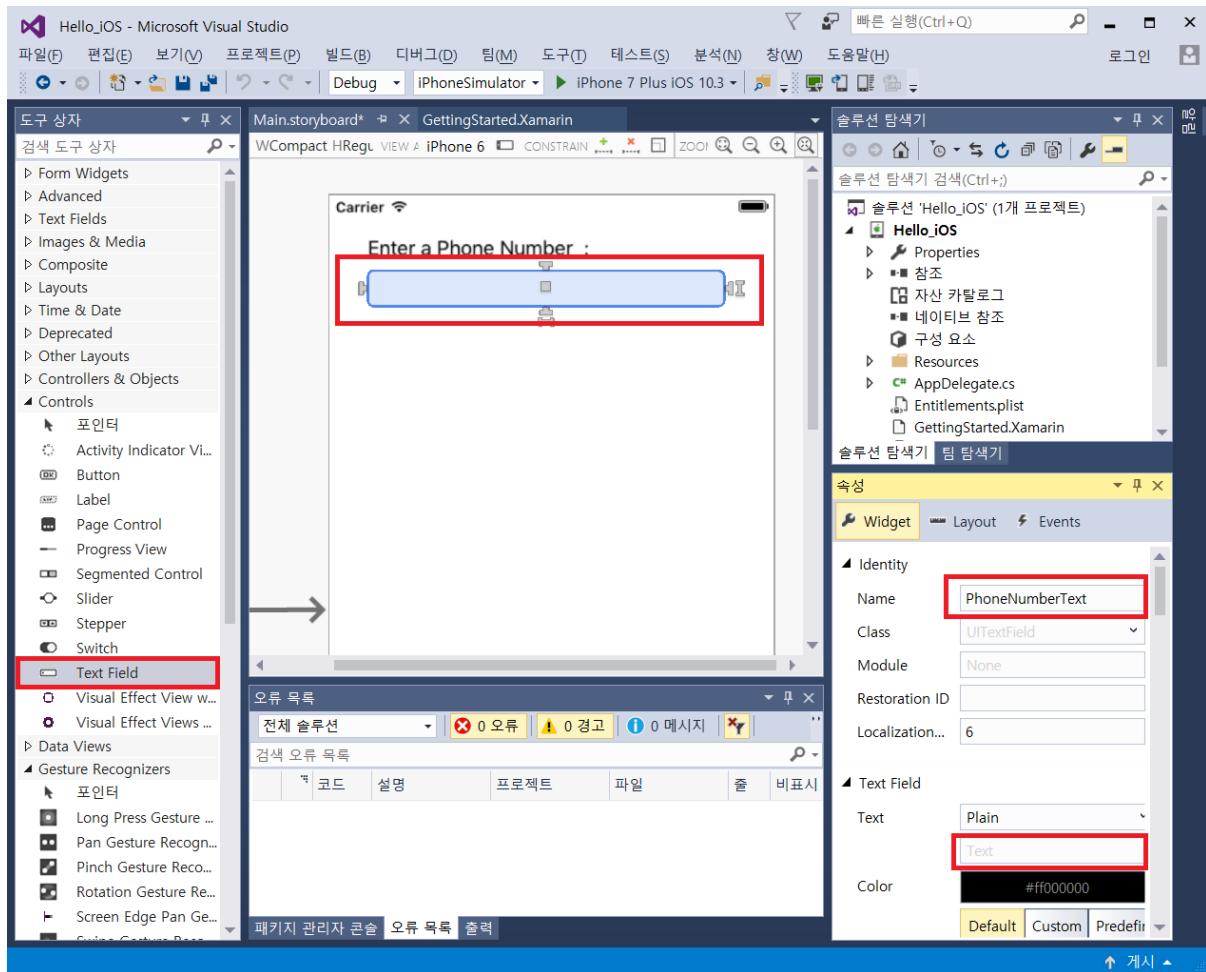
- 솔루션 탐색기에서 Main.storyboard 클릭 후 View AS iPhone6 선택한다. Main.storyboard는 사용자 인터페이스의 비주얼적인 부분을 디자인 하며 iOS Designer라고 하는 Graphical Editor를 포함하고 있다.



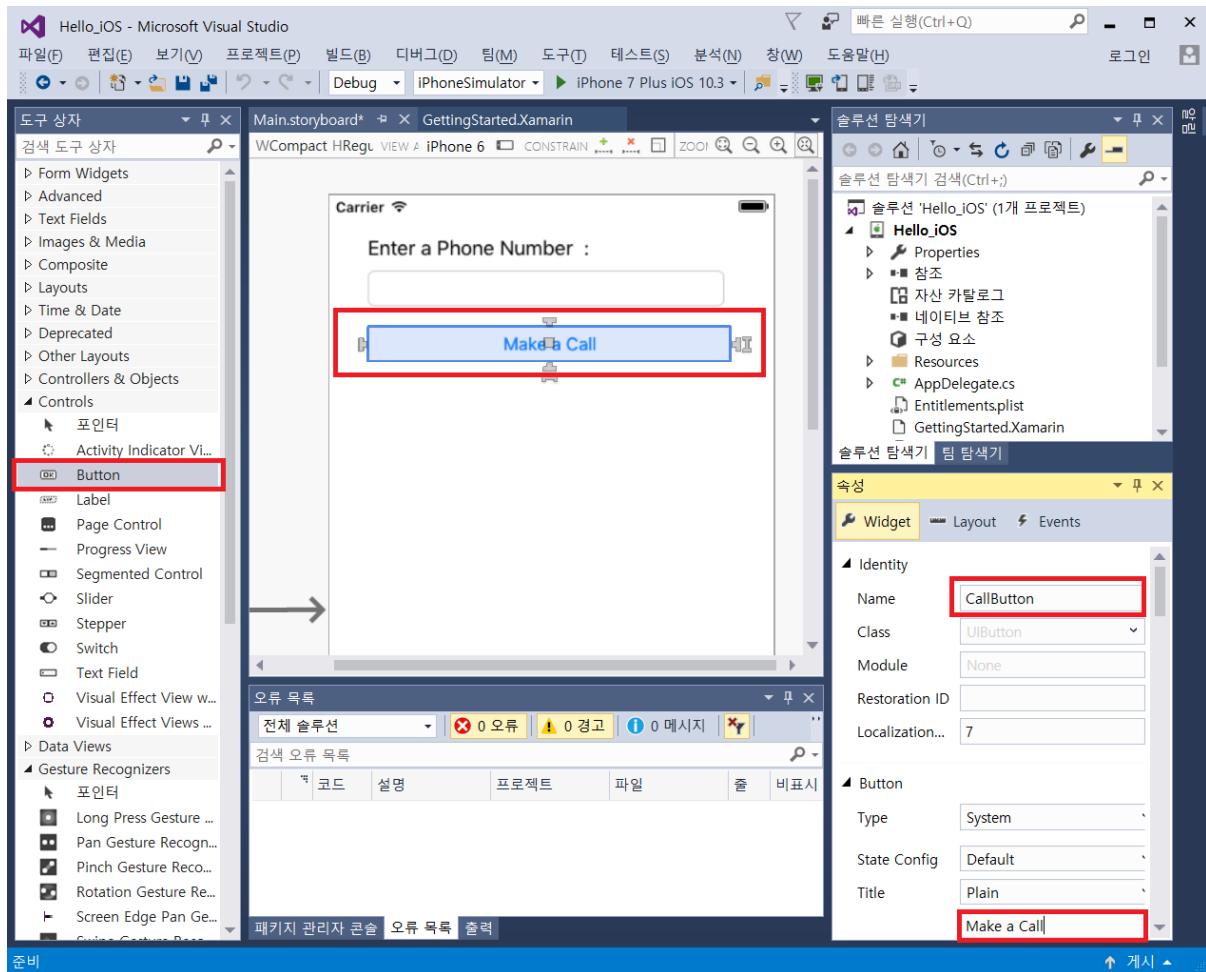
- 보기 -> 도구상자에서 Controls -> Label을 선택해서 디자인 화면 중앙에 위치 시킨 후 크기 를 화면처럼 늘이고 Text 속성에 "Enter a Phone Number :"라고 입력하자.



- 도구상자에서 Controls -> Text Field를 선택해서 Label 아래에 위치 시킨 후 크기를 적당히 조절하고 우측하단 속성창의 Name속성을 “**PhoneNumberText**”, Text 속성을 “”로 설정하자.



- 도구상자에서 Controls -> Button을 선택해서 Text Field 아래에 위치 시킨 후 크기를 적당히 조절하고 속성 창에서 Identity -> Name속성을 “**CallButton**”, Title 속성을 “**Make a Call**”로 설정하자.



- **ViewController.cs** 파일의 **ViewDidLoad()** 메소드 뒤 부분에 전화걸기버튼(CallButton)의 기능 추가 (상단에 using Foundation; 구문 추가)

View Controller는 화면으로부터 Content View Hierarchy의 요소들을 로딩하거나 언로딩 한다. Content View Hierarchy안의 View에서 발생되는 중요한 일에 대해 View의 라이프 사이클 동안 OS는 이벤트를 통해 View Controller에 알린다.

ViewDidLoad : View Controller가 메모리에 Content View Hierarchy를 로드하는 시점에 한번 호출되며 Xamarin.Android의 OnCreate 콜백함수와 같은 기능을 한다.

```

public override void ViewDidLoad()
{
    base.ViewDidLoad();

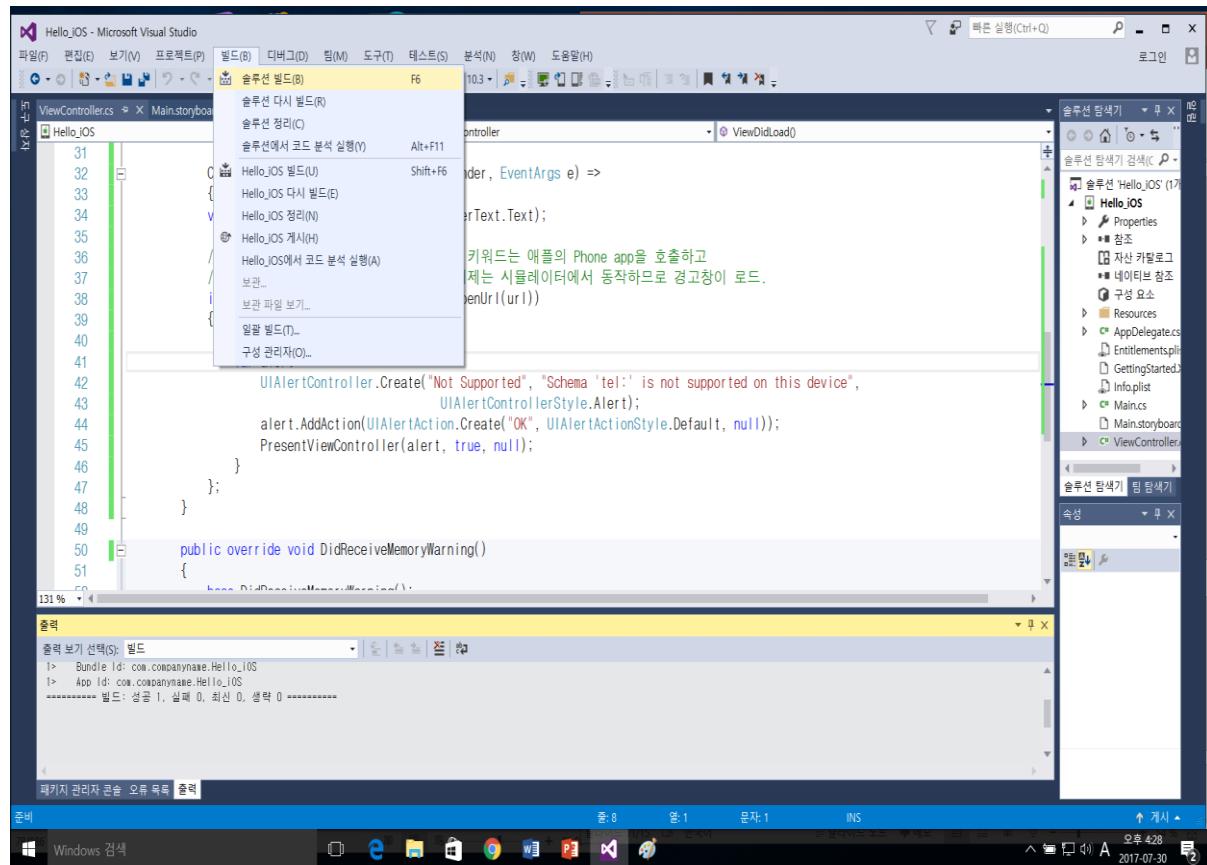
    // 전화번호 입력창에 포커싱이 된 경우 키보드를 화면에서 사라지게 하기위해
    PhoneNumberText.ResignFirstResponder();

    CallButton.TouchUpInside += (object sender, EventArgs e) =>
    {
        var url = new NSUrl("tel:" + PhoneNumberText.Text);

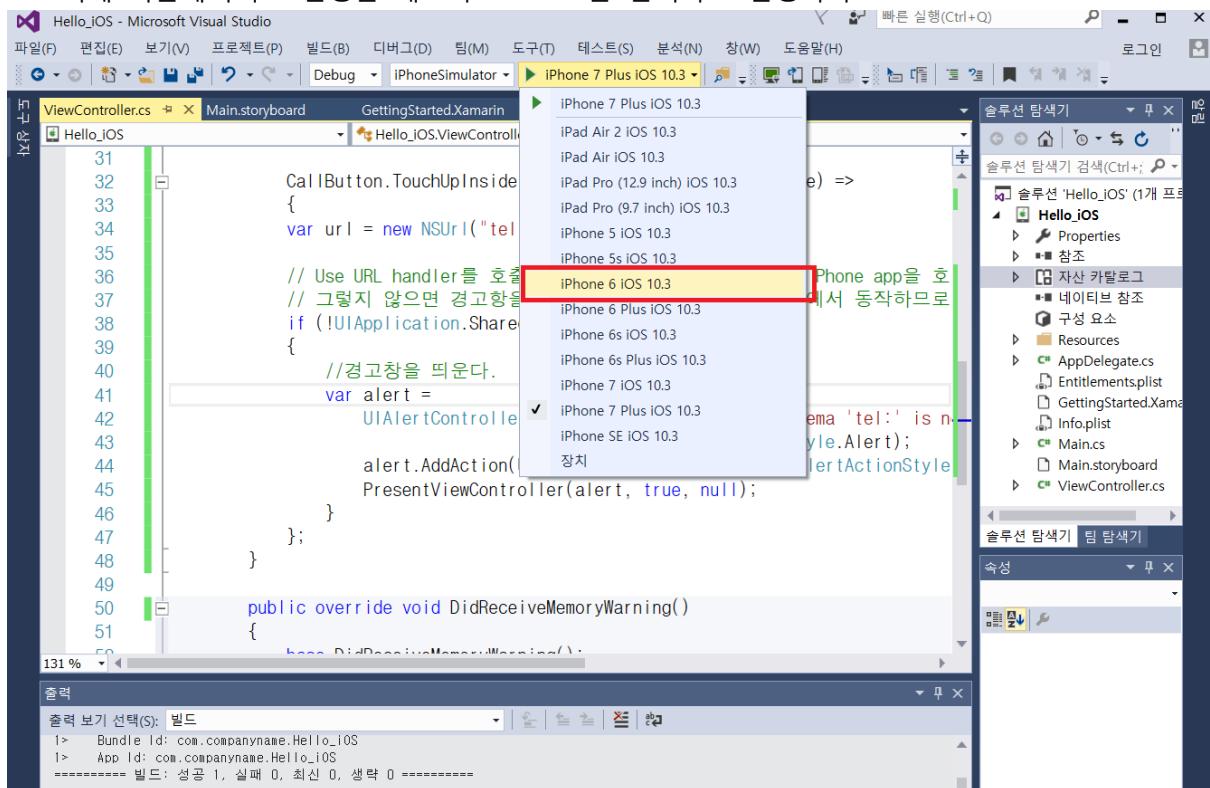
        // Use URL handler를 호출하는데 "tel:" 키워드는 애플의 Phone app을 호출하고
        // 그렇지 않으면 경고창을 띄운다. 본 예제는 시뮬레이터에서 동작하므로 경고창이 로드.
        if (!UIApplication.SharedApplication.OpenUrl(url))
        {
            //경고창을 띄운다.
            var alert =
                UIAlertController.Create("Not Supported", "Schema 'tel:' is not supported on this device",
                    UIAlertControllerStyle.Alert);
            alert.AddAction(UIAlertAction.Create("OK", UIAlertActionStyle.Default, null));
            PresentViewController(alert, true, null);
        }
    };
}

```

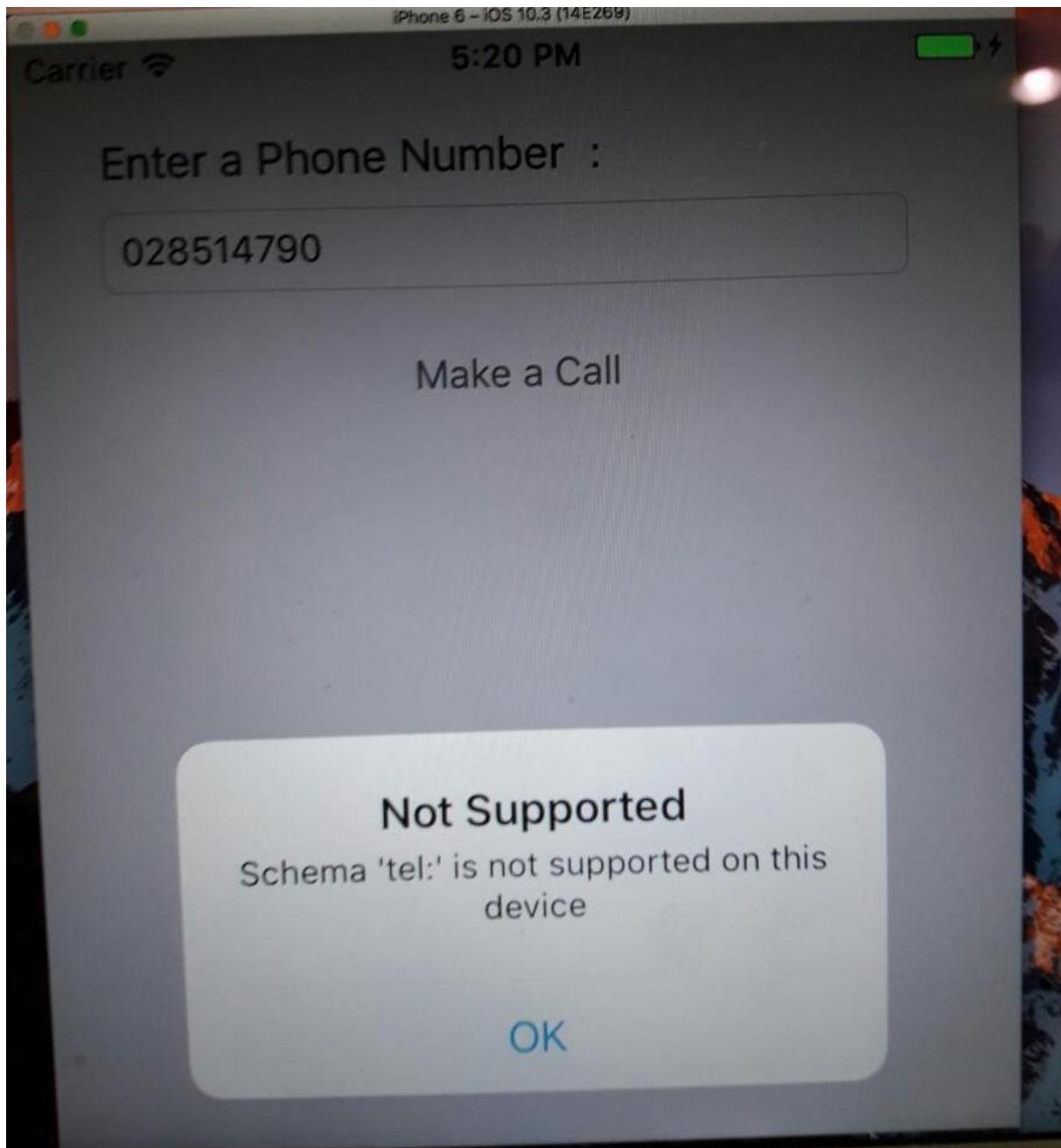
- 코드 작성이 마무리 되었으니 도구 -> 빌드 -> 솔루션 빌드를 클릭해서 빌드를 해보자. 아래 이미지 처럼 오류가 없어야 한다.



■ 이제 시뮬레이터로 실행을 해보자. iPhone6을 선택하고 실행하자.



- 시뮬레이터 실행화면(자마린이 버전업 되면서 원격 맥북에서만 시뮬레이터로 볼 수 있었지만 Xamarin.iOS Simulator를 설치하면 로컬 화면에서 결과화면을 시뮬레이터로 볼 수 있다.)

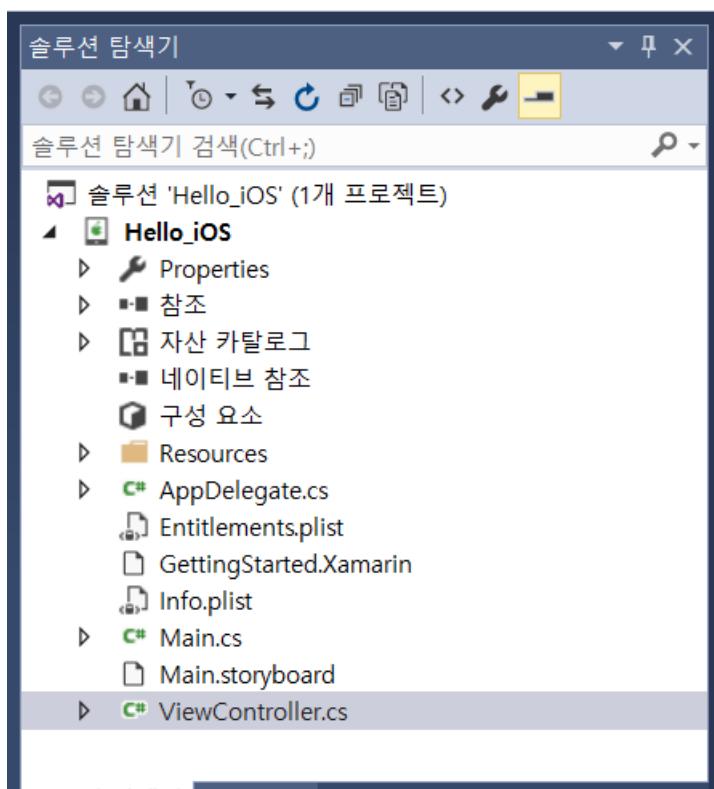


3.3 Xamarin.iOS HelloWorld 자세히 살펴보기

- 비주얼 스튜디오는 Solution과 프로젝트들로 구성되며 하나의 솔루션에 하나 이상의 프로젝트를 포함할 수 있다. 하나의 프로젝트는 iOS, Android Application이 될 수 있으며 라이브러리를 지원하며 테스트가 가능하다.
- 본 Xamarin.iOS 예제는 **단일 뷰 앱**(Single View Application) 템플릿을 이용하여 만들었다.

3.3.1 Xamarin.iOS HelloWorld 해부하기

- Solution Pane의 전체 구조



- **참조(References)** : 빌드하거나 실행할 때 필요한 참조하는 어셈블리들을 포함하고 있다. Xamarin.iOS, System, System.Xml등
- **구성요소(Components)** : 미리 만들어진 Xamarin Components Store이며 자마린 코드를 위한 Public 마켓이다.
- **Resources** : 아이콘, 실행 이미지, 다른 미디어들을 포함한다.
- **Main.cs** : 응용프로그램의 진입점(Entry Point). 응용프로그램의 시작을 위해서는 메인 응용프로그램의 이름(AppDelegate)을 던져 줘야한다.

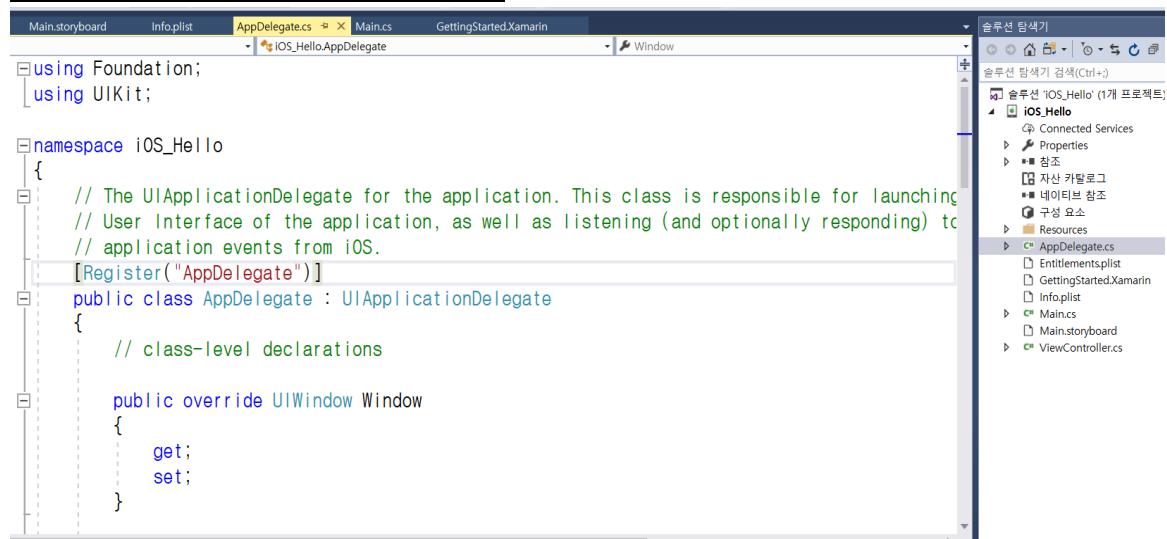
```
namespace Phoneword.iOS
{
    public class Application
    {
        // This is the main entry point of the application.
        static void Main(string[] args)
        {
            // if you want to use a different Application Delegate class from "AppDelegate"
            // you can specify it here.
        }
    }
}
```

```

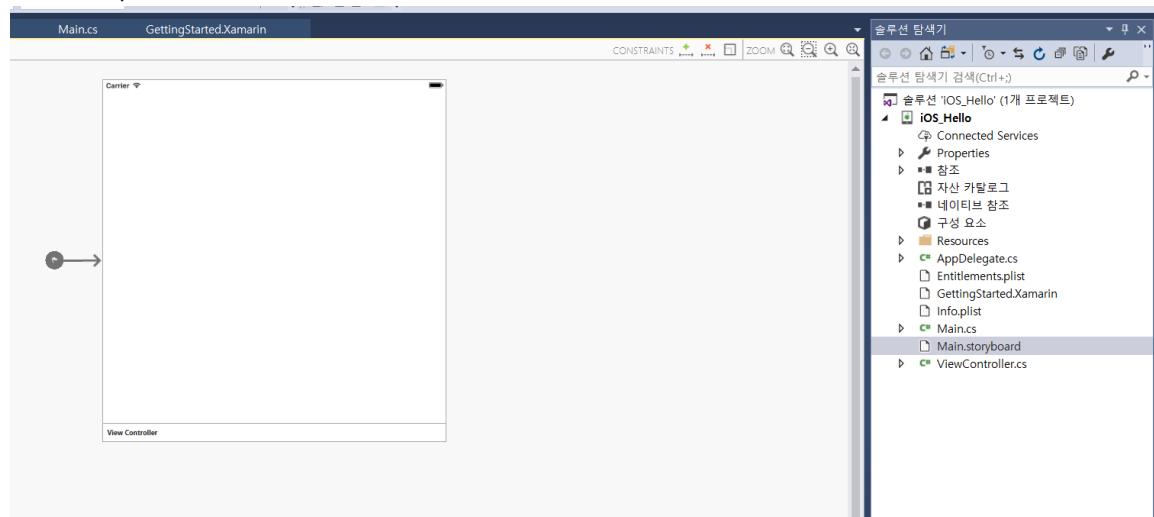
        UIApplication.Main(args, null, "AppDelegate");
    }
}
}

```

- **AppDelegate.cs** : 메인 응용프로그램 클래스를 포함하며 iOS 응용 프로그램에 하나 있는 Window를 생성하고 사용자 인터페이스의 런칭을 책임진다. 사용자 인터페이스의 빌드, iOS로 부터의 시스템 이벤트를 리스닝 하며 중요한 응용프로그램의 이벤트(실행종료, 메모리 부족 등)에 관한 시스템 업데이트를 관리한다.



- **Main.storyboard** : 사용자 인터페이스의 비주얼적인 부분을 디자인 한다. iOS Designer라고 하는 Graphical Editor를 포함하고 있다.



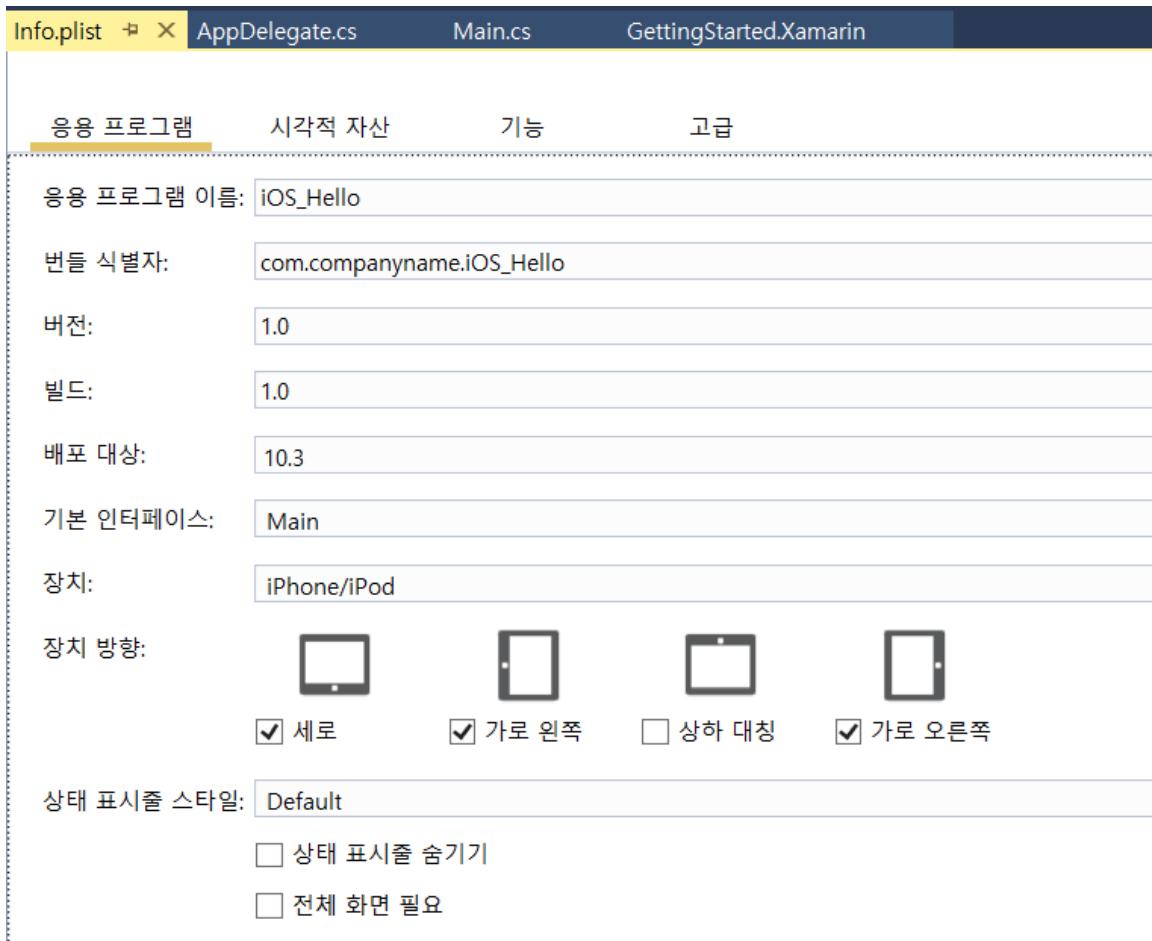
- **ViewController.cs** : 사용자가 보거나 터치하는 스크린(뷰)을 통제하며 사용자와 뷰의 상호 작용을 처리한다.

```

1  using System;
2  using UIKit;
3
4  namespace iOS_Hello
5  {
6      public partial class ViewController : UIViewController
7      {
8          public ViewController(IntPtr handle) : base(handle)
9          {
10         }
11
12         public override void ViewDidLoad()
13         {
14             base.ViewDidLoad();
15             // Perform any additional setup after loading the view, typically from a nib.
16         }
17
18         public override void DidReceiveMemoryWarning()
19     }
}

```

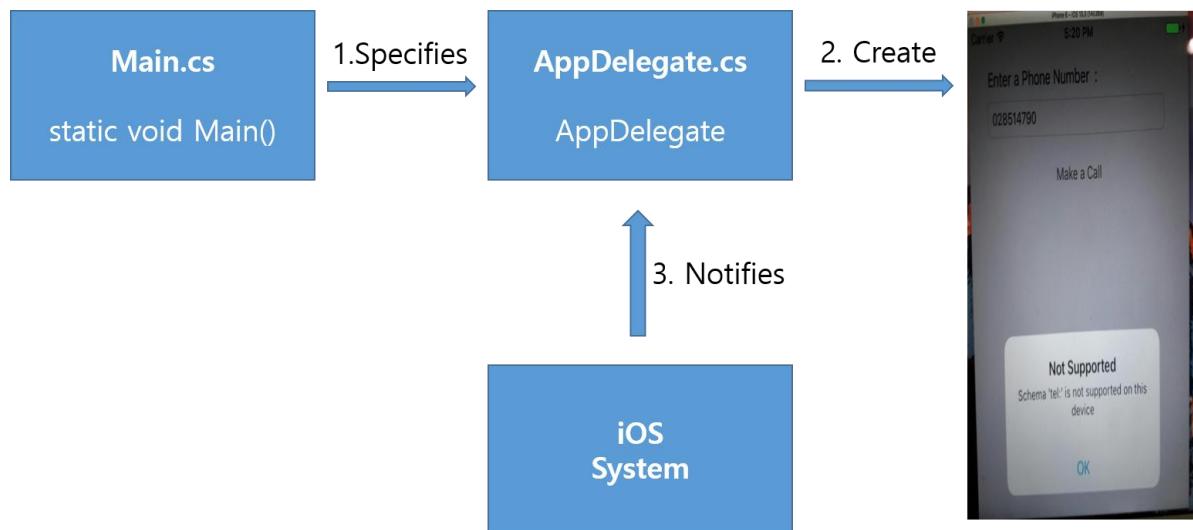
- **ViewController.designer.cs** : 자동 생성되는 파일로 뷰와 뷰컨트롤러 안의 Presentation을 접착제처럼 연결해준다.
- **Info.plist** : 응용프로그램의 이름, 아이콘, 런처이미지와 같은 응용프로그램의 속성을 포함하고 있다.



- **Entitlements.plist** : iCloud, PassKit등과 같은 응용프로그램의 기능(App Store Technologies) 등을 기술하는 파일이다.

3.3.2 Architecture and App Fundamentals

- iOS 응용 프로그램이 사용자 인터페이스로 로딩 되기전에 두가지가 필요한데 하나는 Application Process가 메모리에 로드될 때 시작될 부분인 엔트리 포인트(Main.cs)이고 또 하나는 응용 프로그램을 다루기 위한 전역적인 이벤트와 OS와의 상호작용을 기술한 클래스 (AppDelegate.cs)의 정의이다.



- iOS 응용프로그램의 엔트리 포인트는 static Main 메소드를 포함하는 Main.cs 이다. 이 Main 메소드가 Xamarin.iOS 인스턴스를 생성하고 OS 이벤트를 핸들링하는 Application Delegate 클래스에 전달한다.

```
static void Main (string[] args)
{
    UIApplication.Main (args, null, "AppDelegate");
}
```

- Application Delegate : iOS에서 Application Delegate 클래스는 iOS 시스템 이벤트를 다루며 AppDelegate.cs 파일 안에서 생성된다. 응용프로그램의 원도우를 다루며 사용자 인터페이스를 위한 컨테이너로서의 역할을 하는데 중요한 응용프로그램의 이벤트(실행종료, 메모리 부족 등)에 관한 시스템 업데이트를 관리한다.

```
using Foundation;
using UIKit;

namespace Phoneword.iOS
{
```

```

// 메인 응용프로그램 클래스를 포함하며 윈도우를 만들거나
// 사용자 인터페이스의 빌드, iOS로 부터의 시스템 이벤트를 리스닝 한다.
// AppDelegate 클래스는 응용프로그램 윈도우를 다루며 사용자 인터페이스의 런칭을
책임진다.

// 중요한 응용프로그램의 이벤트(실행종료, 메모리 부족등)에 관한 시스템 업데이트를
관리한다.

[Register("AppDelegate")]
public class AppDelegate : UIApplicationDelegate
{
    // class-level declarations

    public override UIWindow Window
    {
        get;
        set;
    }

    .....
    .....
}

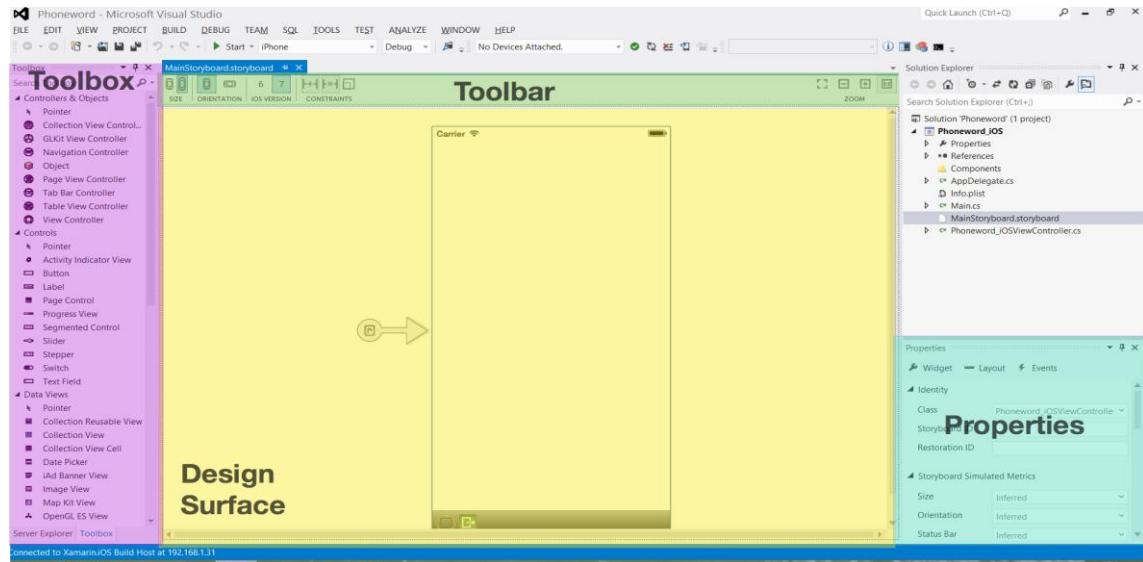
```

3.3.3 User Interface(iOS Designer, Storyboards)

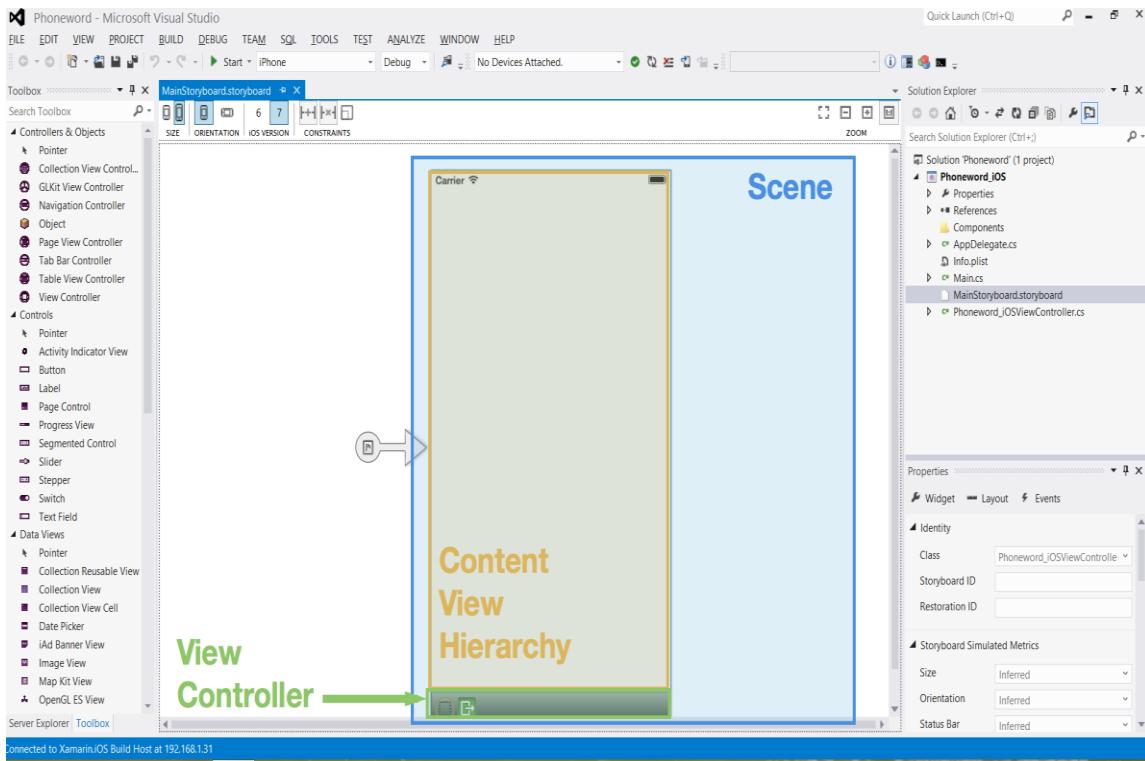
- iOS 앱에서 일반적으로 애플리케이션은 하나의 Window를 가져 오지만 필요한 만큼의 오브젝트로 Window를 채울 수 있고, 앱이 표시 할 항목에 따라 오브젝트와 배열을 변경할 수 있다. 사용자가 보는 것을 뷰(View) 라고 한다.
- 응용 프로그램에서 하나의 화면을 만들려면 View가 Content View Hierarchy 구조에서 서로 위에 겹쳐지고 계층 구조는 View Controller로 관리된다.
- 여러 개의 화면이 있는 응용 프로그램에는 여러 개의 Content View Hierarchy가 있으며 각 Content View Hierarchy에는 별도의 View Controller가 있다.
- 응용 프로그램은 Window에 View를 배치하여 사용자가 있는 화면을 기반으로 다른 Content View Hierarchy를 만든다.
- 앱에 많은 뷰 컨트롤러가 포함될 수 있지만 모든 뷰 컨트롤러를 제어 하려면 하나의 Root View Controller가 있어야 한다.
- iOS 앱에서 단일 화면을 작성하기 위해 뷰는 Content View Hierarchy에서 서로 쌓아 올려지고 계층은 단일 View Controller에서 관리된다. 여러 화면이 있는 응용 프로그램에는 여러 개의 Content View Hierarchy가 있으며, Content View Hierarchy에는 고유 한 View Controller가

있고 응용 프로그램은 Window에 View를 배치하여 사용자가 있는 화면을 기반으로 다른 Content View Hierarchy를 만든다.

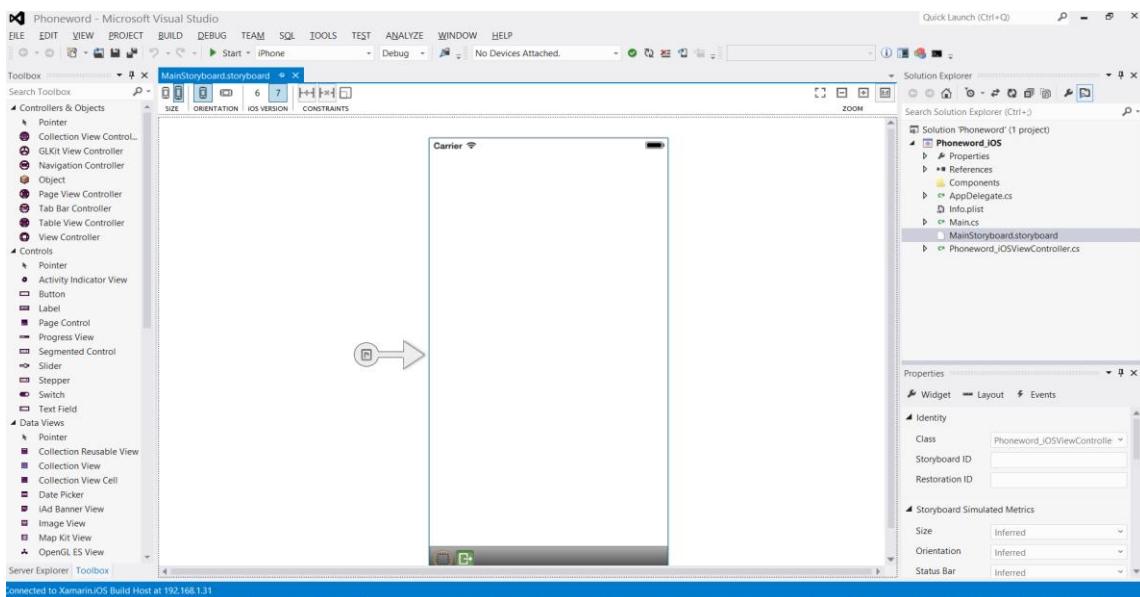
- iOS Designer는 자마린에서 사용자 인터페이스를 만들기 위한 비주얼 툴이다.
- Storyboard에서 응용프로그램의 스크린을 Scene이라한다.
- 각각의 Scene은 View Controller와 Content View Hierarchy를 관리하기 위한 View의 스택을 표현한다.
- iOS Single View Application Project를 생성하면 Main.storyboard 파일이 생성되며 이곳에서 UI를 만들게 된다.



- 스토리 보드(Storyboard)는 애플리케이션 화면의 시각적 디자인과 화면간의 전환 및 관계가 정의된 파일로 **스토리 보드에서 응용 프로그램의 화면을 표현하는 것을 Scene**이라고 한다.
- 각 Scene은 View Controller와 그것이 관리하는 View의 스택 (Content View Hierarchy)을 나타내는데 iOS Project Template에서 새로운 Single View Application Project를 만들면 Visual Studio는 Main.storyboard라는 스토리 보드 파일을 자동으로 생성하고 단일 Scene으로 채운다.

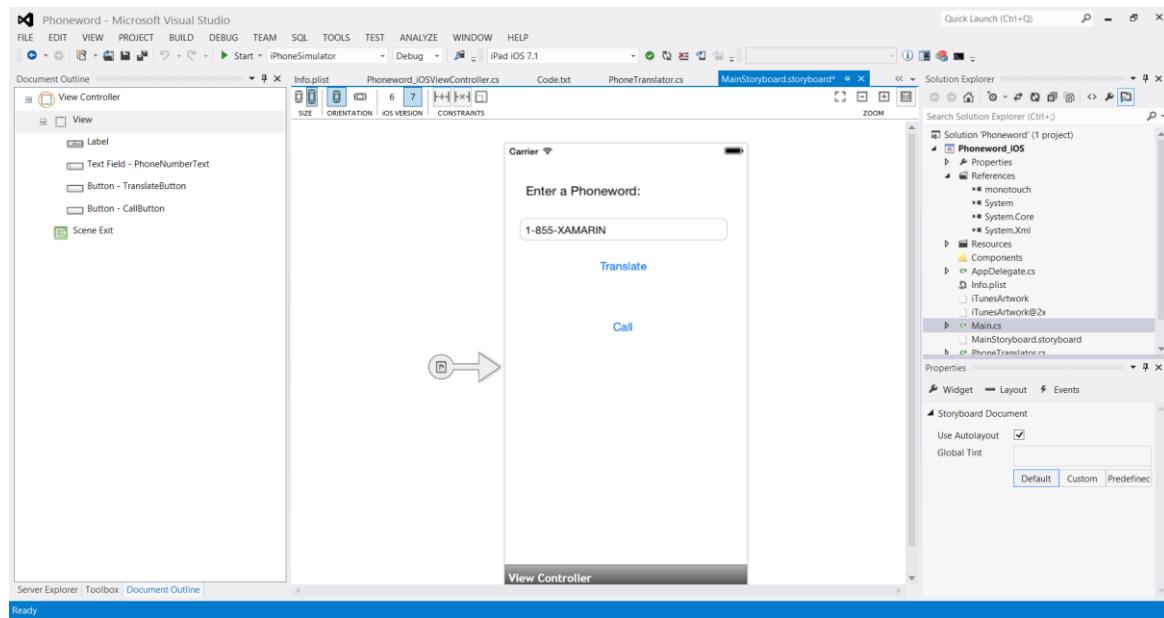


- 스토리 보드(Storyboard) 화면 하단의 가로막대를 선택하여 Scene에 대한 View Controller를 선택할 수 있다.
- View Controller는 Content View Hierarchy에 대한 백업 코드가 포함 된 UIViewController 클래스의 인스턴스로 속성 창에서 View Controller의 속성을 확인하고 설정할 수 있다.



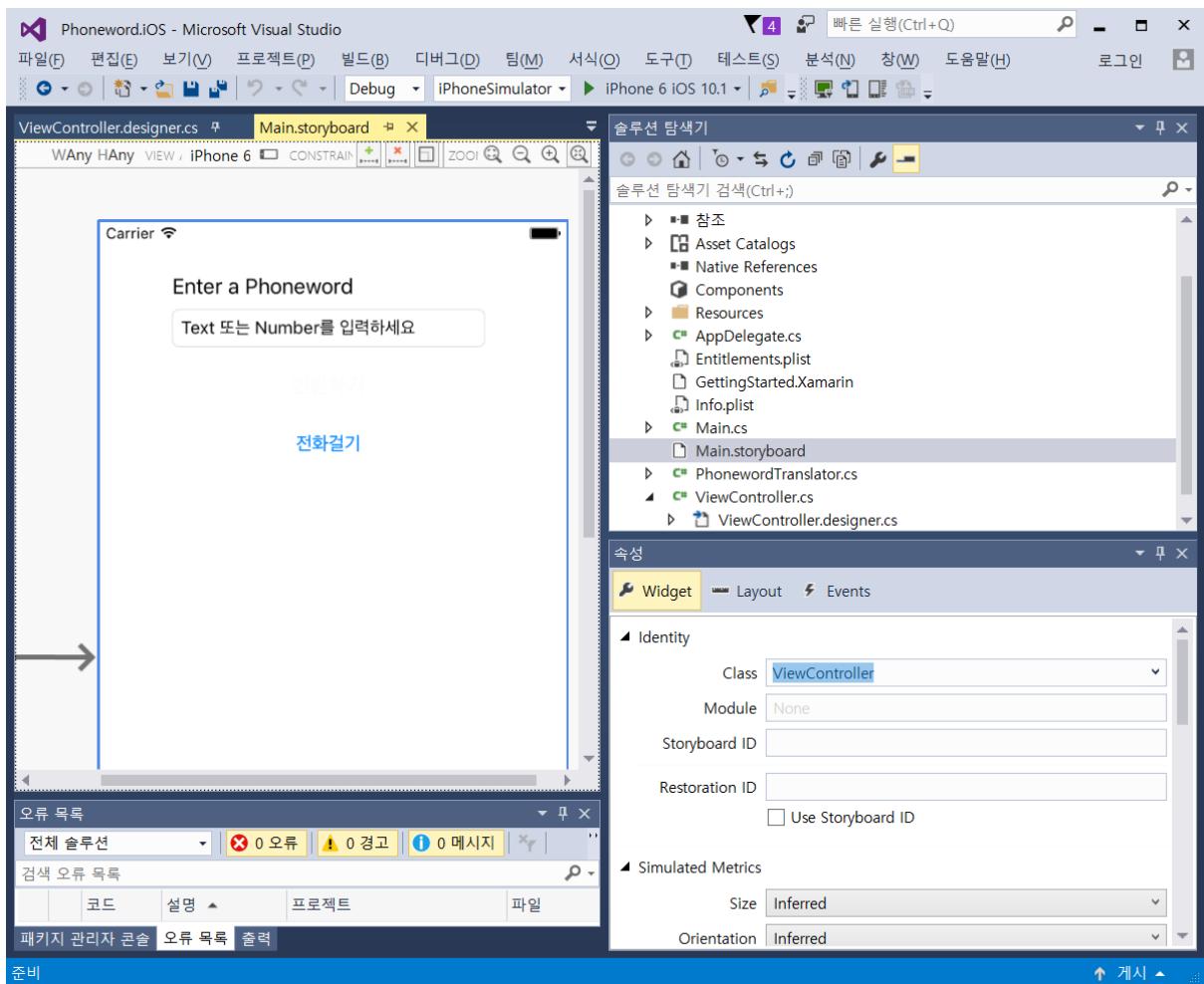
- iOS Designer의 좌측 회색빛 화살표는 스토리보드 전이를 나타내는데 Segue (pronounced "seg-way")라고 부른다. 이 Segue는 소스 원본이 없으므로 Sourceless Segue라고 하고 첫번

째 Scene을 가리킨다. 즉 응용프로그램이 시작할 때 Window에 처음 로드되는 Scene을 가리킨다.



3.4.4 View Controllers and the View Lifecycle

- Content View Hierarchy : View Controller에 의해 관리되는 View or Subview의 스택이다.
- View Controller는 Content View Hierarchy 안에 있는 View들을 관리하는 역할을 한다.
- View Controller는 Scene의 아래 검정색 Bar로 Storyboard안에 표현된다.
- Content View Hierarchy에 대해 사용자 정의 View Controller를 정의할 수 있는데 속성창의 Identity안의 Class 속성에서 지정할 수 있다.



- ViewController는 UIViewController의 하위 클래스로 아래와 같이 기본 모양을 가진다.

```
public partial class ViewController : UIViewController
{
    public ViewController (IntPtr handle) : base (handle)
    {
    }
}
```

- View의 Lifecycle 이벤트

View Controller는 Window로부터 Content View Hierarchy의 요소들을 로딩하거나 언로딩 한다. Content View Hierarchy안의 View에서 발생되는 중요한 일에 대해 View의 라이프 사이클 동안 OS는 이벤트를 통해 View Controller에 알린다.

ViewDidLoad : View Controller가 메모리에 Content View Hierarchy를 로드하는 시점에 한번 호출된다.

ViewWillAppear : View Controller의 View가 Content View Hierarchy에 추가되어 화면에 나타날 때마다 호출된다.(OnResume)

ViewWillDisappear : View Controller의 View가 Content View Hierarchy에서 제거되어 화면에서 사라질 때마다 호출된다. 주로 화면 Clear 또는 저장하는 경우에 사용된다.(OnPause)

ViewDidAppear and **ViewDidDisappear** : View가 Content View Hierarchy에서 추가되었거나 제거될 때 호출된다.

- 사용자의 상호작용에 대한 응답 : View Controller의 가장 중요한 역할은 버튼 조작, 탐색 등과 같은 사용자 상호 작용에 응답하는 것이다. 사용자 상호 작용을 처리하는 가장 간단한 방법은 컨트롤을 연결하여 사용자 요청/입력을 받고 응답하는 이벤트 핸들러를 만들어 연결하는 것인데 Hello iOS 예제처럼 버튼을 터치하여 터치 이벤트에 응답 할 수 있다.
- Properties 창에서 Button 컨트롤에 Name을 할당하면, iOS 디자이너는 ViewController 클래스의 내부에서 사용할 수 있도록 **ViewController.designer.cs**의 컨트롤에 자동으로 맵핑한다. View Lifecycle의 ViewDidLoad 단계에서 컨트롤이 먼저 사용 메서드 내에서 사용자의 터치에 응답 할 준비를 하는 것이다.

```
public override void ViewDidLoad ()  
{  
    base.ViewDidLoad ();  
  
    // wire up TranslateButton here  
}
```

- 본 교재의 Xamarin.iOS HelloWorld에서는 TouchUpInside라는 터치 이벤트를 사용하여 사용자의 터치에 대한 처리를 하는데 TouchUpInside는 컨트롤 범위 내에서 터치 다운(화면을 터치하는 손가락)을 수행하는 터치 업 이벤트(화면에서 손가락이 들리는 동작)를 수신한다.
- TouchUpInside의 반대는 사용자가 컨트롤을 눌렀을 때 발생하는 TouchDown 이벤트로 TouchDown 이벤트는 많은 노이즈를 캡처하고 손가락을 컨트롤에서 밀어서 터치를 취소 할 수 있는 옵션을 제공하지 않는다. TouchUpInside는 버튼 터치에 응답하는 가장 일반적인 방법이다.
- 람다를 사용하여 CallButton의 TouchUpInside 이벤트를 처리한 모양이다.

```
// View Controller는 화면으로부터 Content View Hierarchy의 요소들을 로딩하거나  
언로딩 한다. Content View Hierarchy안의 View에서 발생되는 중요한 일에 대해  
View의 라이프 사이클 동안 OS는 이벤트를 통해 View Controller에 알린다.  
// ViewDidLoad : View Controller가 메모리에 Content View Hierarchy를 로드하는 시  
점에 한번 호출하며 Xamarin.Android의 OnCreate 콜백함수와 같은 기능을 한다  
public override void ViewDidLoad()  
{  
    base.ViewDidLoad();
```

```

// 전화번호 입력창에 포커싱이 된 경우 키보드를 화면에서 사라지게
하기위해
PhoneNumberText.ResignFirstResponder();

CallButton.TouchUpInside += (object sender, EventArgs e) =>
{
    var url = new NSUrl("tel:" + PhoneNumberText.Text);

    // Use URL handler를 호출하는데 "tel:" 키워드는 애플의 Phone app을
    // 호출하고 그렇지 않으면 경고창을 띄운다. 본 예제는 시뮬레이터에서 동작하므로 경
    // 고창이 로드된다.
    if (!UIApplication.SharedApplication.OpenUrl(url))
    {
        //경고창을 띄운다.
        var alert =
            UIAlertController.Create("Not Supported", "Schema 'tel:' is not
supported on this device",
            UIAlertControllerStyle.Alert);
        alert.AddAction(UIAlertAction.Create("OK",
            UIAlertActionStyle.Default, null));
        PresentViewController(alert, true, null);
    }
};

}

```

3.3.5 추가적인 사항

- 버튼의 Text 변경 : CallButton의 텍스트를 "Call"로 변경

```
CallButtonSetTitle ("Call", UIControlState.Normal);
```

- 버튼의 활성화/비활성화

```
CallButton.Enabled = false;
```

- 키보드닫기(DDismiss the Keyboard) : 사용자가 Text Field 누르면 iOS는 사용자가 입력 할 수

있도록 키보드를 표시하는데 아쉽게도 키보드를 닫을 수 있는 기본 제공 기능이 없다. 사용자가 CallButton을 누르면 아래 코드를 TranslateButton에 추가하여 키보드를 닫는다.

```
PhoneNumberText.ResignFirstResponder();
```

- URL로 전화걸기 : Apple URL 스키마를 사용하여 시스템 전화 앱을 실행한다. 스키마는 'tel :' 접두어와 전화 할 전화 번호로 구성된다.

```
var url = new NSURL ("tel:" + translatedNumber);
if (!UIApplication.SharedApplication.OpenUrl (url))
{
    // show alert Controller
}
```

- 경고(Alert Dialog)창 표시 : 사용자가 시뮬레이터 또는 iPod Touch와 같이 통화를 지원하지 않는 장치에서 전화를 걸려고 할 때 통화를 할 수 없음을 알리는 경고 대화 상자를 표시한다 .

```
if (!UIApplication.SharedApplication.OpenUrl (url)) {
    var alert = UIAlertController.Create ("Not supported", "Scheme 'tel:' is
not supported on this device", UIAlertControllerStyle.Alert);
    alert.AddAction (UIAlertAction.Create ("Ok", UIAlertActionStyle.Default,
null));
    PresentViewController (alert, true, null);
}
```

3.4 Xamarin.iOS HelloWorld(멀티 뷰) 실습

- Xamarin.ios 단일뷰 앱 프로젝트를 생성하자. (프로젝트명 : **iOS_Hello**)

- **Main.storyboard**를 더블 클릭해서 아래 화면처럼 UI를 구성하자.

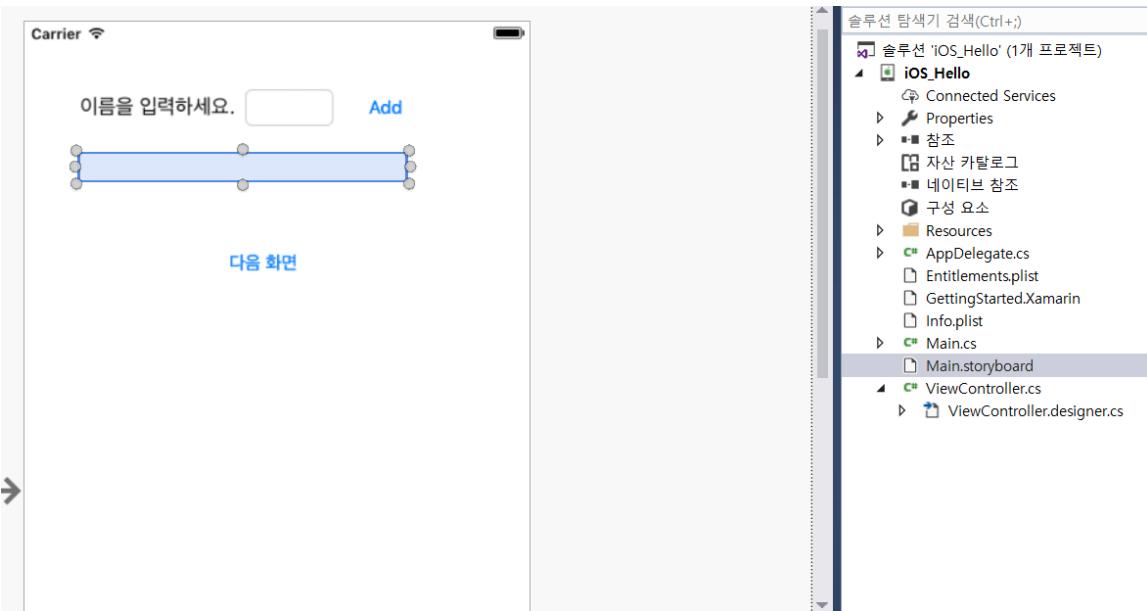
Label(Text : 이름을 입력하세요.)

Text Field(Name : txtName, Text : "")

Button(Name : btnAdd, Text : Add)

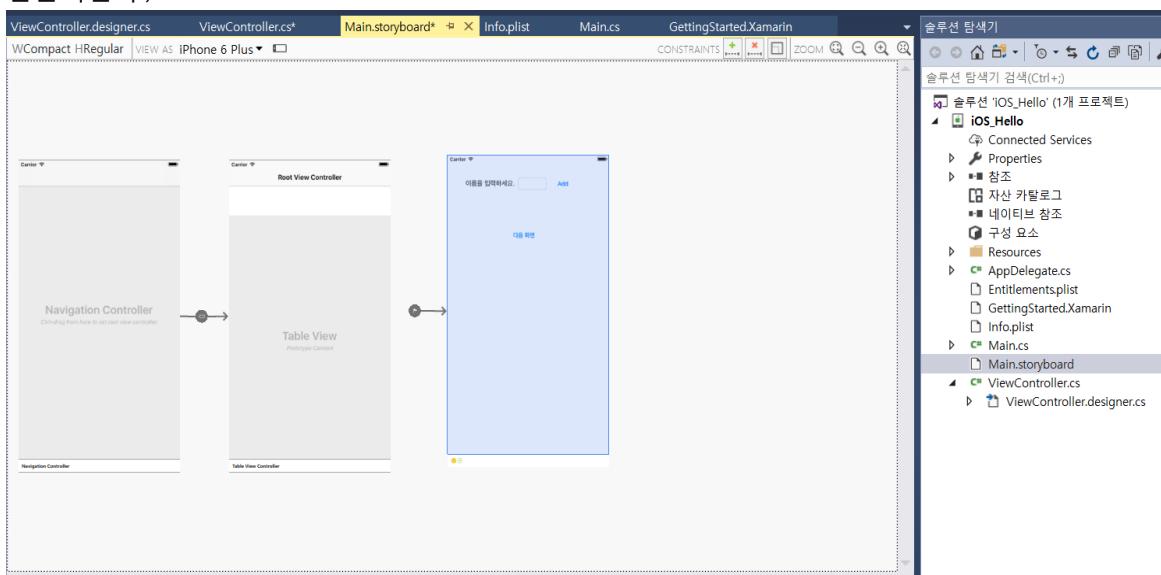
Label(Name : lblNames)

Button(Name : btnNext, Text : 다음 화면)

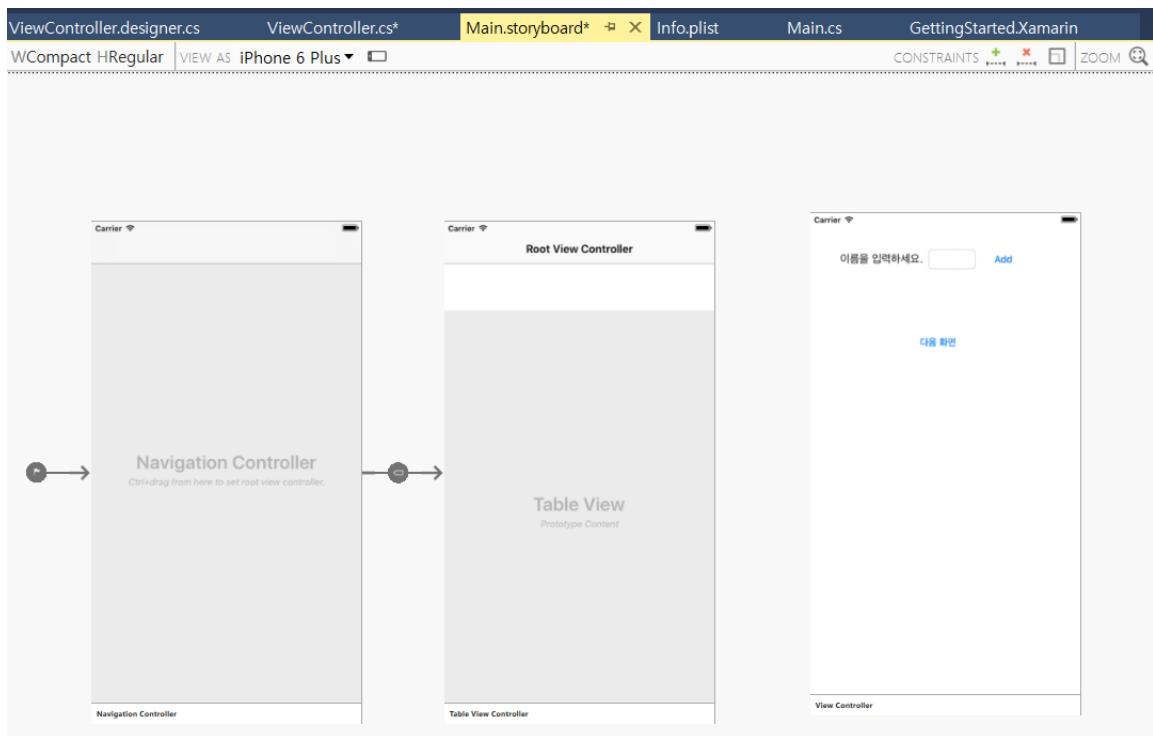


■ 도구상자에서 **Navigation Controller**를 디자인 영역으로 드래그 하자.

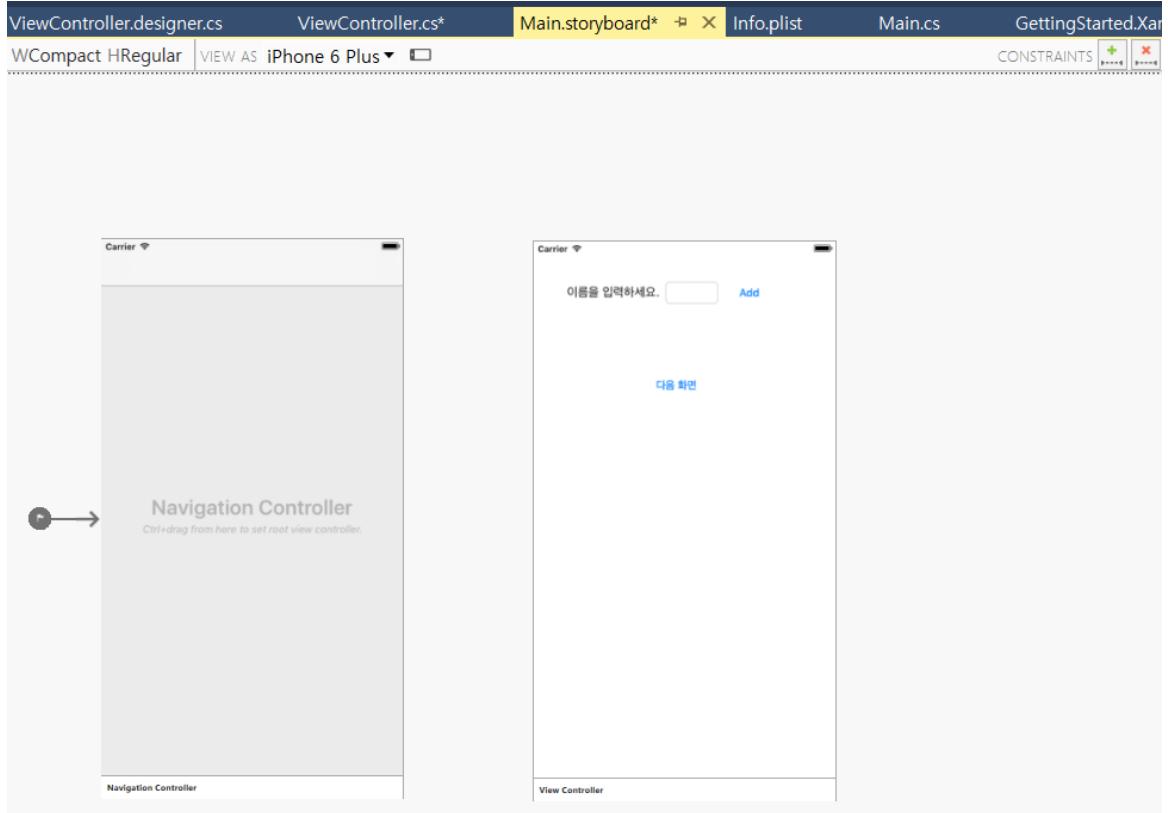
(기존에 만들어진 뷰(Scene)의 왼쪽에 위치시키면 Navigation Controller와 Table View 2개가 만들어진다.)



■ 기존 만들어진 Scene 왼쪽에 있는 화살표(Sourceless Segue)를 Navigation Controller에 이동시키자. (응용프로그램의 시작점을 변경)

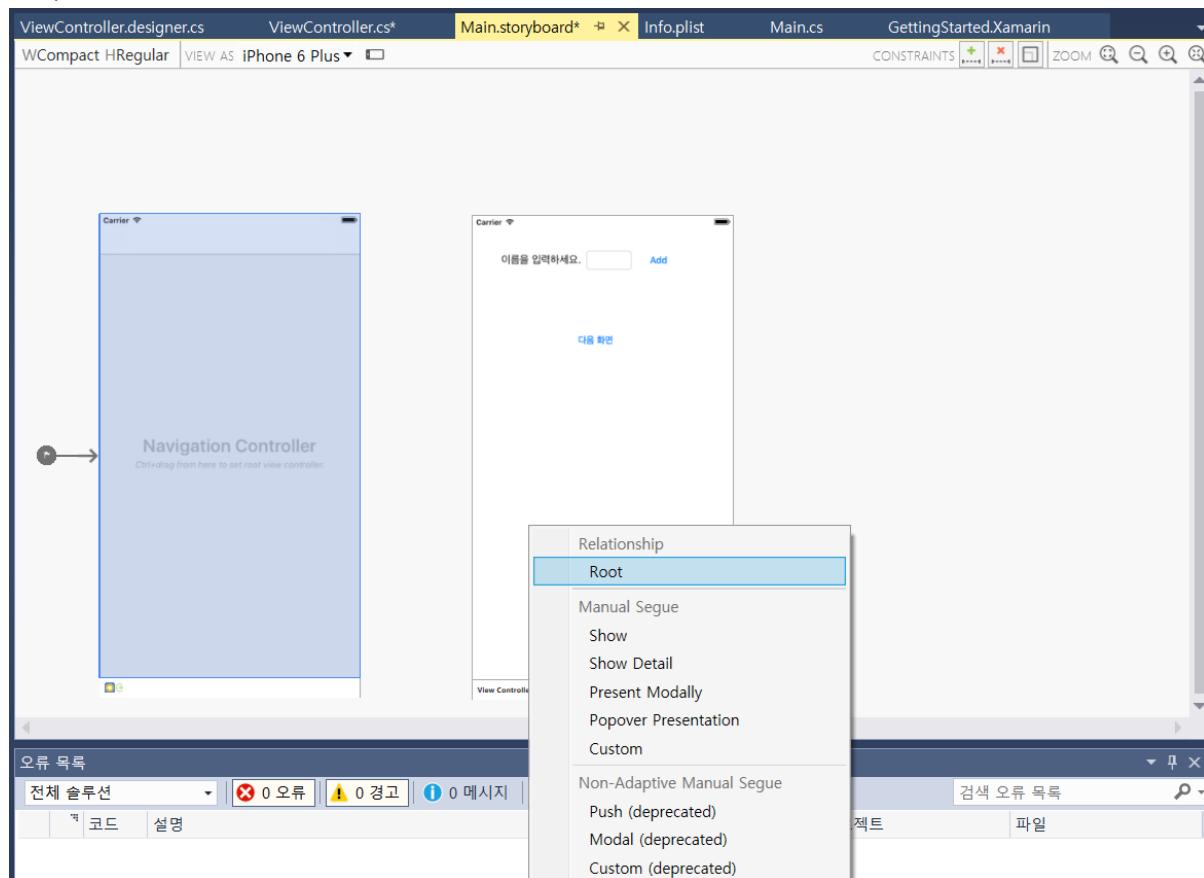


- Root View Controller의 검정색바를 클릭 후 삭제하자. (처음 만든 Scene이 Navigation Controller의 우측에 위치한다.)

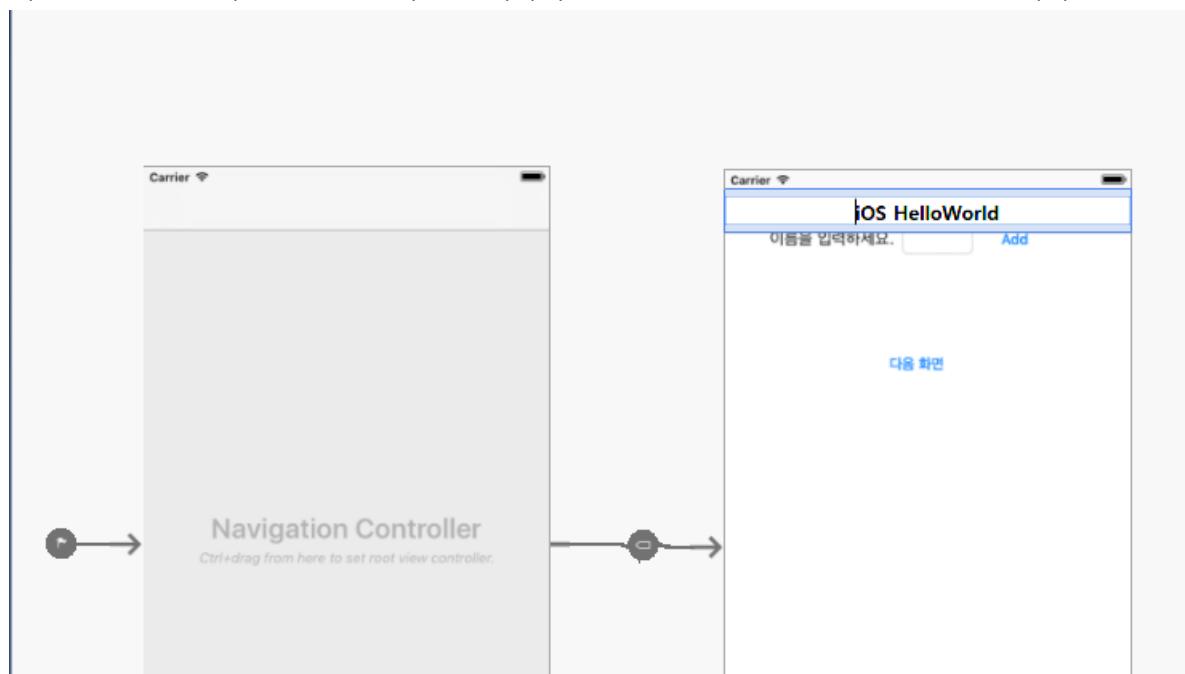


- Navigation Controller의 Root View Controller로서 ViewController를 설정하자. Navigation

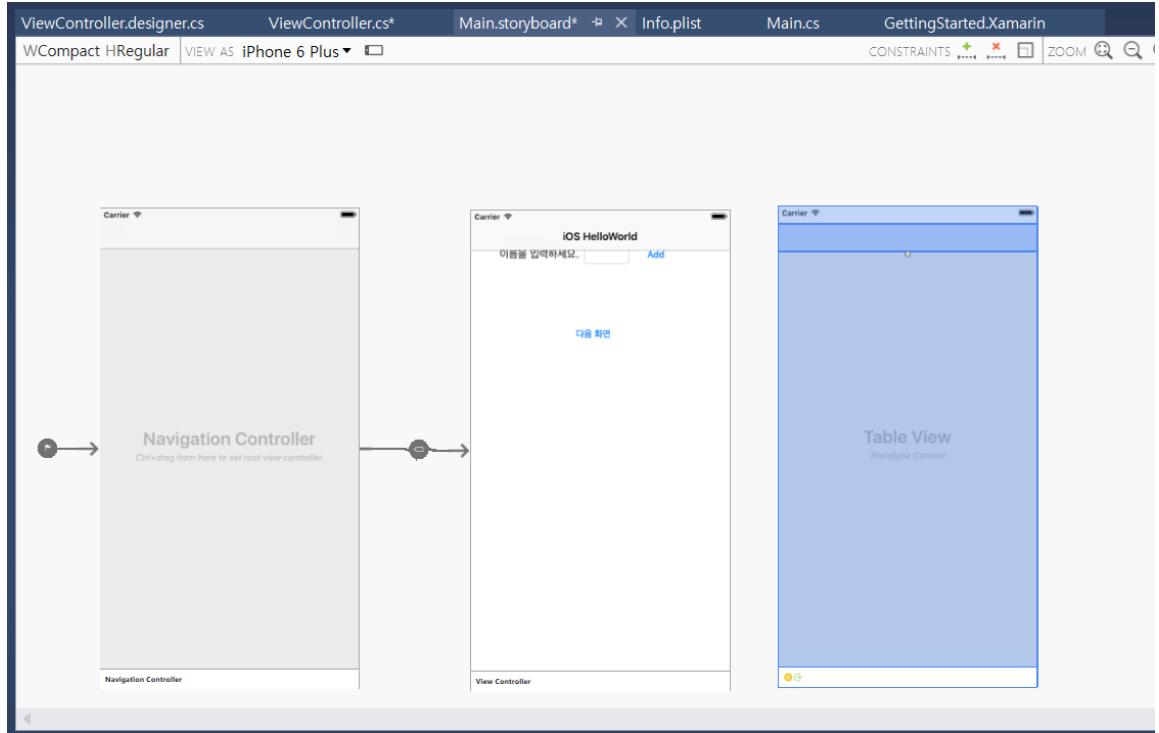
Controller에서 Ctrl + Down 후 맨처음 만든 Scene으로 드래그 하자. 파랑색 선이 생기며 이것을 Ctrl-Dragging이라고 한다. 이때 팝업창이 뜨는데 Relationship을 Root로 선택하자. 그러면 원래 만든 Scene의 ViewController가 Navigation Controller의 Root View Controller가 된다.



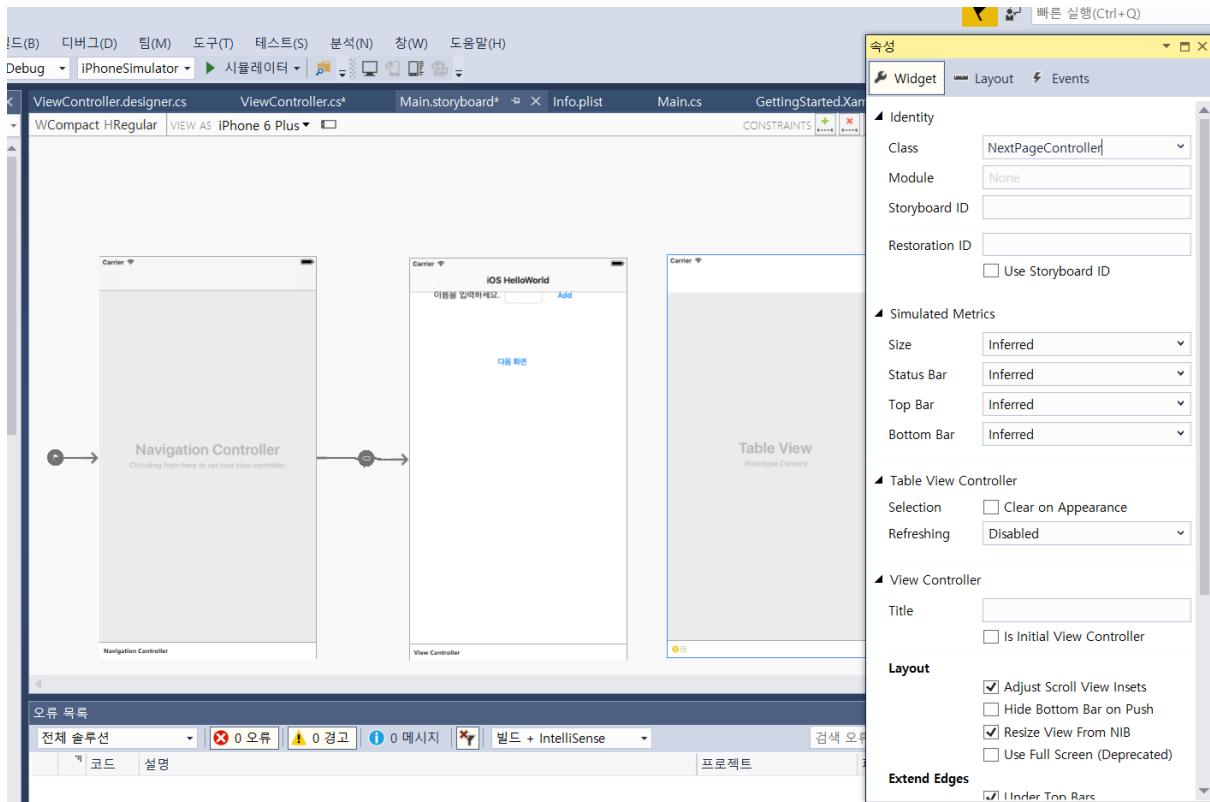
- 처음 만든 Scene의 Title Bar를 더블 클릭하여 Title을 "iOS HelloWorld"로 변경하자.

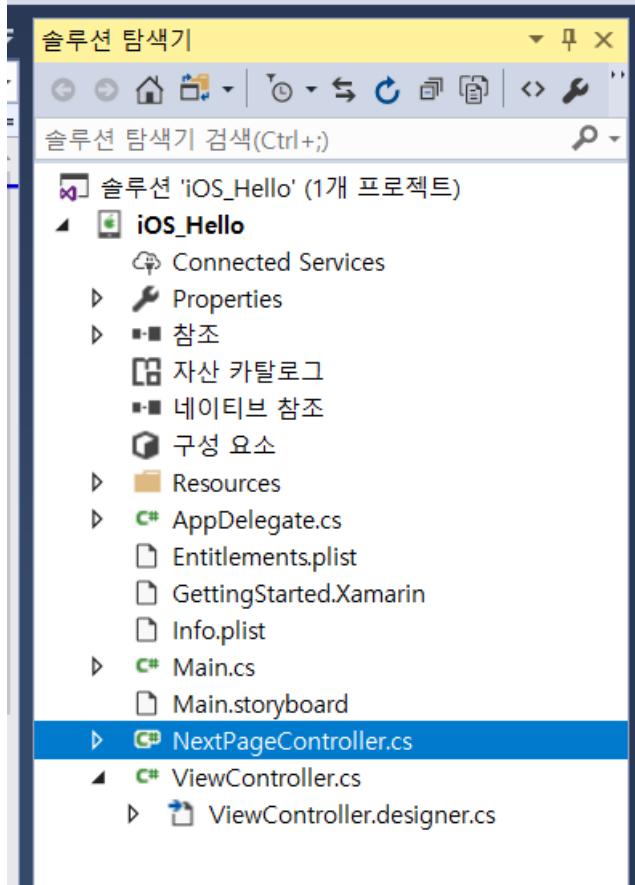


- 사용자가 입력한 이름들을 출력해주는 Scene을 만들자. 도구상자에서 Table View Controller를 디자인 영역의 원래 있던 Scene의 오른쪽으로 드래그 하자.



- Table View Controller의 하단 Table View Controller Bar를 클릭하고 속성(F4)창 Class 속성에 “**NextPageController**” 라고 입력하자. 솔루션 탐색기에 NextPageController.cs 파일이 생기니 확인하자.





- **NextPageController.cs** 파일을 클릭하여 아래 코드를 입력하자.

```
using Foundation;
using System;
using System.Collections.Generic;
using UIKit;

namespace iOS_Hello
{
    public partial class NextPageController : UITableViewController
    {
        public List<string> names { get; set; }
        static NSString nextPageCellId = new NSString("NextPage");
        public NextPageController (IntPtr handle) : base (handle)
        {
            TableView.RegisterClassForCellReuse(typeof(UITableViewCell),
nextPageCellId);
            TableView.Source = new NextPageDataSource(this);
            names = new List<string>();
        }
    }
}
```

```

        }

    class NextPageDataSource : UITableViewSource
    {
        NextPageController controller;

        public NextPageDataSource(NextPageController controller)
        {
            this.controller = controller;
        }

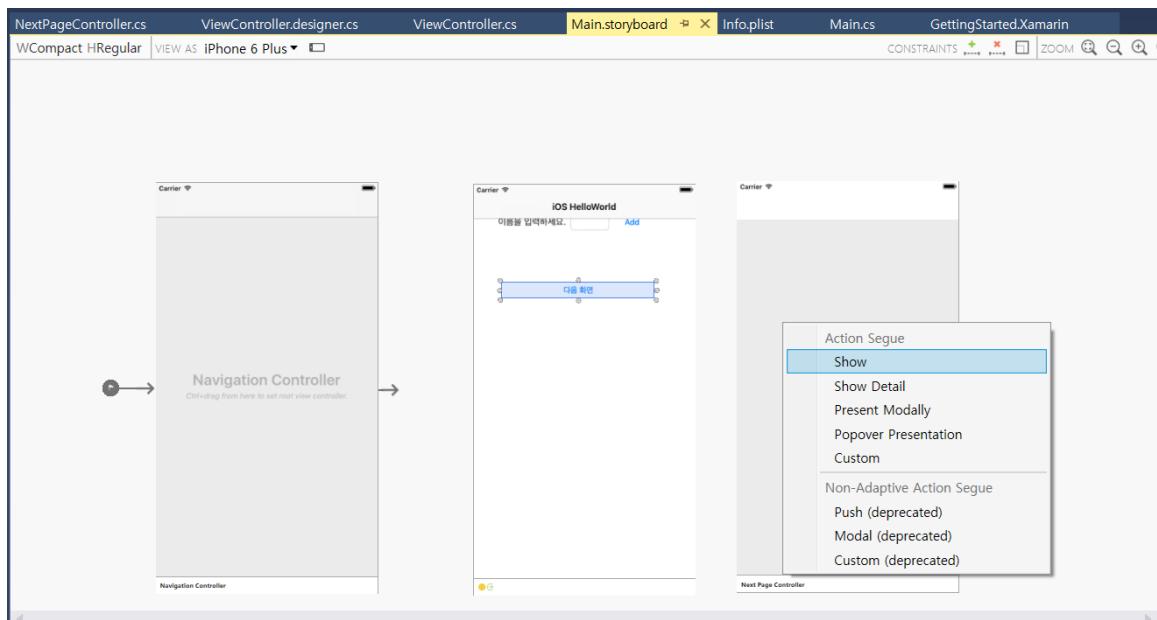
        // Returns the number of rows in each section of the table
        public override nint RowsInSection(UITableView tableView,
nint section)
        {
            return controller.names.Count;
        }

        public override UITableViewCell GetCell(UITableView
tableView, NSIndexPath indexPath)
        {
            var cell =
tableView.DequeueReusableCell(NextPageController.nextPageCellId);

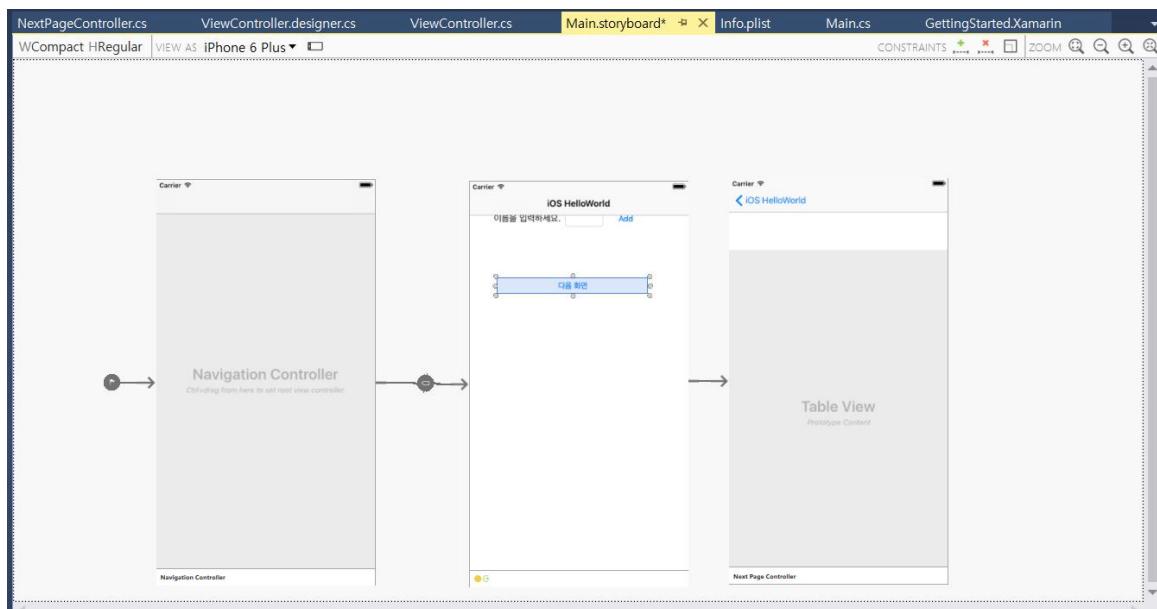
            int row = indexPath.Row;
            cell.TextLabel.Text = controller.names[row];
            return cell;
        }
    }
}

```

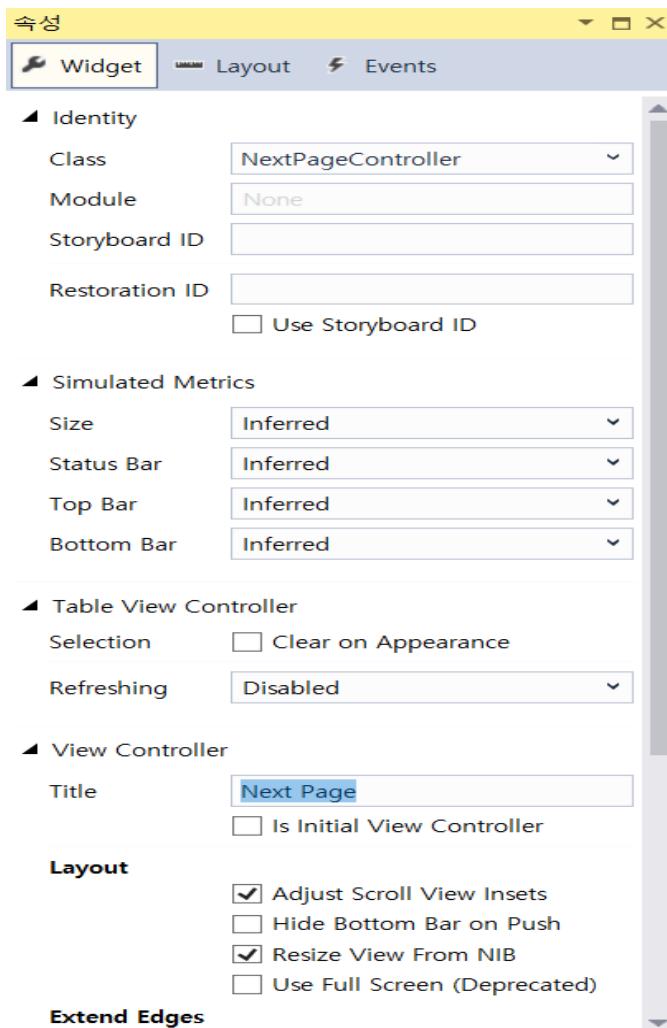
- 처음 만든 Scene과 NextPage Scene을 연결하여 Seque를 만들자. “다음화면” 버튼에서 Ctrl 키를 누른채로 NextPage Scene으로 드래그 하자. 팝업창이 나타나면 “Show”를 선택.



두 Scene 사이에 하나의 Segue가 생긴다.



- 두 번째 추가된 Scene(NetPage)의 하단 Bar를 클릭 후 F4(속성창)를 틀릭하여 Title 속성에 “**Next Page**”라고 입력하자.



- 응용프로그램이 시작될 때 “다음화면” 버튼은 NextPage Scene을 오픈하는데 사용자가 입력한 이름들을 넘겨줘서 띄우게 된다. 코드를 작성하자.(**ViewController.cs**)

```
using System;
using System.Collections.Generic;
using Foundation;
using UIKit;

namespace iOS_Hello
{
    public partial class ViewController : UIViewController
    {
        public List<string> names { get; set; }

        public ViewController(IntPtr handle) : base(handle)
        {
        }
    }
}
```

```
public override void ViewDidLoad()
{
    base.ViewDidLoad();
    // Perform any additional setup after loading the view,
    // typically from a nib.

    // 이름입력창에 포커싱 된 이후 키보드를 사라지게 하기위해
    txtName.ResignFirstResponder();
}

public override void DidReceiveMemoryWarning()
{
    base.DidReceiveMemoryWarning();
    // Release any cached data, images, etc that aren't in use.
}

partial void BtnAdd_TouchUpInside(UIButton sender)
{
    lbNames.Text += " " + txtName.Text;
}

partial void BtnNext_TouchUpInside(UIButton sender)
{
    string[] names2 = lbNames.Text.Split(' ');
    names = new List<string>(names2);
}

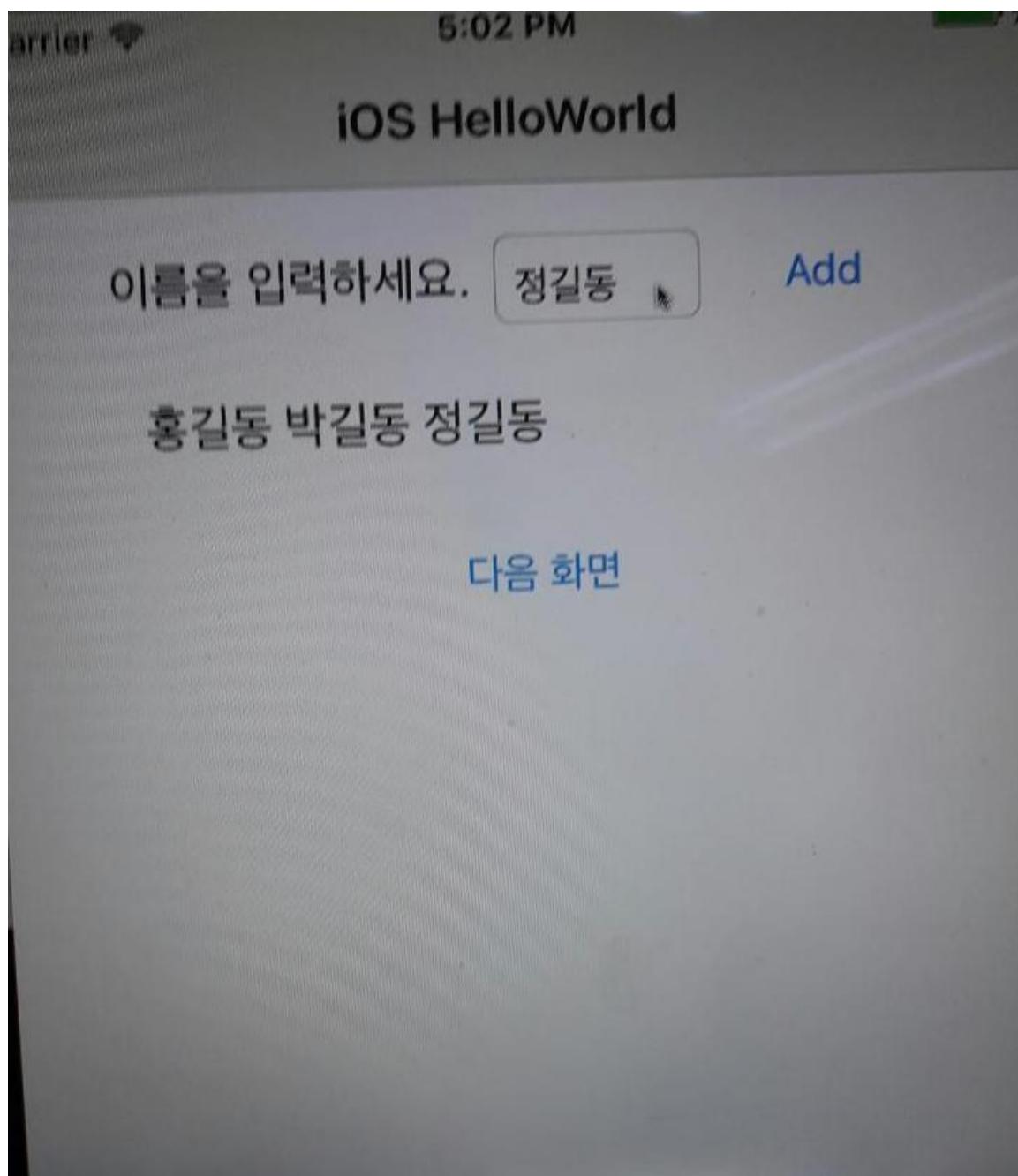
public override void PrepareForSegue(UIStoryboardSegue segue,
NSObject sender)
{
    base.PrepareForSegue(segue, sender);

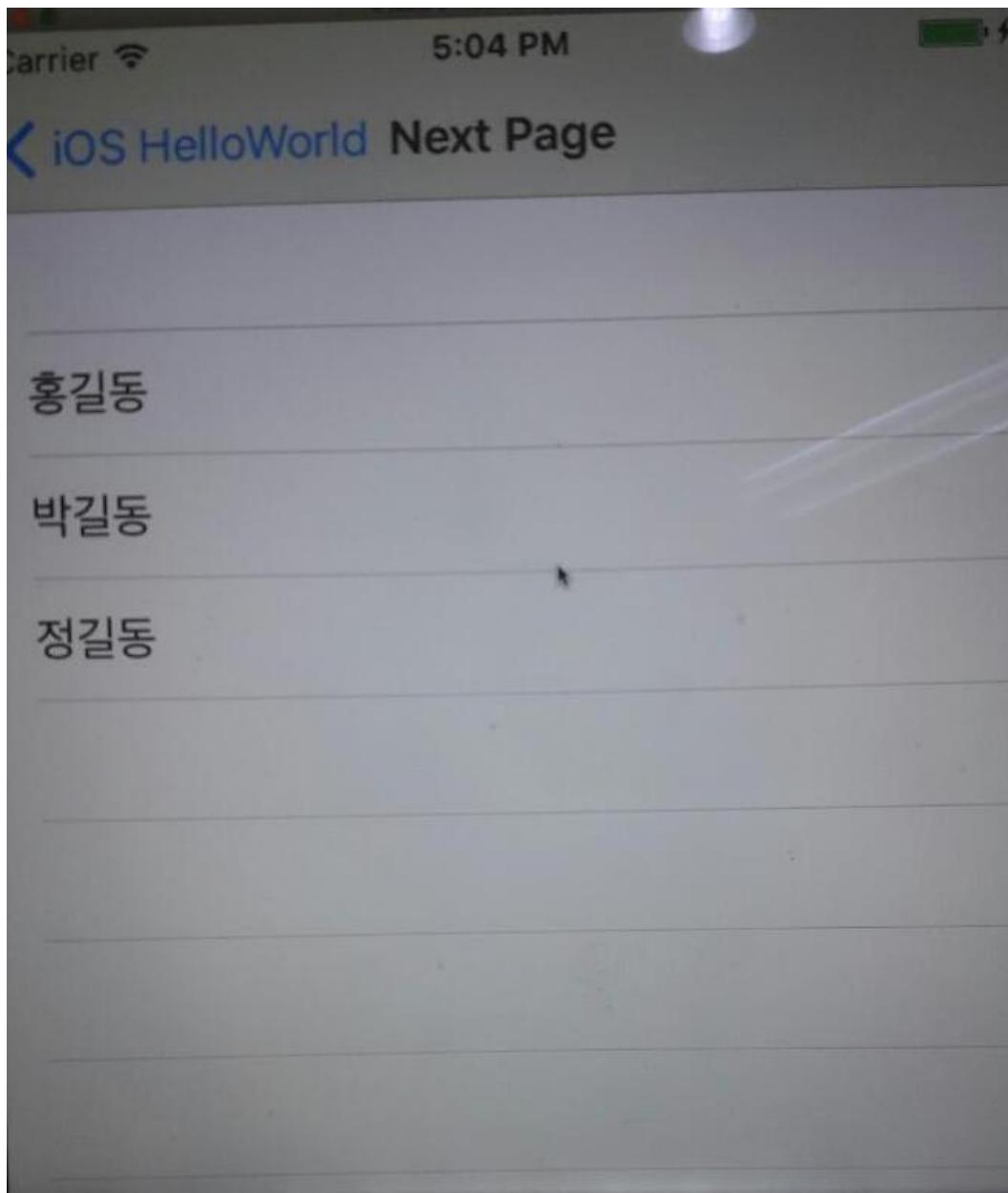
    // 새로 띄울 Scene의 ViewController를 설정
    var nextPageController = segue.DestinationViewController as
    NextPageController;

    if (nextPageController != null)
    {
        nextPageController.names = names;
    }
}
```

}

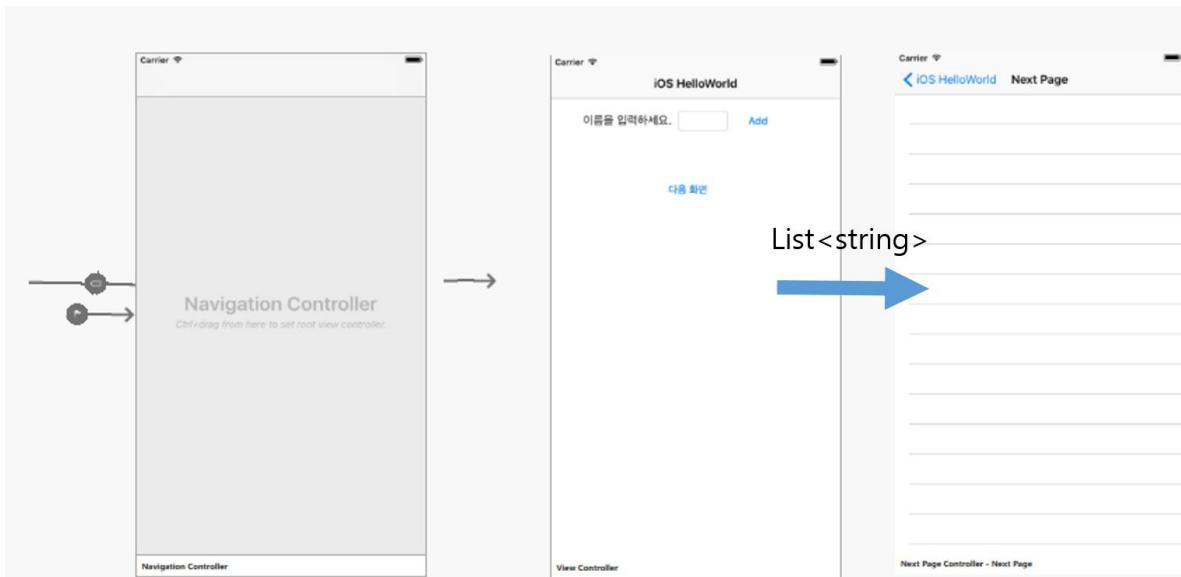
■ 실행 화면



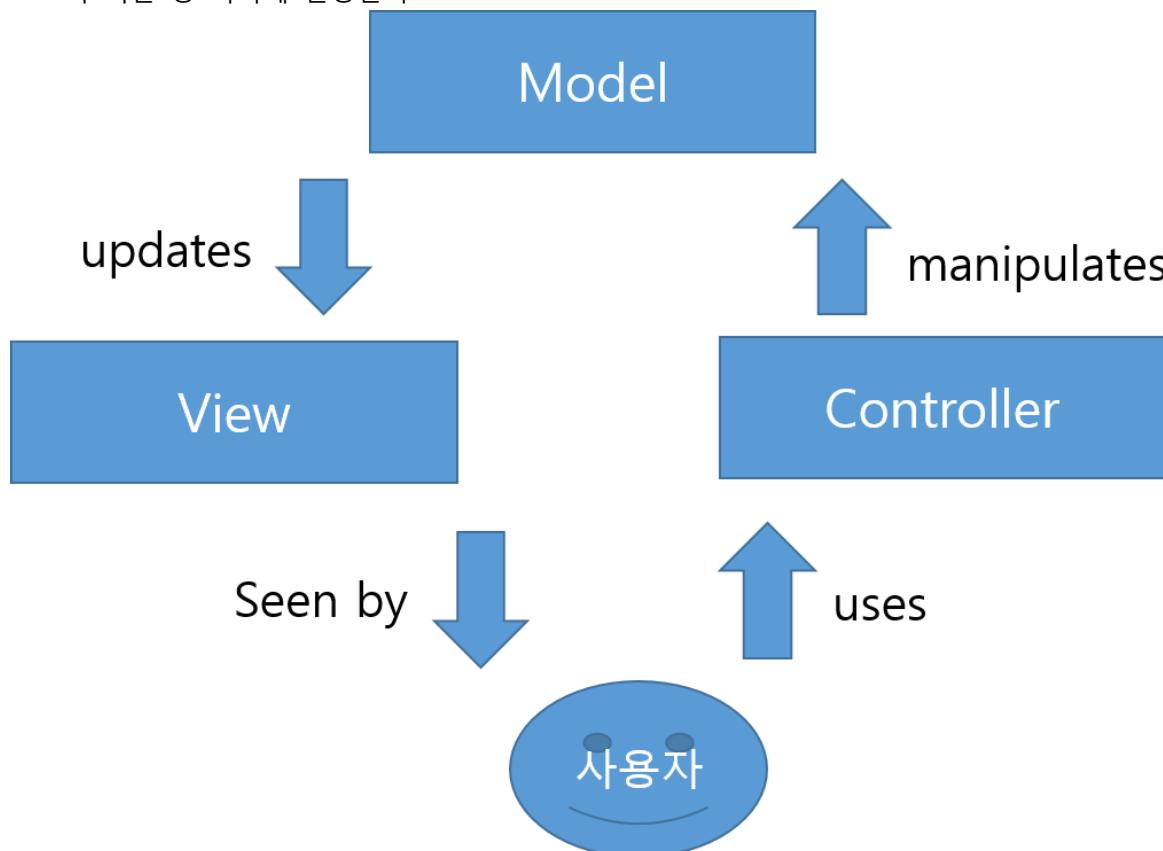


3.4.1 Xamarin.iOS HelloWorld(멀티 뷰) 자세히 살펴보기_MVC, Navigation Controller, View Controller

- 첫번째 작성한 Xamarin.iOS 예제를 통해 View Controller가 자신의 Content View 계층 구조를 Window에 로드하는 하나의 Window만 가지고 있음을 알수 있었고 두번째 멀티화면 예제를 통해 새로운 화면(Scene)을 추가하고 아래 그림과 같이 두 화면(Scene) 사이에 이름을 전달했다.



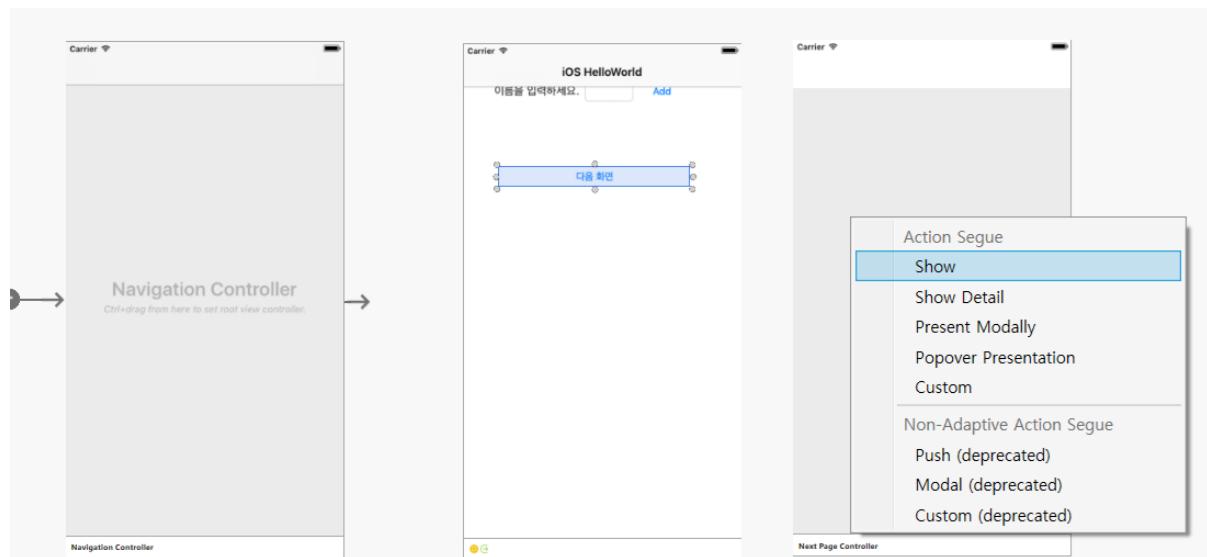
- 사용자가 입력한 여러 개의 이름은 첫 번째 화면에서 입력되고 첫 번째 View Controller에서 두 번째 화면(Scene)으로 전달되어 두 번째 화면에 표시된다. 화면, 뷰 컨트롤러 및 데이터의 분리는 모델, 뷰, 컨트롤러 (MVC) 패턴을 따른다.
- Model-View-Controller는 디자인 패턴으로 재사용 가능한 아키텍처 인데, MVC는 그래픽 사용자 인터페이스(GUI)가 있는 응용 프로그램의 아키텍처이다. 애플리케이션의 객체를 모델 (데이터 또는 애플리케이션 로직), 뷰 (사용자 인터페이스) 및 컨트롤러 (코드 비하인드)의 세 가지 역할 중 하나에 할당한다.



- 모델(Model) : Model 객체는 일반적으로 View에 표시되거나 입력되는 데이터의 표현으로 모

델은 종종 느슨하게 정의된다. 예를 들어, 멀티 스크린 예제에서 이름 목록 (문자열 목록으로 표시됨)이 모델이다. MVC는 모델의 데이터 지속성과 액세스에 대해 완전히 독립적이다. 즉, MVC는 데이터의 모양이나 저장 방법, 데이터 표현 방법 만 신경 쓰고 있다. 예를 들어 데이터를 SQL 데이터베이스에 저장하거나 일부 클라우드 저장 메커니즘에 저장하거나 단순히 List <string>을 사용할 수 있다

- **뷰(View)** : 사용자 인터페이스의 렌더링을 담당하는 구성 요소로 MVC 패턴을 사용하는 거의 모든 플랫폼에서 사용자 인터페이스는 뷰 계층 구조로 구성된다. 이전 예제에서 뷰를 하나의 뷰 (루트 뷰라고 함)가 있는 뷰 계층 구조와 계층 구조의 맨 위와 하위 뷰 (하위 뷰 또는 하위 뷰)로 생각할 수 있다. iOS에서 화면의 콘텐츠 뷰 계층 구조는 MVC의 View에 해당한다.
- **컨트롤러(Controller)** : Controller는 모든 것을 연결하고 UIViewController에 의해 iOS에 표시되는 구성 요소로 컨트롤러를 화면 또는 뷰 집합의 백업 코드로 생각할 수 있다. 컨트롤러는 사용자의 요청을 받아들이고 적절한 뷰를 리턴한다. View (버튼 클릭, 텍스트 입력 등)의 요청을 받아들이고 적절한 처리, View 수정 및 View 리로드를 수행한다. 컨트롤러는 응용 프로그램에 있는 백업 데이터 저장소가 무엇이든 관계없이 Model을 작성하거나 검색하고 해당 데이터로 View를 채우는 일도 담당한다. 컨트롤러는 다른 컨트롤러도 관리 할 수 있는데 예를 들어, 한 컨트롤러가 다른 화면을 표시해야 할 경우 다른 컨트롤러를 로드하거나 컨트롤러 스택을 관리하여 순서와 그 사이의 전환을 모니터링 할 수 있다.
- **화면전환처리** : 멀티화면 예제에서 Storyboard Segue와 프로그래밍 방식 두 가지 방법으로 두 개의 View Controller 간의 화면 이동을 처리했다.
- **PrepareForSegue** : Storyboard에 Show 액션이 있는 Segue를 추가 할 때, iOS에게 두 번째 View Controller를 Navigation Controller의 스택으로 푸시하도록 지시한다.



- Storyboard에 Segue를 추가하면 화면 사이의 간단한 전환을 만들 수 있다. View Controller간에 데이터를 전달하려면 PrepareForSegue 메서드를 재정의하고 데이터를 직접 처리해야 한다.

```

public override void PrepareForSegue (UIStoryboardSegue segue, NSObject sender)
{
    base.PrepareForSegue (segue, sender);
    ...
}

```

- iOS는 화면전환이 발생하기 바로 전에 `PrepareForSegue`를 호출하고 스토리 보드에서 생성한 Segue를 메서드에 전달한다. 이 시점에 Segue의 대상 View Controller를 수동으로 설정해야 한다. 아래 코드는 대상 뷰 컨트롤러에 대한 핸들을 가져 와서 적절한 클래스인 `NextPageController`로 캐스팅 한다.
- 마지막으로 `NextPageController`의 `names` 속성에 View Controller의 이름(Model)을 `NextPageController`로 전달한다.

```

public override void PrepareForSegue(UIStoryboardSegue segue, NSObject sender)
{
    base.PrepareForSegue(segue, sender);

    // 새로 띄울 Scene의 ViewController를 설정
    var nextPageController = segue.DestinationViewController as
NextPageController;

    if (nextPageController != null)
    {
        nextPageController.names = names;
    }
}

```

- **Navigation Without Segues** : C# 코드에서 첫 번째 View Controller에서 두 번째 View Controller로의 전환은 Segue와 동일한 프로세스이지만 몇 단계는 수동으로 수행해야 한다. 먼저, `this.NavigationController`를 사용하여 현재 스택이 있는 네비게이션 컨트롤러에 대한 참조를 오고 그 다음 Navigation Controller의 `PushViewController` 메서드를 사용하여 다음 View Controller를 수동으로 스택에 푸시하여 View Controller를 전달하고 화면전환을 애니메이션화하는 옵션을 설정한다 (이 값을 `true`로 설정).
- 다음 코드는 이름입력 화면에서 이름들을 출력하는 두 번째 화면으로의 전환을 처리한다.

```

this.NavigationController.PushViewController (nextPage, true);

```

- 다음 View Controller로 전환하기 전에 Storyboard.InstantiateViewController를 호출하고 NextPageController의 Storyboard ID를 전달하여 Storyboard에서 수동으로 인스턴스화 해야한다.

```
CallHistoryController callHistory =
thisStoryboard.InstantiateViewController
("NextPageController") as NextPageController;
```

- 마지막으로 Segue로 전환을 처리 할 때와 마찬가지로 NextPageController의 names 속성을 넘어오는 이름들로 설정하여 ViewController의 이름들(모델)을 NextPageController로 전달한다.

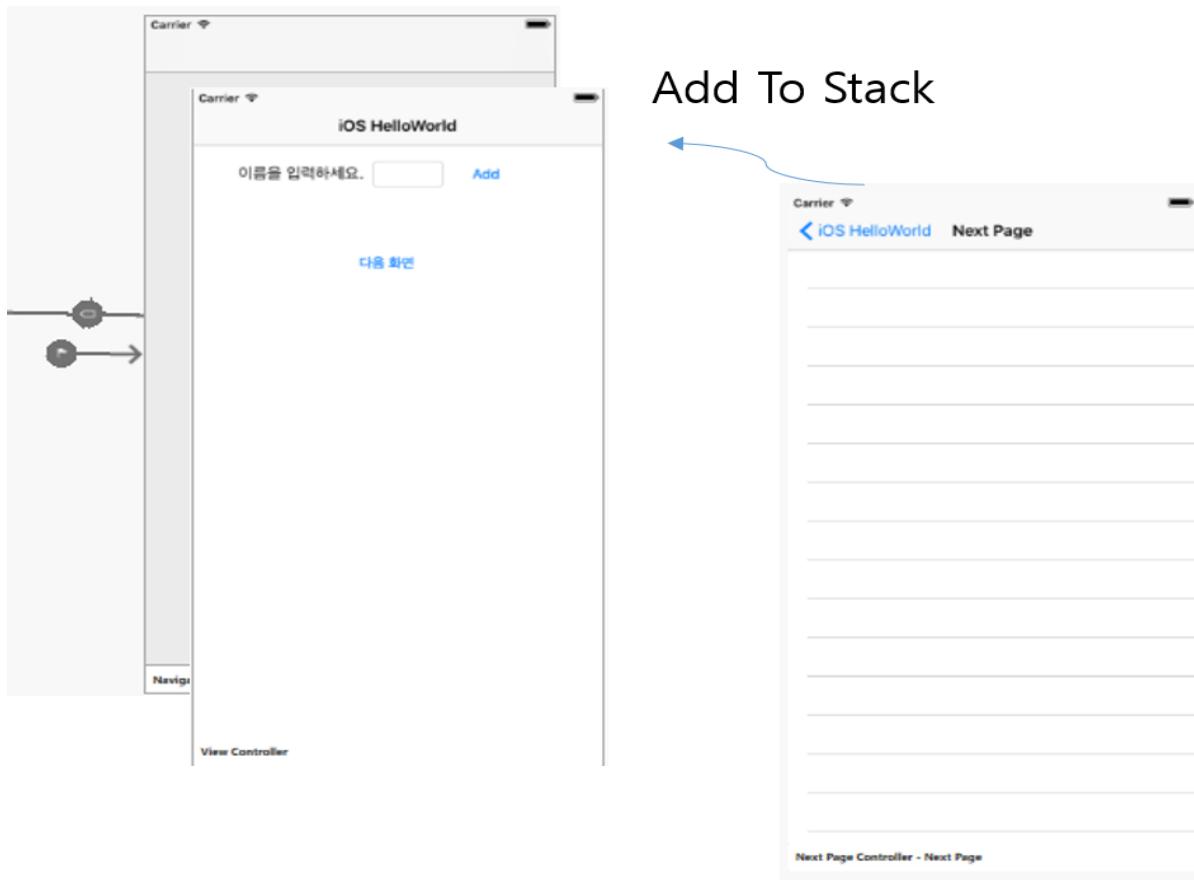
```
nextPageController.names = names;
```

- 아래는 프로그래밍적으로 화면 전환을 하는 완성된 코드이다.

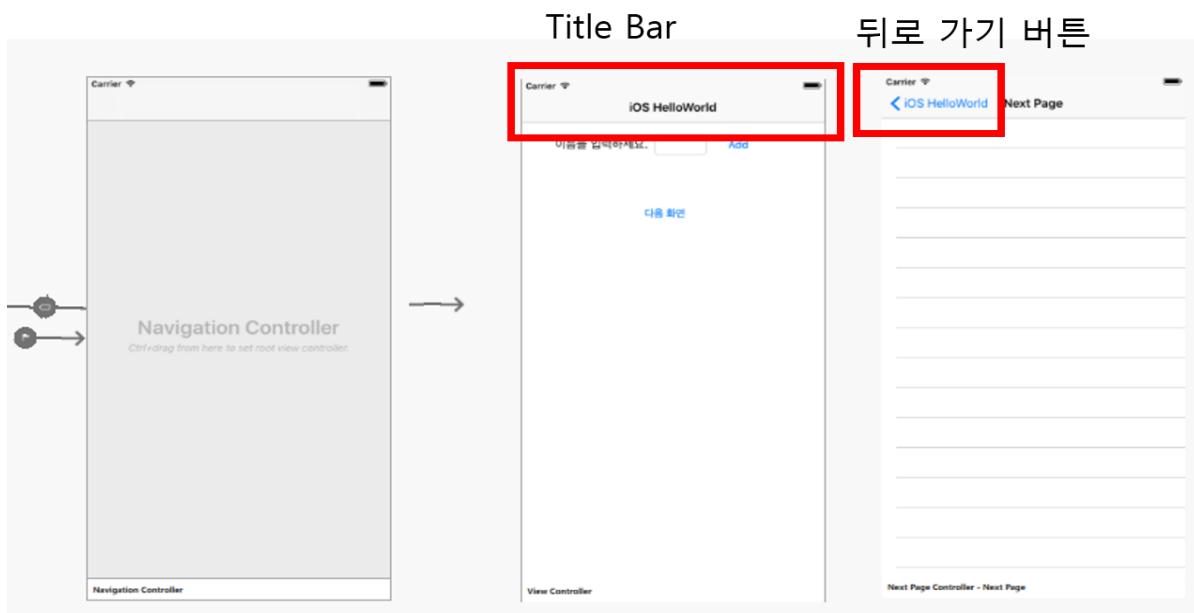
```
CallHistoryButton.TouchUpInside += (object sender, EventArgs e) => {
    // Launches a new instance of CallHistoryController
    NextPageController nextPageController = thisStoryboard.InstantiateViewController
    ("NextPageController") as NextPageController;
    if (nextPageController != null) {
        nextPageController.names = names;
        this.NavigationController.PushViewController(nextPageController, true);
    }
};
```

3.5 네비게이션 컨트롤러(Navigation Controller)

- 멀티 화면 예제에서 Navigation Controller를 사용하여 여러 화면 간의 전환을 관리했는데 네비게이션 컨트롤러는 UINavigationController 클래스로 표시되는 특수화 된 UIViewController로 하나의 컨텐츠 뷰 계층 구조를 관리하는 대신 네비게이션 컨트롤러는 제목, 뒤로 버튼 및 기타 선택적 기능을 포함하는 네비게이션 툴바의 형태로 고유 한 컨텐츠 뷰 계층 구조 뿐만 아니라 다른 뷰 컨트롤러를 관리한다.
- 네비게이션 컨트롤러는 iOS 애플리케이션에서 일반적으로 사용되며 설정 앱과 같은 주요 iOS 애플리케이션에 대한 네비게이션 기능을 제공한다.
- **네비게이션 컨트롤러의 3가지 주요 기능**
- **1. 전달 탐색(forward Navigation)을 위한 후크 제공** - 네비게이션 컨트롤러는 계층 구조 탐색 메타포(Hierachal Navigation Metaphor)를 사용하여 컨텐츠 뷰 계층 구조가 네비게이션 스택에 푸시된다. 네비게이션 스택을 아래 그림과 같이 가장 위에 있는 카드 만 보이는 카드 더미로 생각할 수 있다.

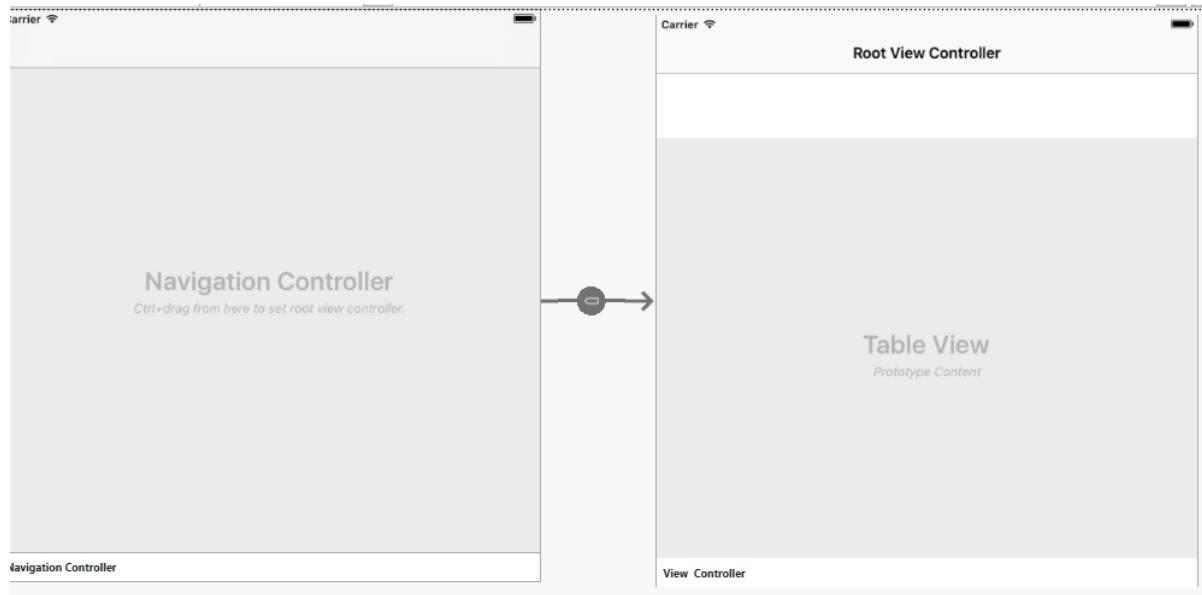


- 2. 뒤로 버튼 제공 - 새 항목을 네비게이션 스택에 푸시하면 제목 표시 줄에 자동으로 뒤로 버튼이 표시되어 사용자가 뒤로 이동할 수 있다. 뒤로 버튼을 누르면 현재 View Controller가 네비게이션 스택에서 빠져 나오고 이전 Content View Hierarchy가 Window에 로드된다.
- 3. 제목 표시 줄 제공 - 네비게이션 컨트롤러의 상단 부분을 “제목 표시 줄”이라고 한다. 아래 그림과 같이 View Controller 제목을 표시한다.

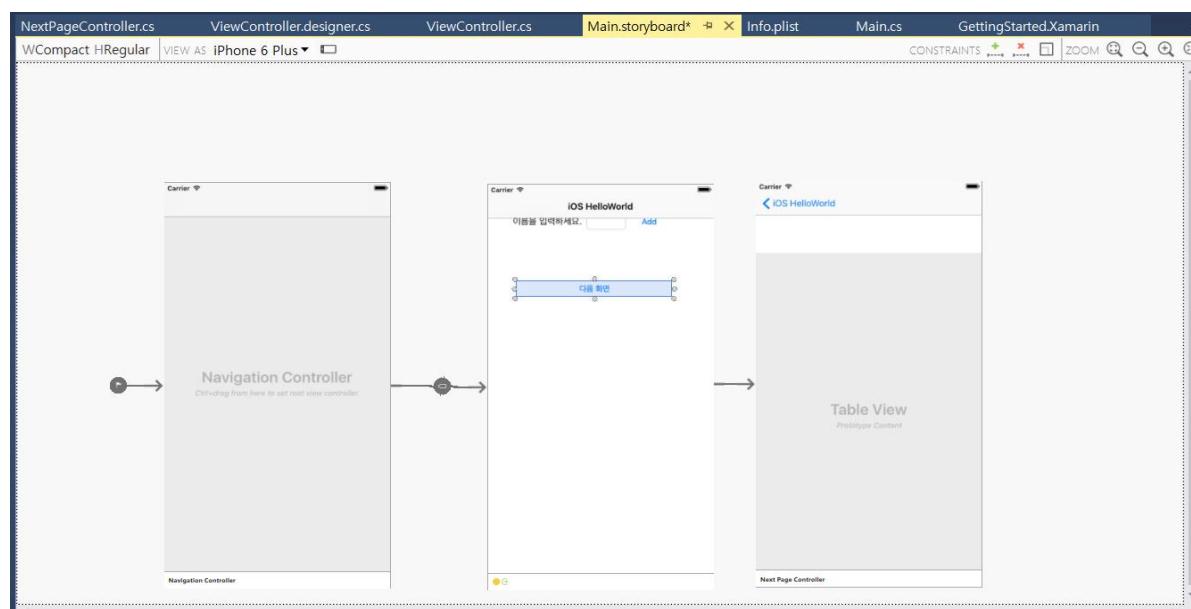


3.6 루트 뷰 컨트롤러(Root View Controller)

- 네비게이션 컨트롤러는 콘텐츠 뷰 계층 구조를 관리하지 않으므로 자체적으로 표시 할 수 없다. 대신 루트 뷰 컨트롤러와 쌍을 이룬다.



- 루트 뷰 컨트롤러는 네비게이션 컨트롤러의 스택에 있는 첫 번째 뷰 컨트롤러를 나타내며 루트 뷰 컨트롤러의 콘텐츠 뷰 계층 구조는 원도우에 로드 할 첫 번째 콘텐츠 뷰 계층 구조이다. 전체 애플리케이션을 네비게이션 컨트롤러의 스택에 넣으려면 Sourceless Segue를 네비게이션 컨트롤러로 옮기고 멀티화면 앱에서와 같이 첫 번째 화면의 View Controller를 루트보기 컨트롤러(Root View Controller)로 설정할 수 있다.



4. Xamarin.Forms

4.1 Xamarin.Forms Requirements

- Xamarin.Forms는 iOS, Android, Windows Phone, Windows Store 및 Universal Windows Platform 응용 프로그램에서 공유 할 수 있는 기본 사용자 인터페이스 레이아웃을 효율적으로 만들 수 있는 크로스 플랫폼 UI 도구이다.
 - Xamarin.Forms는 코드 공유를 극대화하고 Xamarin.iOS 및 Xamarin.Android는 플랫폼 별 API에 대한 직접 액세스를 제공한다.
 - **Target Platform** : Xamarin.Forms 응용프로그램은 다음과 같은 OS에 사용가능 하다.
 - iOS 8 or higher
 - Android 4.0.3 (API 15) or higher (more details)
 - Windows 10 Universal Windows Platform (more details)
 - Windows 8.1 / Windows Phone 8.1 WinRT (more details)
 - Windows Phone 8 Silverlight (DEPRECATED)
 - **Android** : 최신 Android SDK 도구 및 Android API 플랫폼을 설치해야 Android 프로젝트의 대상 / 컴파일 버전은 최신 설치된 플랫폼 사용으로 설정해야 한다. 그러나 최소 버전은 API 15로 설정 될 수 있으므로 Android 4.0.3 이상을 사용하는 기기를 계속 지원할 수 있다.
 - **개발 시스템 요구사항**
 - **Mac 시스템** : Mac 용 Visual Studio를 사용하여 OS X El Capitan (10.11) 이상에서 Xamarin.Forms 응용 프로그램을 개발할 수 있으며 iOS 앱을 개발하기 위해 적어도 iOS 10 SDK와 Xcode 8이 설치되어 있어야 한다.
 - **윈도우 시스템** : iOS 및 Android 용 Xamarin.Forms 응용 프로그램은 Xamarin 개발을 지원하는 모든 Windows에서 구축 할 수 있는데 이를 위해서는 Visual Studio 2013 Update 2 이상이 필요하며 Windows 7 이상을 실행해야 한다. iOS 개발에는 네트워크화 된 Mac이 필요하다.
 - 비주얼 스튜디오에서 Xamarin.Forms 템플릿은 3개의 Windows 템플릿이 사용가능하다.
 - Windows 8.1 - 태블릿, 데스크탑(WinRT 기반, WinRT는 Windows Runtime의 약자로 모바일과 웹서비스를 주로 겨냥한 MS의 차세대 API)
 - Windows Phone 8.1
 - Universal Windows Platform Apps - Xamarin.Forms에서는 윈도우10의 Universal (UWP) 응용프로그램을 폰, 태블릿, 데스크탑 디바이스에서 실행할 수 있도록 지원한다.
- Windows** : Windows8.1 프로젝트는 사전에 기본 템플렛에 포함되어 있지 않으므로 별도로 포함해야 한다.
- Universal Windows Platform(UWP)** : Windows 10 UWP 프로젝트 역시 사전에 포함되어 있지 않으므로 별도로 추가해야 한다. 참고로 타일 UI 앱의 명칭은 다음과 같이 변해왔다.
Metro App -> Modern App -> Windows Store App -> Universal App Platform(Windows 10)

Windows Phone 8(Silverlight) : Xamarin.Forms 1.x and 2.x는 Windows Phone 8 Silverlight 응용프로그램을 지원했지만 Deprecated 됨.

- Universal Windows Platform System 개발을 위해선 비주얼 스튜디오 2015 및 Windows10 이 필수요건 이다.
- 자마린의 Xamarin.Forms는 크로스플랫폼(Android, iOS, Windows, and Windows Phone)을 편하게 개발하도록 지원하는 프레임워크로 사용자가 작성한 응용프로그램은 자마린을 통해 타겟 플랫폼의 Native Control로 렌더링되어 실행된다.
- 스크린 화면은 Xamarin.Forms의 Page에 대응되는데 안드로이드의 Activity, iOS의 View Controller, Windows Universal Platform의 Page와 같다.
- Xamarin.Forms 프로젝트 생성 구조

Phoneword - 모든 공유 코드와 공유 UI를 포함하는 이식 가능한 클래스 라이브러리 (PCL) 프로젝트.

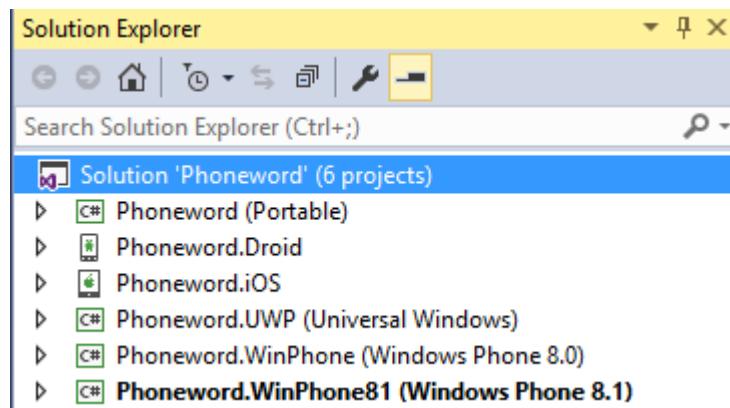
Phoneword.Android - 안드로이드 특정 코드를 보유하고 있으며 안드로이드 응용 프로그램의 진입 점.

Phoneword.iOS - iOS 특정 코드를 보유하고 있으며 iOS 애플리케이션의 시작점.

Phoneword.UWP - UWP (Universal Windows Platform) 전용 코드를 포함하며 UWP 응용 프로그램의 진입점.

Phoneword.WinPhone - Windows Phone 전용 코드를 포함하며 Windows Phone 8.0 응용 프로그램의 시작점.

Phoneword.WinPhone81 - Windows Phone 8.1 특정 코드를 포함하며 Windows Phone 8.1 응용 프로그램의 시작점.



4.2 Xamarin.Forms Quick Start

이전에 작성한 HelloWorld 예제를 Xamarin.Forms로 변경해서 실습해 보자.

<https://developer.xamarin.com/guides/xamarin-forms/getting-started/hello-xamarin-forms/quickstart/>

4.3 Xamarin.Forms HelloWord 분석

4.3.1 Xamarin.Forms HelloWord 프로젝트 구조

- Xamarin.Forms 프로젝트는 참조(Reference), Properties 두개의 폴더로 이루어 진다.

참조(Reference) - 응용 프로그램을 작성하고 실행하는 데 필요한 어셈블리가 들어 있다.

Properties - .NET 어셈블리 메타 데이터 파일인 AssemblyInfo.cs를 포함하며 이 파일을 응용 프로그램에 대한 몇 가지 기본 정보(버전, CopyRight 등)로 채우는 것이 좋다.

- HelloWorld를 구성하는 파일들

App.xaml - 응용 프로그램에 대한 리소스 사전을 정의하는 App 클래스의 XAML 태그입니다.

App.xaml.cs - 각 플랫폼에서 응용 프로그램에 의해 표시되는 첫 번째 페이지를 인스턴스화하고 응용 프로그램 생명주기 이벤트(콜백 메소드)를 처리하는 App 클래스의 코드비하인드.

IDialer.cs - Dialer이라는 추상메소드를 하나 가지며 각 플랫폼에서 이 인터페이스를 구현하여 전화를 걸게 된다.

MainPage.xaml - 응용 프로그램이 시작될 때 표시되는 페이지의 UI를 정의하는 MainPage 클래스의 XAML 태그입니다.

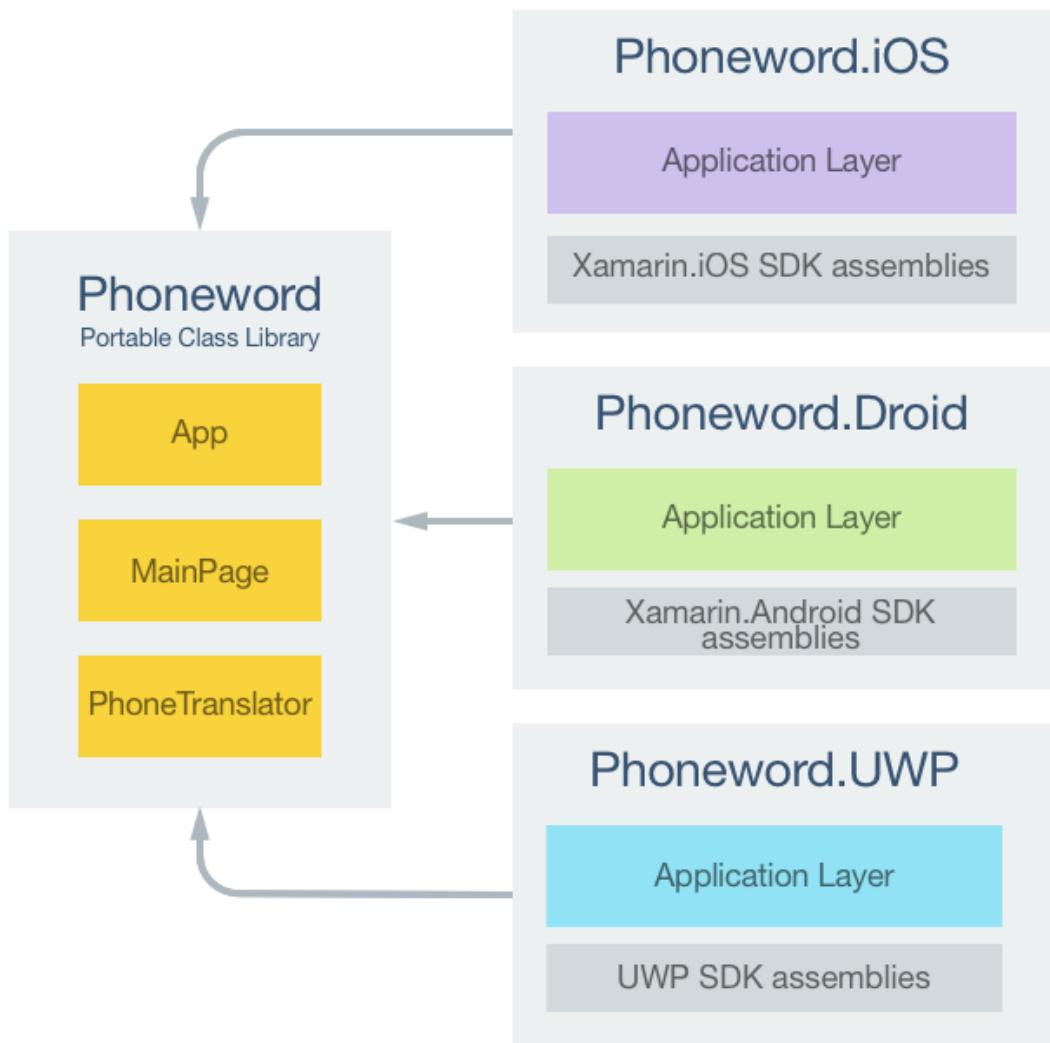
MainPage.xaml.cs - 사용자가 페이지와 상호 작용할 때 실행되는 비즈니스 로직을 포함하는 MainPage 클래스의 코드 비하인드 파일.

packages.config - 필수 패키지와 해당 버전을 추적하기 위해 프로젝트에서 사용중인 NuGet 패키지에 대한 정보가 들어있는 XML 파일로 소스 코드를 다른 사용자와 공유 할 때 누락 된 NuGet 패키지를 자동으로 복원하도록 Xamarin Studio와 Visual Studio를 모두 구성 할 수 있다. 이 파일의 내용은 NuGet 패키지 관리자가 제어하므로 수동으로 편집해서는 안된다.

PhoneTranslator.cs - 전화 단어를 전화 번호로 변환하는 클래스, MainPage.xaml.cs에서 호출된다.

4.3.2 Xamarin.Forms HelloWord Fundamentals

- Xamarin.Forms 응용 프로그램은 전통적인 크로스 플랫폼 응용 프로그램과 같은 방식으로 구성되는데 전체 플랫폼에서 공유할 코드는 PCL (Portable Class Library)에 배치된다.
- 아래 다이어그램은 Phoneword 응용 프로그램의 구조이다.



4.3.3 Xamarin.Forms HelloWord PCL 및 플랫폼별 코드 분석

- 시작 코드 재사용을 극대화하기 위해 Xamarin.Forms 응용 프로그램에는 아래 코드와 같이 각 **플랫폼의 응용 프로그램에서 표시 할 첫 번째 페이지를 인스턴스화 하는 App**이라는 단일 클래스가 있다.

[PCL : App.xaml.cs]

```
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

[assembly: XamlCompilation(XamlCompilationOptions.Compile)]
namespace Phoneword
{
    public partial class App : Application
    {
```

```

public App()
{
    InitializeComponent();
    MainPage = new MainPage();
}
...
}

```

■ iOS 플랫폼을 위한 응용프로그램 시작코드

iOS에서 초기 Xamarin.Forms 페이지를 시작하기 위해 Phoneword.iOS 프로젝트에는 다음 코드 예문과 같이 **FormsApplicationDelegate** 클래스에서 상속 한 **AppDelegate** 클래스가 포함되어 있다.

```

namespace Phoneword.iOS
{
    [Register ("AppDelegate")]
    public partial class AppDelegate : global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
    {
        public override bool FinishedLaunching (UIApplication app, NSDictionary options)
        {
            global::Xamarin.Forms.Forms.Init ();
            LoadApplication (new App ());
            return base.FinishedLaunching (app, options);
        }
    }
}

```

FinishedLaunching 재정의는 Init 메서드를 호출하여 Xamarin.Forms 프레임 워크를 초기화 하고 LoadApplication 메서드를 호출하여 최상단 View Controller를 설정하기 전에 Xamarin.Forms의 iOS 관련 구현이 응용 프로그램에 로드된다.

■ Android 플랫폼을 위한 응용프로그램 시작코드

Android쪽 Xamarin.Forms 페이지를 시작하기 위해 **FormsApplicationActivity** 클래스에서 상속 한 Activity와 함께 MainLauncher 특성을 사용하여 Activity를 만드는 코드가 포함된다.

```

namespace Phoneword.Android
{
    [Activity (Label = "Phoneword.Android",
              Icon = "@drawable/icon",
              MainLauncher = true,
              ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
    public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
    {
        protected override void OnCreate (Bundle bundle)
        {
            base.OnCreate (bundle);
            global::Xamarin.Forms.Init (this, bundle);
            LoadApplication (new App ());
        }
    }
}

```

4.3.4 Xamarin.Forms HelloWorld 사용자 인터페이스

- Xamarin.Forms 응용 프로그램의 사용자 인터페이스를 만드는 데 사용되는 네 개의 주요 컨트롤 그룹이 있다.

- | |
|--|
| ■ Page - 크로스 플랫폼 모바일 응용 프로그램 화면을 나타내며 Phoneword 응용 프로그램은 ContentPage 클래스를 사용하여 단일 화면을 표시한다. |
| ■ Layouts - 뷰를 논리 구조로 구성하는 데 사용되는 컨테이너로 Phoneword 애플리케이션은 StackLayout 클래스를 사용하여 컨트롤을 수평 스택에 정렬했다 |
| ■ Views - Label, Button 및 Text Entry Box와 같이 사용자 인터페이스에 표시되는 컨트롤로 Phoneword 응용 프로그램은 Label, Entry 및 Button 컨트롤을 사용한다. |
| ■ Cells - 셀은 목록의 항목에 사용되는 특수 요소이며 목록의 각 항목을 그리는 방법을 설명한다. Phoneword 응용 프로그램은 모든 셀을 사용하지 않도록 구성되었다. |

- 런타임시 각 컨트롤은 렌더링 될 원시 네이티브 객체에 매핑된다.
- Phoneword 응용 프로그램이 모든 플랫폼에서 실행되면 Xamarin.Forms의 Page에 해당하는 단일 화면을 표시하며 Page는 Android의 ViewGroup, iOS의 View Controller 또는 Universal Windows Platform의 Page를 나타낸다.

- Phoneword 응용 프로그램은 또한 아래 코드 예에서 XAML 태그가 표시된 MainPage 클래스를 나타내는 ContentPage 객체를 인스턴스화 한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Phoneword.MainPage">
    ...
    <ContentPage.Content>
        <StackLayout VerticalOptions="FillAndExpand"
            HorizontalOptions="FillAndExpand"
            Orientation="Vertical"
            Spacing="15">
            <Label Text="Enter a Phoneword:" />
            <Entry x:Name="phoneNumberText" Text="1-855-XAMARIN" />
            <Button x:Name="translateButton" Text="Translate" Clicked="OnTranslate" />
            <Button x:Name="callButton" Text="Call" IsEnabled="false" Clicked="OnCall" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

- MainPage 클래스는 StackLayout 컨트롤을 사용하여 화면 크기에 관계없이 화면에 컨트롤을 자동으로 정렬하는데 각 하위 요소는 추가 된 순서대로 수직으로 배치된다.
- StackLayout에 사용되는 화면 공간의 양은 HorizontalOptions 및 VerticalOptions 속성 값에 따라 다르며 StackLayout 컨트롤에는 페이지에 텍스트를 표시하는 Label 컨트롤, 텍스트 사용자 입력을 허용하는 Entry 컨트롤 및 터치 이벤트에 대한 응답으로 코드를 실행하는 데 사용되는 Button 컨트롤이 있다.

4.3.5 Xamarin.Forms HelloWorld User Interaction

- **Page** - 크로스 플랫폼 모바일 응용 프로그램 화면을 나타내며 Phoneword 응용 프로그램의 ContentPage MainPage 클래스는 StackLayout 컨트롤을 사용하여 화면 크기에 관계없이 화면에 컨트롤을 자동으로 정렬하는데 각 XAML에 정의 된 개체는 코드비하인드 파일에서 처리되는 이벤트를 발생시킬 수 있다. MainPage 클래스의 코드비하인드의 OnTranslate 메서드를 보여주는데 이 메서드는 Translate 버튼에서 Clicked 이벤트가 발생하면 이에 따라 실행된다.

- [MainPage.xaml]

```
<Button x:Name="translateButton" Text="Translate" Clicked="OnTranslate" />
```

■ [MainPage.xaml.cs]

```
void OnTranslate(object sender, EventArgs e)
{
    translatedNumber = Core.PhonewordTranslator.ToNumber(phoneNumberText.Text);
    if (!string.IsNullOrWhiteSpace(translatedNumber)) {
        callButton.IsEnabled = true;
        callButton.Text = "Call " + translatedNumber;
    } else {
        callButton.IsEnabled = false;
        callButton.Text = "Call";
    }
}
```

- OnTranslate 메서드는 phoneword를 해당 전화 번호로 변환하고 이에 대한 응답으로 호출 Button의 isEnabled 속성을 설정한다.
- XAML 클래스의 코드 숨김 파일은 x : Name 특성과 함께 지정된 이름을 사용하여 XAML에 정의 된 개체에 액세스 할 수 있다. 이 특성에 할당 된 값은 문자 또는 밑줄로 시작하고 포함 된 공백을 포함하지 않아야하므로 C # 변수와 동일한 규칙을 갖는다.

4.3.6 Xamarin.Forms HelloWorld 추가적인 개념

- Alert Dialog 표시 : 사용자가 CALL 버튼을 누르면 Phoneword 응용 프로그램은 전화를 걸거나 취소 할 수 있는 경고 대화 상자를 표시하고 DisplayAlert 메서드는 아래 예제와 같이 대화 상자를 만드는 데 사용된다.

```
await this.DisplayAlert (
    "Dial a Number",
    "Would you like to call " + translatedNumber + "?",
    "Yes",
    "No");
```

- **DependencyService** 클래스를 통해 네이티브 기능에 액세스하며 **IDialer** 인터페이스를 통해 플랫폼 별 전화 다이얼링 기능을 구현한다.

[MainPage.xaml.cs]

```

async void OnCall (object sender, EventArgs e)
{
    ...
    var dialer = DependencyService.Get<IDialer> ();
    ...
}

```

- URL로 전화 걸기 : Phoneword 응용 프로그램은 OpenURL을 사용하여 시스템 전화걸기 앱을 시작하고 URL은 iOS 프로젝트의 다음 코드 예제에서와 같이 호출 할 전화 번호 앞에 tel : 접두사가 온다.

```

Xamarin.iOS : [PhoneDialer.cs]
using Foundation;
using Phoneword.iOS;
using UIKit;
using Xamarin.Forms;

[assembly: Dependency(typeof(PhoneDialer))]
namespace Phoneword.iOS
{
    public class PhoneDialer : IDialer
    {
        public bool Dial(string number)
        {
            return UIApplication.SharedApplication.OpenUrl(
                new NSUrl("tel:" + number));
        }
    }
}

```

- 플랫폼별 레이아웃 조정 : Device 클래스를 사용하면 개발자는 iOS 플랫폼에서 다른 Padding 값을 사용하여 각 페이지를 올바르게 표시한다. 다음 코드 예제와 같이 플랫폼별로 응용 프로그램 레이아웃과 기능을 사용자 지정할 수 있다.

```

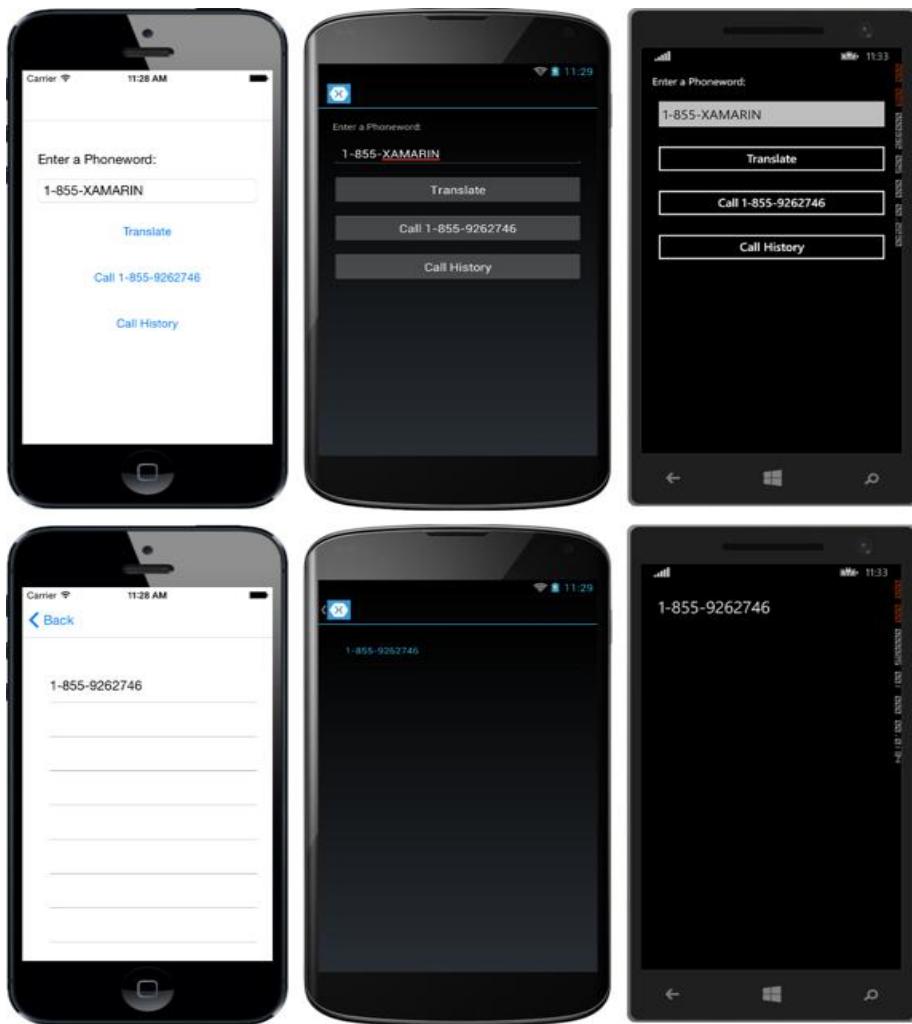
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    ...
    <ContentPage.Padding>
        <OnPlatform x:TypeArguments="Thickness"
            iOS="20, 40, 20, 20"

```

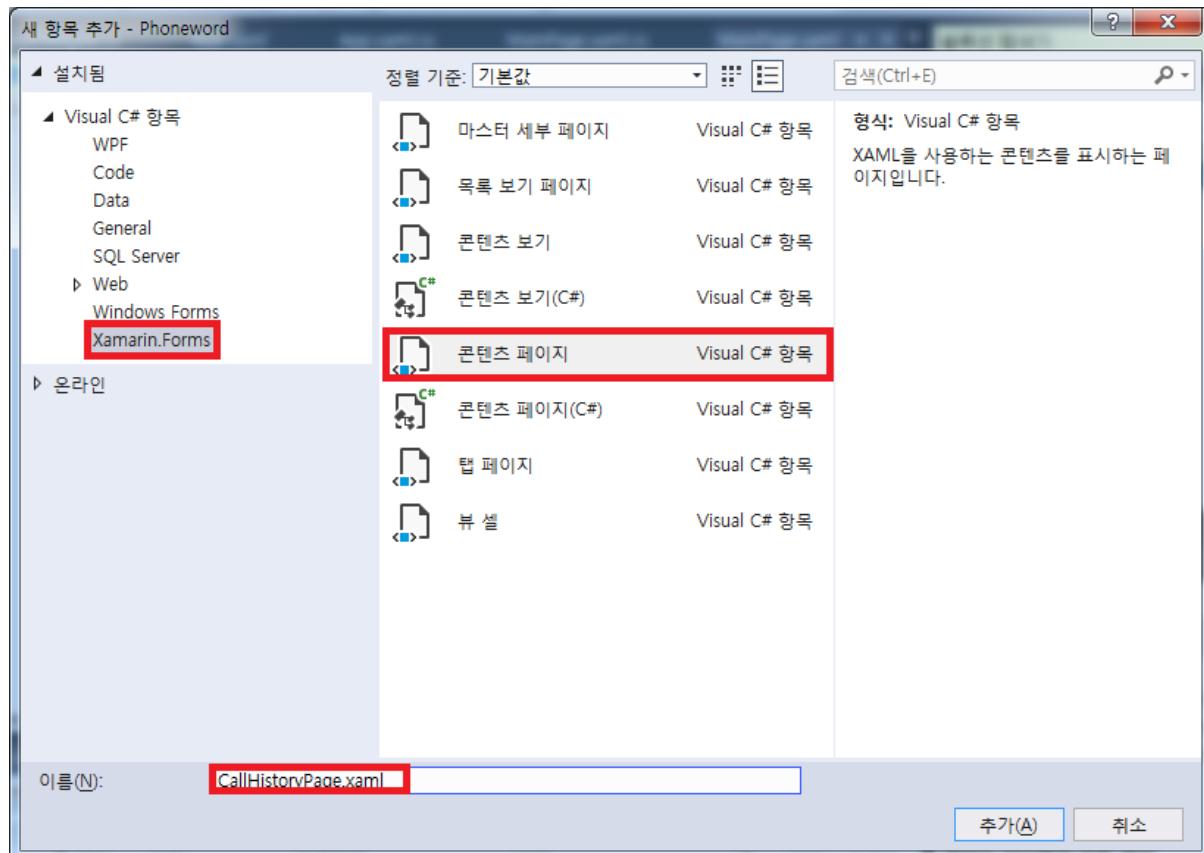
```
... />  
</ContentPage.Padding>  
...  
</ContentPage>
```

4.4 Xamarin.Forms Multiscreen Quick Start Example

- 이전의 Xamarin.Forms Quick Start Hello 예제에 Call History를 저장하기 위한 두 번째 화면이 추가되었다.



- 비주얼 스튜디오의 최근 솔루션 목록에서 이전에 작성한 Phoneword를 오픈하자.
- phoneword(portable) 프로젝트에서 추가 -> 새 항목 -> Visual C# -> Xamarin.Forms -> 컨텐츠 페이지를 선택하고 이름은 **CallHistoryPage** 라고 하자. (이 파일은 페이지의 사용자 인터페이스를 정의한다)



```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:local="clr-namespace:Phoneword;assembly=Phoneword"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Phoneword.CallHistoryPage">
<ContentPage.Padding>
    <OnPlatform x:TypeArguments="Thickness"
        iOS="20, 40, 20, 20"
        Android="20, 20, 20, 20"
        WinPhone="20, 20, 20, 20" />
</ContentPage.Padding>
<ContentPage.Content>
    <StackLayout VerticalOptions="FillAndExpand"
        HorizontalOptions="FillAndExpand"
        Orientation="Vertical"
        Spacing="15">
        <ListView ItemsSource="{x:Static local:App.PhoneNumbers}" />
    </StackLayout>

```

```
</ContentPage.Content>  
</ContentPage>
```

- Phoneword(portable) 프로젝트에서 App.xaml.cs를 오픈 후 System.Collections.Generic 네임스페이스를 import하고 PhoneNumbers라는 속성을 추가후 생성자에서 초기화하고 NavigationPage를 통해 MainPage 속성을 초기화 시키자.

```
using System.Collections.Generic;  
using Xamarin.Forms;  
using Xamarin.Forms.Xaml;  
  
[assembly: XamlCompilation(XamlCompilationOptions.Compile)]  
namespace Phoneword  
{  
    public partial class App : Application  
    {  
        public static IList<string> PhoneNumbers { get; set; }  
  
        public App()  
        {  
            InitializeComponent();  
            PhoneNumbers = new List<string>();  
            MainPage = new NavigationPage(new MainPage());  
        }  
        ...  
    }  
}
```

- phoneword(portable) 프로젝트에서 MainPage.xaml을 오픈하고 CallHistory Button 컨트롤을 StackLayout의 마지막에 추가하자.

```
<?xml version="1.0" encoding="UTF-8"?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
             x:Class="Phoneword.MainPage">  
    <ContentPage.Padding>
```

```

<OnPlatform x:TypeArguments="Thickness"
    iOS="20, 40, 20, 20"
    Android="20, 20, 20, 20"
    WinPhone="20, 20, 20, 20" />
</ContentPage.Padding>
<ContentPage.Content>
    <StackLayout VerticalOptions="FillAndExpand"
        HorizontalOptions="FillAndExpand"
        Orientation="Vertical"
        Spacing="15">
        <Label Text="Enter a Phoneword:" />
        <Entry x:Name="phoneNumberText" Text="1-855-XAMARIN" />
        <Button x:Name="translateButton" Text="Translate" Clicked="OnTranslate" />
        <Button x:Name="callButton" Text="Call" IsEnabled="false" Clicked="OnCall" />
        <Button x:Name="callHistoryButton" Text="Call History" IsEnabled="false"
            Clicked="OnCallHistory" />
    </StackLayout>
</ContentPage.Content>
</ContentPage>

```

- phoneword(portable) 프로젝트에서 MainPage.xaml.cs 파일을 오픈하고 위에서 정의한 OnCallHistory 이벤트 핸들러를 작성하고 OnCall 이벤트 핸들러에서 App.PhoneNumbers 컬렉션에 호출한 전화번호를 저장, callHistoryButton을 활성화하는 로직을 추가하자.

```

using System;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace Phoneword
{
    public partial class MainPage : ContentPage
    {
        string translatedNumber;

        public MainPage()
        {
            InitializeComponent();
        }
    }
}

```

```

void OnTranslate(object sender, EventArgs e)
{
    translatedNumber
Core.PhonewordTranslator.ToNumber(phoneNumberText.Text);
    if (!string.IsNullOrWhiteSpace(translatedNumber))
    {
        callButton.IsEnabled = true;
        callButton.Text = "Call " + translatedNumber;
    }
    else
    {
        callButton.IsEnabled = false;
        callButton.Text = "Call";
    }
}

async void OnCall(object sender, EventArgs e)
{

    if (await this.DisplayAlert(
        "Dial a Number",
        "Would you like to call " + translatedNumber + "?",
        "Yes",
        "No"))
    {
        var dialer = DependencyService.Get<IDialer>();

        if (dialer != null)
        {
            App.PhoneNumbers.Add(translatedNumber);
            callHistoryButton.IsEnabled = true;
            dialer.Dial(translatedNumber);
        }
    }
}

async void OnCallHistory(object sender, EventArgs e)
{
    await Navigation.PushAsync(new CallHistoryPage());
}

```

```
    }  
}  
}
```

플랫폼별로 디플로이 하고 실행해 보자.

4.5 Views And Layout

- Xamarin.Forms는 사용자 인터페이스를 위한 4개의 메인 그룹이 있다.
 1. **Page** : 크로스플랫폼 모바일 응용프로그램 화면을 위한 것
 2. **Layouts** : 구성된 뷰를 논리적인 구조안으로 담기 위한 것
 3. **Views** : 라벨, 버튼, 텍스트 엔트리 박스처럼 사용자 인터페이스위에 Display되는 컨트롤
 4. **Cells** : 리스트안의 아이템을 위한 특별한 요소로 리스트안에 그려질 아이템을 나타낸다.

4.5.1 Stack Layout

스택처럼 겹겹히 쌓아올려 만드는 구조, 스크린의 사이즈를 무시하고 자동으로 컨트롤을 쌓아서 정돈하는 모양으로 각 자식 요소들은 수직또는 수평으로 뒤쪽에 채워지는데 기본은 수직이다. 아래는 StackLayout을 위한 XAML이다.

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
             x:Class="HelloXamarinFormsWorldXaml.StackLayoutExample1" Padding="20">  
    <StackLayout Spacing="10"> <!--기본은 수직 레이아웃 →  
        <Label Text="Stop" BackgroundColor="Red" Font="20" />  
        <Label Text="Slow down" BackgroundColor="Yellow" Font="20" />  
        <Label Text="Go" BackgroundColor="Green" Font="20" />  
    </StackLayout>  
</ContentPage>
```

C#코드로 작성하면 다음과 같다.

```
public class StackLayoutExample : ContentPage  
{  
    public StackLayoutExample()  
    {
```

```

Padding = new Thickness(20);
var red = new Label
{
    Text = "Stop", BackgroundColor = Color.Red, FontSize = 20
};
var yellow = new Label
{
    Text = "Slow down", BackgroundColor = Color.Yellow, FontSize = 20
};
var green = new Label
{
    Text = "Go", BackgroundColor = Color.Green, FontSize = 20
};

Content = new StackLayout //기본은 수직레이아웃
{
    Spacing = 10,
    Children = { red, yellow, green }
};
}
}

```



수평레이아웃으로 바꾸려면 아래처럼 하면 된다.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage           xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="HelloXamarinFormsWorldXaml.StackLayoutExample2" Padding="20">
    <StackLayout      Spacing="10"      VerticalOptions="End"      Orientation="Horizontal"
    HorizontalOptions="Start">
        <Label Text="Stop" BackgroundColor="Red" Font="20" />
        <Label Text="Slow down" BackgroundColor="Yellow" Font="20" />
        <Label Text="Go" BackgroundColor="Green" Font="20" />
    </StackLayout>
</ContentPage>
```

동일한 C#코드

```
Content = new StackLayout
{
    Spacing = 10,
    VerticalOptions = LayoutOptions.End,
    Orientation = StackOrientation.Horizontal,
    HorizontalOptions = LayoutOptions.Start,
    Children = { red, yellow, green }
};
```



4.5.2 Lists in Xamarin.Forms

- ListView 컨트롤은 화면에 아이템의 컬렉션을 담기 위한 컨트롤이다.
- 각 ListView는 하나의 셀안에 포함되며 하나의 ListView는 TextCell 템플릿 안에 내장되고 한 라인의 텍스트로 렌더링 된다.
- ListView 예제

```
var listView = new ListView
{
    RowHeight = 40
};
listView.ItemsSource = new string []
{
    "Buy pears", "Buy oranges", "Buy mangos", "Buy apples", "Buy bananas"
};
Content = new StackLayout
{
    VerticalOptions = LayoutOptions.FillAndExpand,
    Children = { listView }
};
```

Buy pears
Buy oranges
Buy mangos
Buy apples
Buy bananas

4.5.3 ListView Data Sources

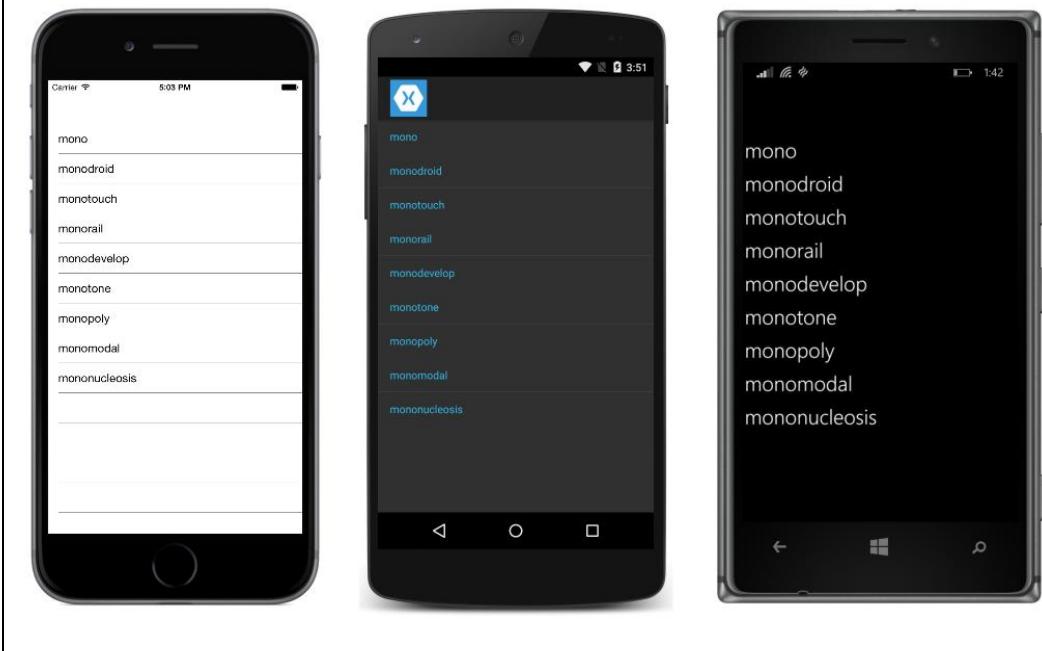
- ListView는 데이터를 리스트업 하기 위해 사용되는 컨트롤이다.
- ItemSource

ListView는 ItemSource 속성에 의해 채울 데이터를 정의하는데 IEnumerable을 구현한 어떤것이라도 ItemSource에 포함될 수 있다. 아래는 문자열 배열을 이용한 예문이다.

```
var listView = new ListView();
listView.ItemsSource = new string[]{
    "mono",
    "monodroid",
    "monotouch",
    "monorail",
    "monodevelop",
    "monotone",
    "monopoly",
    "monomodal",
    "mononucleosis"
};

//monochrome will not appear in the list because it was added
//after the list was populated.

listView.ItemsSource.Add("monochrome");
```



4.5.4 Selecting an Item in a ListView

- ListView의 셀에 사용자의 터치에 반응하기 위해서는 **ItemSelected** 이벤트 핸들러를 이용한

다.

```
listView.ItemSelected += async (sender, e) => {
    await DisplayAlert("Tapped!", e.SelectedItem + " was tapped.", "OK");
};
```

- 다른 페이지로 보낼때는 ItemSelected 이벤트안에 **PushAsync** 메소드를 사용하면 되는데 ItemSelected 이벤트는 SelectedItem 속성과 연계되며 PushAsync 메소드를 통해 새로운 페이지로 연결된다.

```
listView.ItemSelected += async (sender, e) => {
    var todoItem = (TodoItem)e.SelectedItem;
    var todoPage = new TodoItemPage(todoItem); // so the new page shows correct data
    await Navigation.PushAsync(todoPage);
};
```

4.5.5 DataTemplateSelector

- ListView등에 데이터를 나열할 때 데이터의 성격에 따라 다른 형태로 보여주는 경우 여러 템플릿에서 원하는 템플릿을 선택하는 역할을 한다. **OnSelectTemplate** 메소드의 인자로 넘어오는 Item의 데이터 타입 및 데이터 속성, Container에 의해 DataTemplate을 선택한다.

```
public class PersonDataTemplateSelector : DataTemplateSelector {
    public DataTemplate ValidTemplate { get; set; }
    public DataTemplate InvalidTemplate { get; set; }

    protected override DataTemplate OnSelectTemplate(object item, BindableObject container) {
        return ((Person)item).DateOfBirth.Year >= 1980 ? ValidTemplate : InvalidTemplate;
    }
}
```

- DataTemplateSelector.OnSelectTemplate 메소드는 Data Type과 부모 컨테이너에서 DataTemplate을 선택하는 용도로 사용한다. 생성자 또는 속성을 통해 DataTemplate을 만들거나 저장하며 OnSelectTemplate 메소드에서 작성된 템플릿중 적절히 리턴을 하면 된다.
- Android의 ListView는 템플릿을 20개로 제한되어 있다.

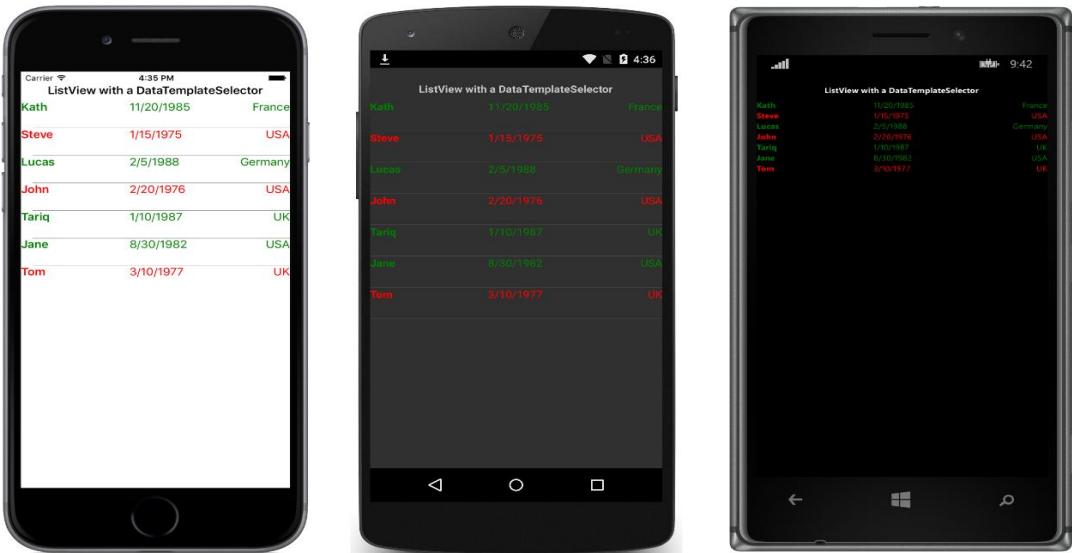
```
class MyDataTemplateSelector : DataTemplateSelector
```

```
{  
    private readonly DataTemplate templateOne;  
    private readonly DataTemplate templateTwo;  
  
    public MyDataTemplateSelector ()  
    {  
        // Retain instances  
        this.templateOne = new DataTemplate (typeof (ViewA));  
        this.templateTwo = new DataTemplate (typeof (ViewB));  
    }  
  
    protected override DataTemplate OnSelectTemplate (object item, BindableObject container)  
    {  
        if (item is double)  
            return this.templateTwo;  
        else  
            return this.templateOne;  
    }  
}
```

4.5.6 ListView, DataTemplateSelector Example

- 런타임중에 ListView안에 아이템을 담는 예제, DataTemplate을 사용하기 위해 DataTemplateSelector를 사용한 예제

<https://developer.xamarin.com/samples/xamarin-forms/Templates/DataTemplateSelector/>



- Xamarin.Forms 프로젝트명 : Selector(Xamarin Cross Platform 프로젝트, blank xaml app portable)

맨위의 Portable 프로젝트(Selector)에 아래 파일들을 작성하자.

■ [MainPage.xaml]

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:Selector;assembly=Selector"
    x:Class="Selector.MainPage">

<ContentPage.Resources>
    <ResourceDictionary>
        <DataTemplate x:Key="validPersonTemplate">
            <ViewCell>
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="0.4*" />
                        <ColumnDefinition Width="0.3*" />
                        <ColumnDefinition Width="0.3*" />
                    </Grid.ColumnDefinitions>
                    <Label Text="{Binding Name}" />
                    <Label Grid.Column="1" Text="{Binding TextColor, FontAttributes=Bold}" />
                    <Label Grid.Column="1" Text="{Binding DateOfBirth, StringFormat='{0:d}'}" TextColor="Green" />
                </Grid>
            </ViewCell>
        </DataTemplate>
    </ResourceDictionary>
</ContentPage.Resources>

```

```

        <Label Grid.Column="2" Text="{Binding
Location}" TextColor="Green" HorizontalTextAlignment="End" />
    </Grid>
</ViewCell>
</DataTemplate>
<DataTemplate x:Key="invalidPersonTemplate">
    <ViewCell>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="0.4*" />
                <ColumnDefinition Width="0.3*" />
                <ColumnDefinition Width="0.3*" />
            </Grid.ColumnDefinitions>
            <Label Text="{Binding
Name}" TextColor="Red" FontAttributes="Bold" />
            <Label Grid.Column="1" Text="{Binding
DateOfBirth, StringFormat='{0:d}'}" TextColor="Red" />
            <Label Grid.Column="2" Text="{Binding
Location}" TextColor="Red" HorizontalTextAlignment="End" />
        </Grid>
    </ViewCell>
</DataTemplate>
<local:PersonDataTemplateSelector>
x:Key="personDataTemplateSelector" ValidTemplate="{StaticResource validPersonTemplate}"
InvalidTemplate="{StaticResource invalidPersonTemplate}" />
</ResourceDictionary>
</ContentPage.Resources>
<StackLayout Padding="0,20,0,0">
    <Label Text="ListView with a DataTemplateSelector" FontAttributes="Bold"
HorizontalOptions="Center" />
    <ListView x:Name="listView" ItemTemplate="{StaticResource
personDataTemplateSelector}" />
</StackLayout>
</ContentPage>

```

■ MainPage.xaml.cs

```

using System;
using System.Collections.Generic;
using Xamarin.Forms;

```

```

namespace Selector
{
    public partial class MainPage : ContentPage
    {
        public MainPage ()
        {
            InitializeComponent ();

            var people = new List<Person> {
                new Person ("Kath", new DateTime (1985, 11, 20), "France"),
                new Person ("Steve", new DateTime (1975, 1, 15), "USA"),
                new Person ("Lucas", new DateTime (1988, 2, 5),
"Germany"),
                new Person ("John", new DateTime (1976, 2, 20), "USA"),
                new Person ("Tariq", new DateTime (1987, 1, 10), "UK"),
                new Person ("Jane", new DateTime (1982, 8, 30), "USA"),
                new Person ("Tom", new DateTime (1977, 3, 10), "UK")
            };

            listView.ItemsSource = people;
        }
    }
}

```

■ [Person.cs]

```

using System;

namespace Selector
{
    public class Person
    {
        public string Name { get; private set; }
        public DateTime DateOfBirth { get; private set; }
        public string Location { get; private set; }

        public Person (string name, DateTime dob, string location)
        {
            Name = name;
            DateOfBirth = dob;
        }
    }
}

```

```
        Location = location;
    }
}
```

■ [PersonDataTemplateSelector.cs]

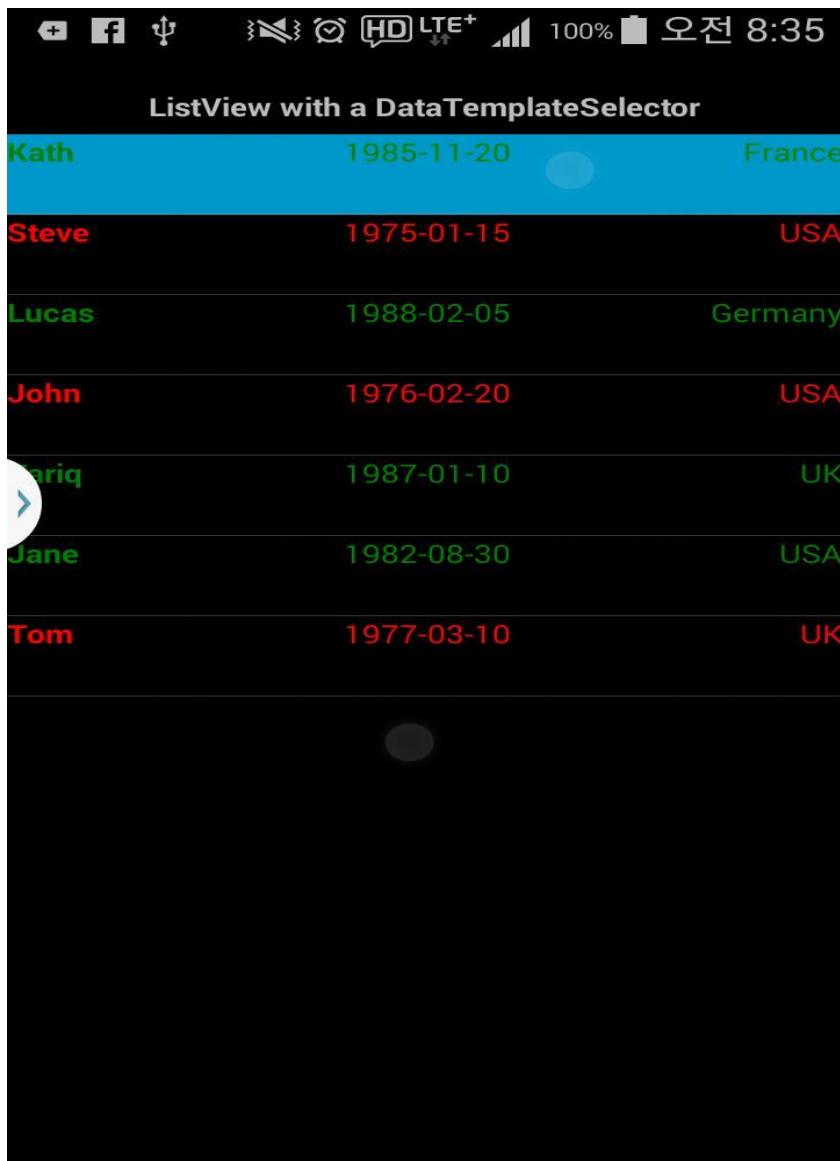
```
using Xamarin.Forms;

namespace Selector
{
    public class PersonDataTemplateSelector : DataTemplateSelector
    {
        public DataTemplate ValidTemplate { get; set; }

        public DataTemplate InvalidTemplate { get; set; }

        protected override DataTemplate OnSelectTemplate (object item, BindableObject
container)
        {
            return ((Person)item).DateOfBirth.Year >= 1980 ? ValidTemplate :
InvalidTemplate;
        }
    }
}
```

■ 빌드 후 실행하자



항목을 선택했을 때 간단히 ALERT창을 띄우려면 다음과 같은 코드를 MainPage.xaml.cs 파일에 삽입하면 된다.

```
listView.ItemsSource = people;  
listView.ItemSelected += async (sender, e) => {  
    await DisplayAlert("Alert", "You have been alerted", "OK");  
};
```

물론 아래와 같이 선택한 아이템의 Name을 출력하는 것도 가능하다.

```
listView.ItemSelected += async (sender, e) => {
```

```

        await DisplayAlert("Alert",
            (e.SelectedItem as Person).Name + " selected!", "OK");
    }
}

```

4.6 MVVM개요(Model/View/ViewModel) 및 MVVM Example

- Model-View-ViewModel(MVVM)



Model : 비즈니스 로직과 데이터를 캡슐화 한것이다. 앱의 데이터를 캡슐화 하는 비 시각적 클래스 이다. 따라서 모델은 비즈니스 및 유효성 확인 로직과 함께 데이터 모델을 포함하는 앱의 도메인 모델을 나타내는 것으로, 모델 객체의 예로는 데이터 전송 개체 (DTO), 일반 올드 CLR 개체 (POCO) 및 생성 된 엔터티 및 프록시 개체가 있다.

모델 클래스는 일반적으로 데이터 액세스 및 캐싱을 캡슐화하는 서비스 또는 리포지토리와 함께 사용된다.

View : 사용자가 화면에서 보는 구조, 레이아웃 및 모양을 정의하는 역할을 한다. 각각의 뷰는 비즈니스 로직을 포함하지 않는 제한된 코드 숨김으로 XAML에 정의되는 것이 이상적이며, 경우에 따라 코드 숨김 파일에는 애니메이션과 같이 XAML에서 표현하기 어려운 시각적 동작을 구현하는 UI 코드가 포함될 수 있다. Xamarin.Forms 응용 프로그램에서 뷰는 일반적으로 Page 또는 ContentView 자식클래스로 뷰는 객체가 표시 될 때 이를 시각적으로 나타내는데 사용할 UI 요소를 지정하는 데이터 템플릿으로 표현할 수도 있다. 뷰로 사용되는 데이터 템플릿에는 코드 숨김이 없으며 특정 뷰 모델 유형에 바인딩 되도록 설계되었다.

코드 숨김에서 활성화 및 비활성화하지 않고 뷰 모델 속성에 바인딩하여 UI 요소를 사용하거나 사용하지 않도록 설정하는 것이 바람직하다.

뷰의 상호 작용 (예 : 버튼 클릭 또는 항목 선택)에 대한 응답으로 뷰 모델에서 코드를 실행하기위한 몇 가지 옵션이 있다. 컨트롤이 명령(Command)을 지원하면 컨트롤의 Command 속성을 뷰 모델의 ICommand 속성에 데이터 바인딩 할 수 있다. 컨트롤의 명령이 호출되면 뷰

모델의 코드가 실행되는 것이다. 명령 외에도 동작을 뷰의 개체에 연결할 수 있으며 호출 할 명령이나 발생시킬 이벤트를 수신 할 수 있다. 이에 대한 응답으로 동작은 뷰 모델에서 ICommand를 호출하거나 뷰 모델에서 메서드를 호출 할 수 있다.

ViewModel : View를 나타내주기 위한 Model이며 프리젠테이션 로직과 뷰를 위한 데이터를 캡슐화하고 있다. 뷰 모델은 뷰가 데이터를 바인딩 할 수 있는 속성 및 명령을 구현하고 변경 알림 이벤트를 통해 변경 사항을 뷰에 알린다.

뷰 모델에서 제공하는 속성 및 명령은 UI에서 제공하는 기능을 정의하지만 뷰는 해당 기능을 표시하는 방법을 결정한다. 뷰와 뷰가 요구하는 모델 클래스들 간의 상호작용을 조직할 책임이 있으며 뷰가 표시하는 데이터와 Command를 구현한 것이다. View보다는 Model과 유사하게 디자인 되며, View의 바인딩 될 때 가장 강력하다.

뷰 모델에서는 I / O 작업에 비동기 메서드를 사용하고 속성 변경사항을 뷰에 비동기 적으로 알리려면 이벤트를 발생시킨다. 뷰 모델은 뷰의 상호 작용에 필요한 모든 모델 클래스와 조화시키는 역할을 하며 일반적으로 뷰 모델과 모델 클래스 사이에는 일대 다 관계이다. 뷰 모델은 뷰의 컨트롤이 뷰에 직접 바인딩 할 수 있도록 모델 클래스를 뷰에 직접 표시하도록 선택할 수 있으며 이 경우 모델 클래스는 데이터 바인딩을 지원하고 알림 이벤트를 변경하도록 설계되어야 한다.

각 뷰 모델(ViewModel)은 뷔에서 쉽게 사용할 수 있는 양식으로 모델의 데이터를 제공하며 이를 수행하기 위해 때때로 데이터 변환을 수행하는데 이 데이터 변환을 뷔 모델에 배치하는 것은 뷔가 바인딩 할 수 있는 속성을 제공하기 때문에 좋은 방법이다. 예를 들어 뷔 모델은 두 속성의 값을 결합하여 뷔로 표시하기 쉽게 만들 수 있다.

데이터에 뷔 모델에서 제공하지 않는 특수한 형식이 필요할 때가 있는데 뷔 모델이 뷔와의 양방향 데이터 바인딩에 참여하려면 해당 속성이 PropertyChanged 이벤트를 발생시켜야 한다. 뷔 모델은 INotifyPropertyChanged 인터페이스를 구현하고 속성이 변경 될 때 PropertyChanged 이벤트를 발생시키면 된다.

컬렉션의 경우 ObservableCollection<T>이 제공되며 이 경우 컬렉션 변경(삽입 또는 삭제)을 알리기 위해 INotifyChangedChanged 인터페이스를 구현하지 않아도 된다.

- 뷔는 뷔 모델을 "인식"하고 뷔 모델은 모델을 알지만 모델은 뷔 모델을 인식하지 못하고 뷔 모델은 뷔를 인식하지 못한다. 따라서 뷔 모델은 뷔를 모델에서 분리하고 모델이 뷔와 독립적으로 전개되도록 한다.
- 뷔 모델은 모델 클래스의 어댑터 역할을 하므로 뷔의 변경에 따라 모델 코드를 크게 변경하지 않아도 된다.
- 뷔가 XAML로 완전히 구현 된 경우 코드를 건드리지 않고 앱 UI를 다시 디자인 할 수 있다.
- Model-View-ViewModel(MVVM) 패턴에 기반하여 XAML을 탄생 시켰으며 기본 패턴은 View 와 Model 이다(즉 ViewModel). XAML은 사용자 인터페이스와 (*.xaml) 기본데이터(Model)와

의 연결은 코드 비하인드 C#(*.cs)으로 가능하도록 완벽한 분리를 해놓았다. View와 ViewModel은 XAML XML파일의 Data Binding 정의를 통해 연결될 수도 있으며 View에 대한 BindingContext는 일반적으로 ViewModel의 인스턴스이다.

- Xamarin MVVM 예제

<https://developer.xamarin.com/samples/xamarin-forms/XAMLSamples/>

4.6.1 ViewModel을 View에 연결하기

- ViewModel은 Xamarin.Forms의 데이터 바인딩 기능을 사용하여 뷰(View)에 연결할 수 있다.
- 뷰를 구성하고 모델을 보고 런타임에 연관시키는 데 사용할 수 있는 많은 접근 방식이 있는데 이러한 접근법은 View 첫 번째 컴포지션과 ViewModel 첫 번째 컴포지션이라는 두 가지 범주로 나뉘는데 View 첫 번째 컴포지션과 ViewModel 첫 번째 컴포지션 중 하나를 선택하는 것은 선호도와 복잡성의 문제이다. 그러나 모든 접근법은 뷰가 BindingContext 속성에 할당된 뷰 모델을 가지는 것과 동일한 목표를 공유한다.
- 첫 번째 컴포지션 View에서 앱은 개념적으로 의존하는 ViewModel에 연결하는 View로 구성된다. 이 접근법의 이점은 뷰 모델이 뷰 자체에 의존하지 않기 때문에 느슨하게 결합된 단위 테스트 가능한 애플리케이션을 쉽게 구성 할 수 있다는 것이다. 클래스의 생성 및 연관을 이해하기 위해 코드 실행을 추적하지 않고도 시각적 구조를 따라 앱 구조를 쉽게 이해할 수 있다.
- ViewModel의 첫 번째 컴포지션에서 앱은 개념적으로 뷰 모델로 구성되며 서비스는 뷰 모델에 대한 뷰의 위치를 결정한다. 뷰 생성은 추상화되어 앱의 논리적 인 비 UI 구조에 집중할 수 있으므로 모델의 첫 번째 컴포지션 View는 일부 개발자에게 자연스럽게 느껴지며 또한 다른 ViewModel에서 ViewModel을 만들 수 있다. 그러나 이 방법은 종종 복잡하며 앱의 다양한 부분이 어떻게 만들어지고 연관되는지 이해하기 어려울 수 있다.
- 뷰 모델 및 뷰를 독립적으로 유지한다. 뷰의 데이터 소스 속성에 대한 바인딩은 해당 뷰 모델에 대한 뷰의 주요 종속성이어야 하며 특히 ViewModel에서 Button 및 ListView와 같은 뷰 유형을 참조하면 안된다.

4.6.2 Creating a View Model Declaratively

- 가장 간단한 접근법은 뷰가 XAML에서 해당 ViewModel을 선언적으로 인스턴스화 하는 것이다. 뷰가 구성되면 해당 ViewModel의 객체도 함께 생성된다.
- 아래 예문에서 ContentPage가 생성되면 EmpViewModel의 객체도 생성되고 뷰의 BindingContext로 EmpViewModel이 할당된다.

```
<ContentPage ... xmlns:local="clr-namespace:App6">
    <ContentPage.BindingContext>
        <local:EmpViewModel />
    </ContentPage.BindingContext>
```

```
...  
</ContentPage>
```

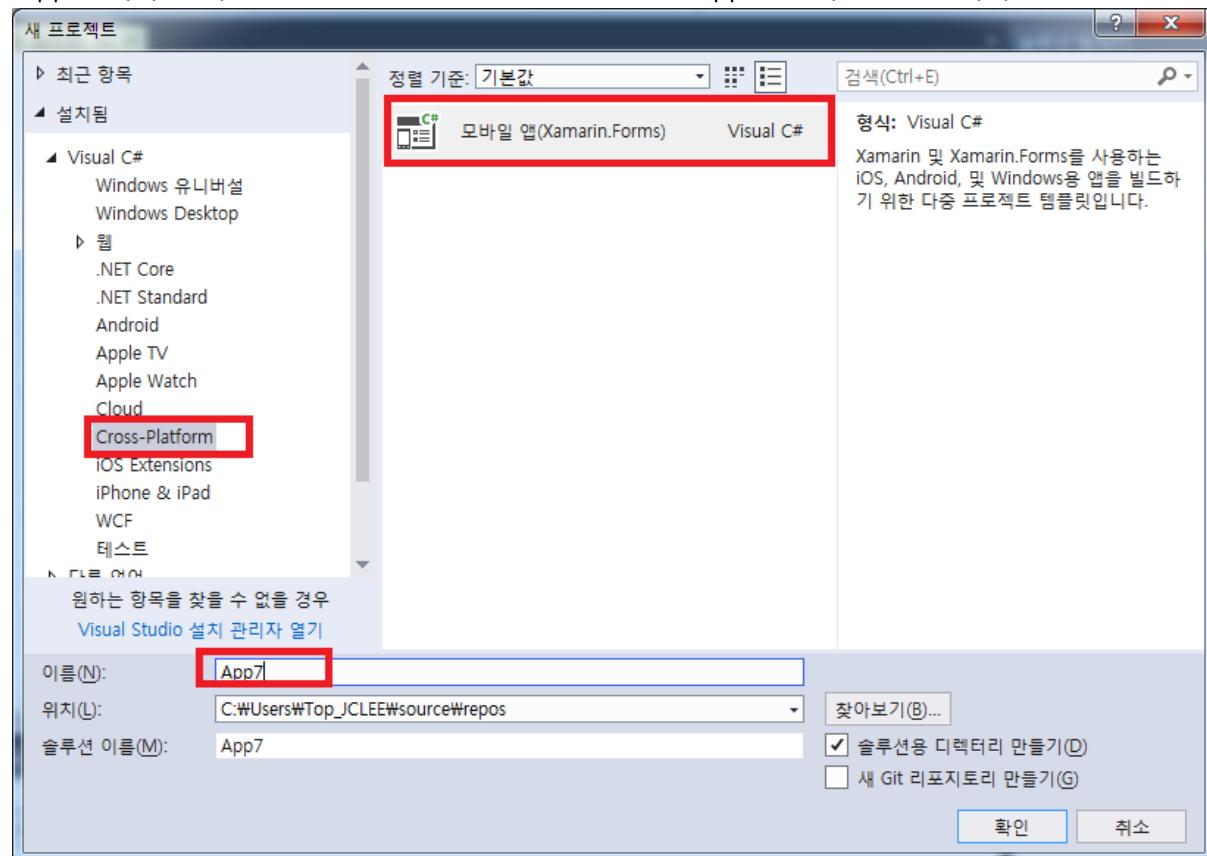
4.6.3 Creating a View Model Programmatically

- 뷰에는 코드 숨김 파일에 코드가 있어 ViewModel이 BindingContext 속성에 할당된다. 다음 코드 예제와 같이 뷰(View)의 생성자에서 수행되는 경우가 많다.
- 뷰의 코드 숨김 내에서 뷰 모델을 프로그래밍 방식으로 구성하고 할당하면 간단하다는 이점이 있으나 이 방법의 가장 큰 단점은 뷰가 뷰 모델에 필요한 종속성을 제공해야 한다는 것이다.

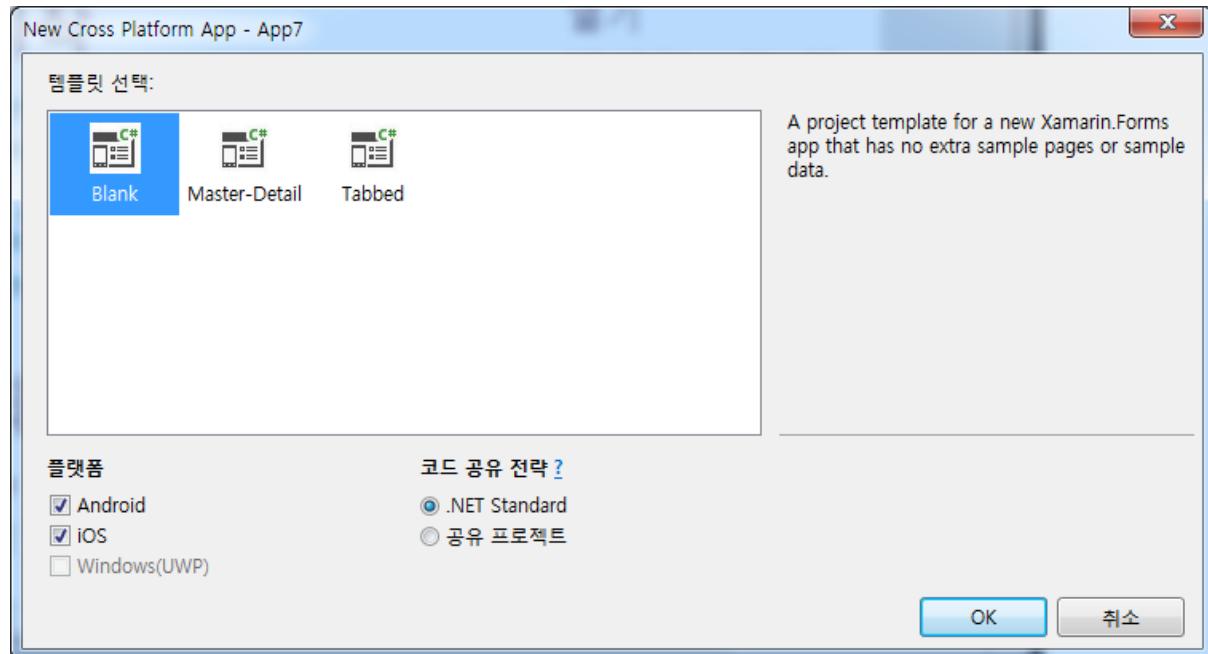
```
public MainPage()  
{  
    InitializeComponent();  
    BindingContext = new EmpViewModel(navigationService);  
}
```

4.6.4 Xamarin.Forms MVVM Hello World(Command Data Binding)

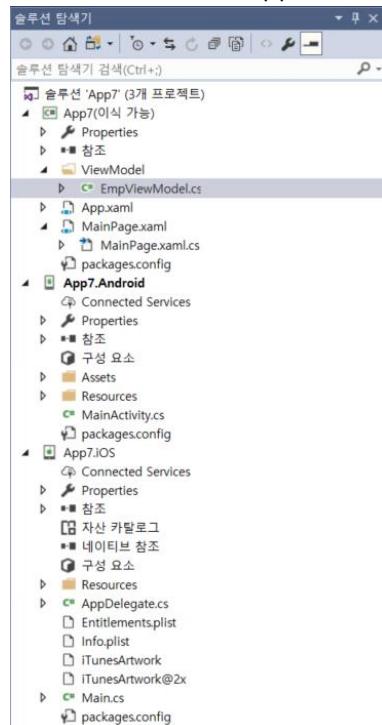
- “App7”이라는 이름으로 Xamarin.Forms Cross-Platform App 프로젝트를 생성하자.



- 비어있는 앱, .Net Standard를 선택한다.



- 이식가능 프로젝트(App7)에서 추가 >> 폴더로 ViewModel 폴더를 생성하자.



- [EmpViewModel.cs] ViewModel 클래스를 작성하자.

- INotifyPropertyChanged 인터페이스는 클라이언트(뷰의 컨트롤)에 속성값의 변경을 통지를 하기위해 사용하며 사용자가 버튼을 클릭했을때 변경된 값이 라벨 컨트롤에 반영된다.

- XAML의 Entry 텍스트 박스를 위한 public 프로퍼티 Ename과 private 프로퍼티 _Message, Label을 위한 public 프로퍼티인 Message가 포함되어 있다.
- public 프로퍼티인 Message의 값이 변경되면 OnPropertyChanged() 메소드를 호출하여 PropertyChanged 이벤트를 호출하여 값의 변경을 Message 프로퍼티에 바인딩 되어 있는 Label 컨트롤에 알린다.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Text;
using Xamarin.Forms;

namespace App7.ViewModel
{
    // INotifyPropertyChanged 인터페이스는 클라이언트(뷰의 컨트롤)에 속성값의 변경을
    // 통지할 때 사용, 사용자가 버튼을 클릭했을 때 변경된 값이 라벨 컨트롤에 반영된다.
    public class EmpViewModel : INotifyPropertyChanged
    {
        public string Ename { get; set; }
        private string _Message { get; set; }
        public string Message
        {
            get { return _Message; }
            set
            {
                _Message = value;
                OnPropertyChanged();
            }
        }

        public Command Introduce
        {
            get
            {
                return new Command( () => { Message = "My name is " + Ename; });
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName =
null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}

```

- View 역할을 하는 **[MainPage.xaml]**
- Button의 Command 이벤트가 ViewModel의 Introduce(Command 타입)에 바인딩 되어 있음

을 알수 있는데 버튼을 클릭하면 ViewModel의 public Command Introduce가 프로퍼티의 get이 호출된다.

- ContentPage의 BindingContext로 ViewModel 네임스페이스의 EmpViewModel이 설정되어 있고 Entry 및 Label에 EmpViewModel의 Ename, Message 속성이 바인딩 되어 있다.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:viewModel="clr-namespace:App7.ViewModel;assembly=App7"
    x:Class="App7.MainPage">

    <ContentPage.BindingContext>
        <viewModel:EmpViewModel />
    </ContentPage.BindingContext>

    <StackLayout Orientation="Vertical" Margin="10" VerticalOptions="Center"
    HorizontalOptions="CenterAndExpand">
        <Entry x:Name="Ename" Text="{Binding Ename}" />
        <Label x:Name="Message" Text="{Binding Message}" />
        <Button Text="당신을 소개하세요." Command="{Binding Introduce}" />
    </StackLayout>
</ContentPage>
```

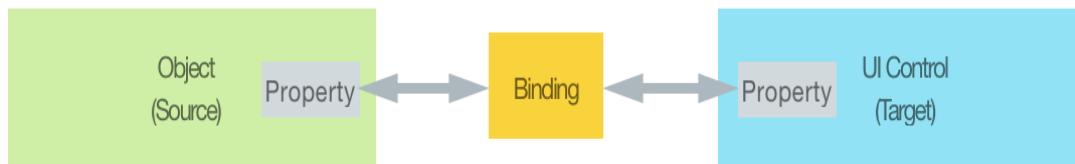
- 실행화면



4.7 XAML 데이터 바인딩(Data Binding)

4.7.1 데이터 바인딩(Data Binding) 개요

- 데이터 바인딩은 소스(데이터)와 타겟(소비자)을 연결시켜 주는 것이다. 예를들면 Label은 Text 속성이 문자열(소스 Object)의 public string 속성에 바인딩되는데 Label이 타겟이고 문자열이 소스다.



- 바인딩을 하는 장점은 더 이상 소스의 변화에 대해 고민할 필요가 없다는 것인데 소스가 변하면 타겟 객체에 자동으로 반영된다.
- ViewModel과 같은 CLR Object의 속성에 사용자 인터페이스의 Object의 속성을 접착제처럼 바인딩하는 것이다.
- 바인딩 되는 값과 컨트롤의 데이터 바인딩은 바인딩 된 값이 변하는 경우 동기화를 이루어야 하는데 컨트롤의 값이 변하는 것에 따른 모든 이벤트 핸들러를 통해 동기화하는 대신 ViewModel안의 데이터와 자동으로 동기화를 이루게 하는 것이다.
- 데이터 바인딩을 위해서는 두단계가 필요한데 **타겟 객체의 BindingContext 속성을 소스객체로 설정하고 타겟쪽의 XAML안에서 Binding 태그를 사용하거나 C#코드에서 SetBinding 메소드를 이용한다.**
- 바인딩은 언제나 타겟에 설정해야 하며 target 속성은 BindableObject 여야한다. 즉, 대상 객체는 BindableObject에서 파생되어야 한다.
- 타겟 개체의 BindingContext를 설정하는 방법은 여러가지가 있는데 코드비하인드 파일에서 설정할 수도 있고 **StaticResource** 또는 **x : Static** 태그 확장을 사용하는 경우도 있고 때로는 **BindingContext 속성 요소 태그의 내용**으로 사용하는 경우도 있다.
- 일반적으로 바인딩은 프로그램의 UI를 기본 데이터 모델(View Model)과 연결하는 데 사용되며, MVVM (Model-View-ViewModel) 패턴을 구현하는 용도로 사용된다.
- 데이터 바인딩은 이벤트 핸들러를 대체할 수 있는데 이는 C#코드를 줄이는 역할을 한다. XAML에서 정의된 데이터 바인딩은 C# 코드 비하인드 파일에서 이벤트 핸들러를 정의할 필요가 없으며 코드 비하인드 파일 자체가 필요없는 경우도 있다.
- 바인딩 기본 예문

```
<Entry Text="{Binding FirstName}" ... />
```

Entry.Text 속성과 소스의 FirstName 속성이 연결되며 Entry 컨트롤의 변화가 자동으로 employeeDisplay 객체에 전달된다. 또한 employeeToDisplay.FirstName 속성이 변하면 Xamarin.Forms의 바인딩 엔진이 Entry 컨트롤을 변경한다.

아래 코드는 EmployeeDetailPage 클래스의 BindingContext 속성이 XAML 안에 설정될 수 있

지만 C# 코드 비하인드를 이용하여 Employee 객체의 인스턴스에 설정하는 예문이다.

```
public EmployeeDetailPage(Employee employee)
{
    InitializeComponent();
    this.BindingContext = employee;
}
```

각 타겟 객체의 BindingContext 속성이 개별적으로 설정될 수 있지만 별로 필요하지는 않다. 왜냐하면 BindingContext는 모든 자식들이 상속받을 수 있는 특별한 속성이기에 ContentPage의 BindingContext 속성에 Employee가 설정되었다면 ContentPage의 모든 자식들이 같은 BindingContext를 가지며 Employee 객체의 public 속성에 바인딩 된다.

아래는 C# Data Binding 예문이다.

```
public EmployeeDetailPage(Employee employeeToDisplay)
{
    this.BindingContext = employeeToDisplay;
    var firstName = new Entry()
    {
        HorizontalOptions = LayoutOptions.FillAndExpand
    };
    firstName.SetBinding(Entry.TextProperty, "FirstName");
    ...
}
```

- Bindding Mode :
 - ✓ OneWay - 값이 원본에서 대상으로 전송.
 - ✓ OneWayToSource - 값이 대상에서 소스로 전송.
 - ✓ TwoWay - 값은 소스와 타겟간에 양방향으로 전송.
- Bindding Cells : 셀의 속성은 ItemSource 내에 있는 Object의 속성에 바운드할 수 있는데 아래 예문은 ListView가 이미지와 함께 사원리스트에 바인딩되는 예문이다.

[Employee Class]

```
public class Employee{
    public string DisplayName {get; set;}
}
```

[ObservableCollection<Employee>이 생성되고 ListView의 ItemsSource로 설정된다.]

```
ObservableCollection<Employee> employees = new ObservableCollection<Employee>();  
  
public EmployeeListPage()  
{  
    //defined in XAML to follow  
    EmployeeView.ItemsSource = employees;  
    ...
```

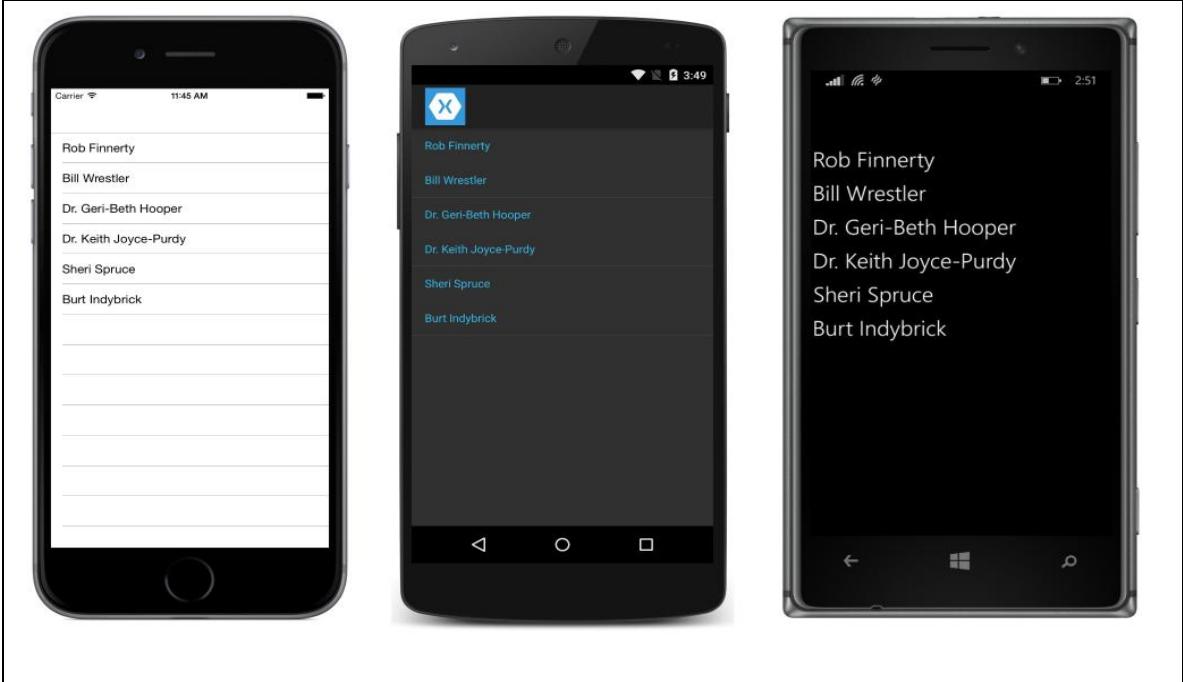
[사원 데이터를 채운다.]

```
employees.Add(new Employee{ DisplayName="Rob Finnerty"});  
employees.Add(new Employee{ DisplayName="Bill Wrestler"});  
employees.Add(new Employee{ DisplayName="Dr. Geri-Beth Hooper"});  
employees.Add(new Employee{ DisplayName="Dr. Keith Joyce-Purdy"});  
employees.Add(new Employee{ DisplayName="Sheri Spruce"});  
employees.Add(new Employee{ DisplayName="Burt Indybrick"});
```

아래 코드를 통해 ListView가 사원리스트에 바인딩 된다.

하나의 ListView를 포함하는 ContentPage이며 ListView의 data source는 ItemSource 속성을 경유해서 설정되며 ItemSource내 각 행의 Layout은 ListView.ItemTemplate 요소안에 정의된다.

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
xmlns:constants="clr-namespace:XamarinFormsSample;assembly=XamarinFormsXamlSample"  
x:Class="XamarinFormsXamlSample.Views.EmployeeListPage"  
Title="Employee List">  
    <ListView x:Name="EmployeeView">  
        <ListView.ItemTemplate>  
            <DataTemplate>  
                <TextCell Text="{Binding DisplayName}" />  
            </DataTemplate>  
        </ListView.ItemTemplate>  
    </ListView>  
</ContentPage>
```



■ Binding SelectedItem

간혹 ListView의 선택된 아이템을 이벤트 핸들러를 이용하는 것보다 바인딩을 원하기도 하는데 아래처럼 하면 된다. **SelectedItem** 속성을 이용한다.

```
<ListView x:Name="listView"
  SelectedItem="{Binding Source={x:Reference SomeLabel},
  Path=Text}">
  ...
</ListView>
```

■ Binding to a Custom Class

ListView 컨트롤은 기본 TextCell 템플릿을 사용해서 사용자정의 Object를 출력한다.

아래 예문을 확인하자.

```
public class TodoItem
{
    public string Name { get; set; }
    public bool Done { get; set; }
}

......

listView.ItemsSource = new TodoItem [] {
```

```
new TodoItem { Name = "Buy pears" },  
new TodoItem { Name = "Buy oranges", Done=true} ,  
new TodoItem { Name = "Buy mangos" },  
new TodoItem { Name = "Buy apples", Done=true } ,  
new TodoItem { Name = "Buy bananas", Done=true }  
};
```

ListView에 의해 출력될 TodoItem 속성을 설정하기 위해 바인딩한다.

```
listView.ItemTemplate = new DataTemplate(typeof(TextCell));  
listView.ItemTemplate.SetBinding(TextCell.TextProperty, "Name");
```

4.7.2 View-to-View 데이터 바인딩

- 데이터 바인딩은 동일한 페이지에서 두 View의 속성을 연결하도록 정의 할 수 있는데 이 경우 **x : Reference** 태그 확장을 사용하여 대상 객체의 **BindingContext**를 설정한다.
- 아래 예문은 Slider 및 두 개의 Label 뷰가 포함 된 XAML 파일로 위에 있는 Label은 하나는 Slider 값(Value 속성)이 변함에 따라 회전하고 아래 Label은 Slider 값(Value 속성)을 표시한다.
- **Xamarin.Forms 프로젝트 생성(Xamarin Cross-Platform, Blank Xaml App)해서 간단히 테스트 해보자.**



```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```

x:Class="XamlSamples.SliderBindingsPage"
Title="Slider Bindings Page"

<StackLayout>
    <Label Text="ROTATION"
        BindingContext="{x:Reference Name=slider}"
        Rotation="{Binding Path=Value}"
        FontAttributes="Bold"
        FontSize="Large"
        HorizontalOptions="Center"
        VerticalOptions="CenterAndExpand" />

    <Slider x:Name="slider"
        Maximum="360"
        VerticalOptions="CenterAndExpand" />

    <Label BindingContext="{x:Reference slider}"
        Text="{Binding Value,
            StringFormat='The angle is {0:F0} degrees'}"
        FontAttributes="Bold"
        FontSize="Large"
        HorizontalOptions="Center"
        VerticalOptions="CenterAndExpand" />
</StackLayout>
</ContentPage>

```

4.7.3 ListView 심플 데이터 바인딩, 컬렉션 바인딩(Collection Binding), ListView에서 클릭시 새창 띄우면서 데이터 넘기기

- Xamarin.Forms의 ListView는 IEnumerable 형식의 ItemsSource 속성을 가지며 컬렉션의 항목들을 표시한다. 이 항목은 모든 유형의 개체가 될 수 있는데 기본적으로 ListView는 각 항목의 ToString() 메서드를 사용하여 해당 항목을 표시하도록 설계 되어 있다.
- ListView 컬렉션의 항목(Item)은 셀(Cell)에서 파생되는 클래스가 포함 된 템플릿을 통해 원하는 방식으로 표시 할 수 있는데 템플릿은 ListView의 모든 항목에 대해서 복제된다.
- ViewCell 클래스를 사용하여 이러한 항목에 대한 사용자 정의 셀을 생성할 때 C# 코드에서는 다소 복잡하지만 XAML에서는 매우 간단하다.
- 예제 프로젝트에는 Emp라는 클래스가 있고 int 타입의 사번(Empno), string 타입의 이름(Ename) 속성을 가진다. 또한 Emp 클래스에는 여러 사원을 컬렉션 형태로 담을 수 있는

- Emps 속성을 가지며 static 생성자에서 Emps 속성을 초기화 했다.
- static 속성인 Emp.Emps를 ListView의 ItemsSource로 설정하는 방법은 XAML에서는 x : Static 태그 확장을 사용하면 되고, C# 코드에서는 리스트뷰의 ItemSource 속성에 컬렉션을 할당하면 된다.

```
XAML : <ListView ItemsSource="{x:Static local:Emp.Emps}" />
```

```
C#코드 : listView.ItemsSource = Emp.Emps;
```

- Xamarin.Forms, PCL(이식 가능한 클래스 라이브러리) 프로젝트를 생성하고 프로젝트명을 "App6"으로 입력하자.**

- [MainPage.xaml]**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:App6"
    x:Class="App6.MainPage">
    <ListView ItemsSource="{x:Static local:Emp.Emps}" />
</ContentPage>
```

[Emp.cs]

```
using System;
using System.Collections.Generic;
namespace App6
{
    public class Emp
    {
        public int Empno { get; set; }
        public string Ename { get; set; }

        public static IEnumerable<Emp> Emps { get; set; }

        static Emp()
        {
            Emps = new List<Emp>
            {
                new Emp() { Empno = 1, Ename = "홍길동" },
                new Emp() { Empno = 2, Ename = "박길동" },
                new Emp() { Empno = 3, Ename = "정길동" },
                new Emp() { Empno = 4, Ename = "나길동" },
                new Emp() { Empno = 5, Ename = "김길동" }

            };
        }
}
```

```

        public override string ToString()
    {
        return string.Format("Empno : {0}, Ename : {1}", Empno,
Ename);
    }
}

```

- [실행화면] : Emp 클래스의 ToString() 메소드가 실행됨을 알 수 있다.



- 이번에는 ListView의 항목(Item)에 대한 템플릿을 정의해서 데이터를 출력해 보자.
- 프로퍼티 엘리먼트로 ItemTemplate 속성을 분리하여 DataTemplate으로 설정 한 다음 ViewCell을 참조해야 하는데 ViewCell의 View 속성에 각 항목을 표시 할 하나 이상의 레이아웃을 정의 할 수 있다.
- 아래 예문에서는 Grid 요소가 ViewCell의 View 프로퍼티 안에 정의되고 그 안에 Label 요소를 들어하고 데이터 바인딩을 통해 Emp의 Empno, Ename 속성값들이 출력된다.
- [MainPage.xaml] 을 수정하자.

```

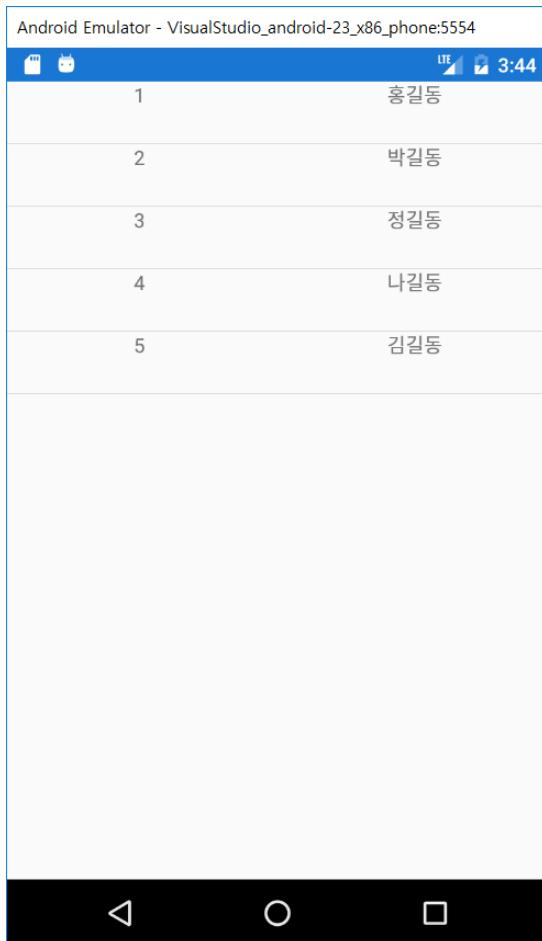
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:App6"
    x:Class="App6.MainPage">

    <ListView ItemsSource="{x:Static local:Emp.Emps}">

```

```
<ListView.ItemTemplate>
    <DataTemplate>
        <ViewCell>
            <ViewCell.View>
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="1*" />
                        <ColumnDefinition Width="1*" />
                    </Grid.ColumnDefinitions>
                    <Label Grid.Column="0" Text="{Binding Empno}" HorizontalOptions="Center" />
                    <Label Grid.Column="1" Text="{Binding Ename}" HorizontalOptions="Center" />
                </Grid>
            </ViewCell.View>
        </ViewCell>
    </DataTemplate>
</ListView.ItemTemplate>
</ListView>
</ContentPage>
```

■ [실행결과]



- 이번에는 ListView에서 사용자가 항목을 클릭했을 때 새창을 띄우고 사번과 이름을 새로운 화면에 출력해 보자.
- 새창을 띄우기 위해서는 Navigation.PushAsync()를 사용해야 하는데 App.xaml.cs의 생성자 부분을 아래와 같이 수정하자.

- [App.xaml.cs]

```
public App()
{
    InitializeComponent();

    // MainPage = new App6.MainPage();
    // PushAsync is not supported globally on Android, please use
    // a NavigationPage 오류 발생을 방지하기 위해 NavigationPage를 사용
    MainPage = new NavigationPage(new MainPage());
}
```

- [MainPage.xaml]에서는 ListView를 코드 비하인드에서 사용해야 하므로 ListView에 이름속성 (x:Name)이 추가되었다.

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:App6"
    x:Class="App6.MainPage">

    <ListView x:Name="listView" ItemsSource="{x:Static local:Emp.Emps}">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <ViewCell.View>
                        <Grid>
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="1*" />
                                <ColumnDefinition Width="1*" />
                            </Grid.ColumnDefinitions>
                            <Label Grid.Column="0" Text="{Binding Empno}" HorizontalOptions="Center" />
                            <Label Grid.Column="1" Text="{Binding Ename}" HorizontalOptions="Center" />
                        </Grid>
                    </ViewCell.View>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</ContentPage>

```

- [MainPage.xaml.cs] 파일에서는 ListView의 ItemSelected 이벤트 처리용 코드가 추가되었다.

```

using Xamarin.Forms;

namespace App6
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();

            listView.ItemSelected += async (sender, e) => {
                if (e.SelectedItem == null) return;

```

```

        var emp = e.SelectedItem as Emp;

        //await DisplayAlert("Tapped!", e.SelectedItem + " was
        tapped.", "OK");
        var nextPage = new NextPage();
        nextPage.BindingContext = emp;
        await Navigation.PushAsync(nextPage);
    };
}
}
}

```

- 추가 >> 새항목추가 >> Xamarin.Forms ContentPage 템플릿 형식으로 "NextPage.xaml"을 추가하자.

- [NextPage.xaml]

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="App6.NextPage">
    <ContentPage.Content>
        <StackLayout>
            <Label Text="{Binding Empno}" />
            <Label Text="{Binding Ename}" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>

```

- [NextPage.xaml.cs] 파일은 수정할 필요 없다.
- [Emp.cs] 파일은 이전과 동일하다.

```

using System;
using System.Collections.Generic;
namespace App6
{
    public class Emp
    {
        public int Empno { get; set; }
        public string Ename { get; set; }

        public static IEnumerable<Emp> Emps { get; set; }

        static Emp()
        {

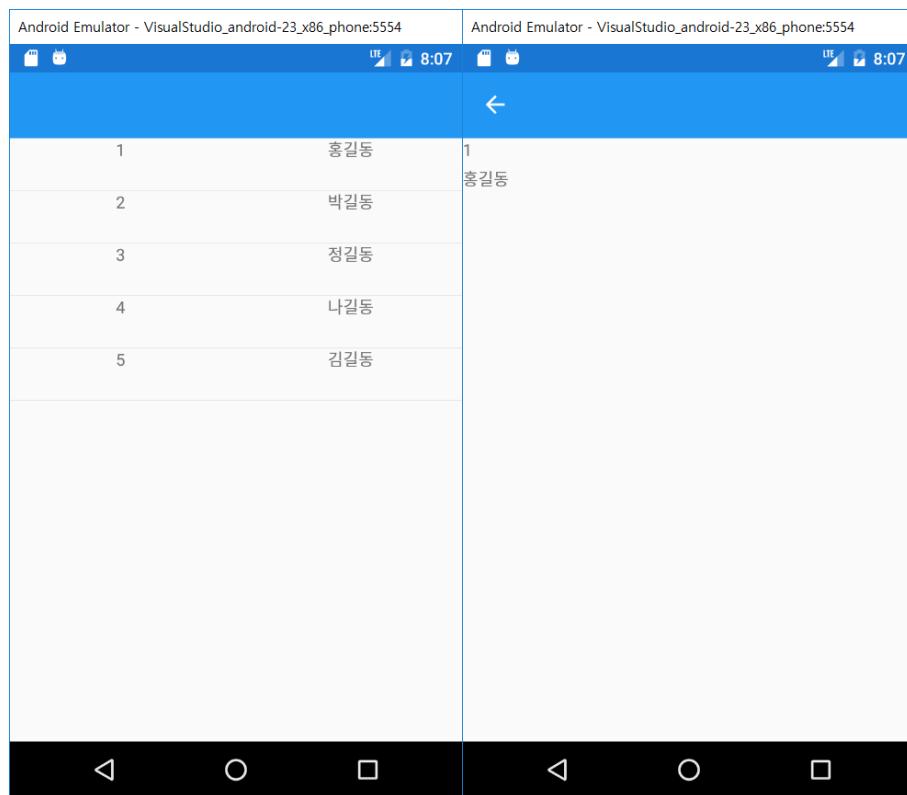
```

```

        Emps = new List<Emp>
    {
        new Emp() { Empno = 1, Ename = "홍길동" },
        new Emp() { Empno = 2, Ename = "박길동" },
        new Emp() { Empno = 3, Ename = "정길동" },
        new Emp() { Empno = 4, Ename = "나길동" },
        new Emp() { Empno = 5, Ename = "김길동" }
    };

    public override string ToString()
    {
        return string.Format("Empno : {0}, Ename : {1}", Empno,
Ename);
    }
}

```

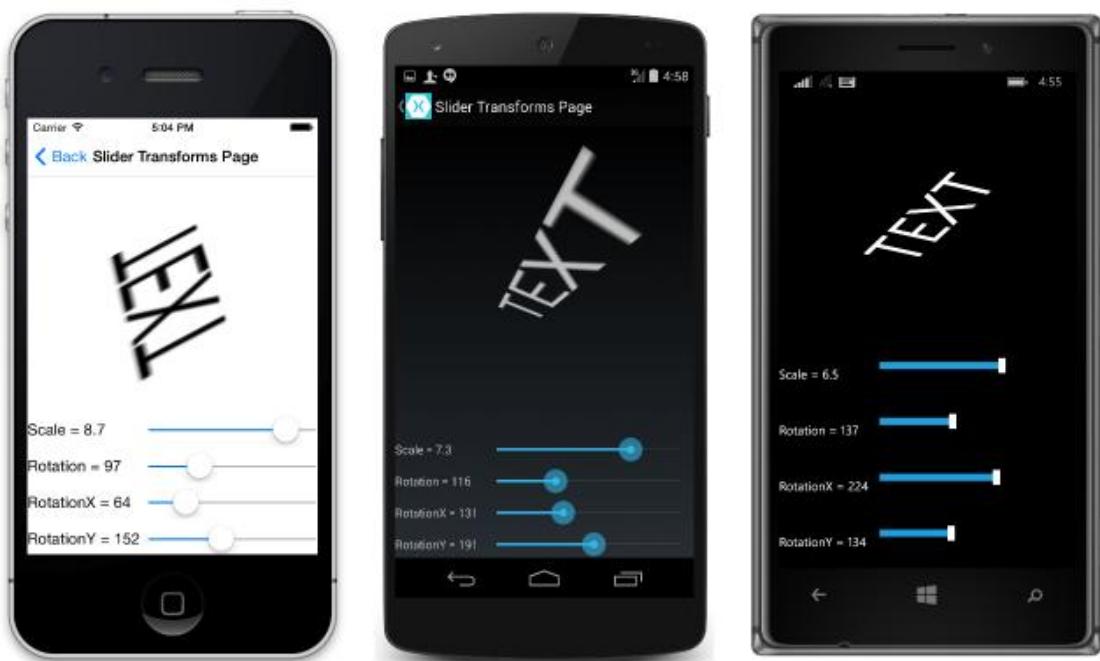


4.7.4 Backwards 바인딩

- 단일 뷰는 여러 속성이 데이터 바인딩을 가질 수 있다. 그러나 각 VIEW에는 하나의

BindingContext만 있을 수 있으므로 해당 VIEW의 여러 데이터 바인딩은 모두 동일한 개체의 속성을 참조해야 한다.

- 이러한 제한 사항을 피하기 위해 OneWayToSource 또는 TwoWay 모드를 사용하여 View - View 바인딩을 정의해야 하는 경우가 있다.
- 아래 예제에서는 Label의 Scale, Rotate, RotateX 및 RotateY 속성을 제어하기 위한 4 개의 Slider 뷰가 있다. 이를 위해 Slider에서 Label을 설정하기 때문에 Label의 네 가지 속성이 데이터 바인딩 타겟이어야 하는 것처럼 보이지만 Label의 BindingContext는 하나만 가능한데 4 개의 다른 슬라이더가 있기 때문에 모든 바인딩이 뒤집어져 있다.
- 네 개의 슬라이더 각각에 BindingContext가 Label로 설정되면서 바인딩이 슬라이더의 Value 속성에 설정되고 OneWayToSource 및 TwoWay 모드를 사용하면서 이 Value 속성을 사용하여 레이블의 Scale, Rotate, RotateX 및 RotateY 속성인 소스 속성을 설정한다.



```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="App5.MainPage"
    Title="Slider Transforms Page">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid>
```

```

        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

    <StackLayout Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2">

        <!-- Scaled and rotated Label -->
        <Label x:Name="label"
            Text="TEXT"
            HorizontalOptions="Center"
            VerticalOptions="CenterAndExpand" />

    </StackLayout>

        <!-- Slider and identifying Label for Scale -->
        <Slider x:Name="scaleSlider"
            BindingContext="{x:Reference label}"
            Grid.Row="1" Grid.Column="1"
            Maximum="10"
            Value="{Binding Scale, Mode=TwoWay}" />

        <Label BindingContext="{x:Reference scaleSlider}"
            Text="{Binding Value, StringFormat='Scale = {0:F1}'}"
            Grid.Row="1" Grid.Column="0"
            VerticalTextAlignment="Center" />

        <!-- Slider and identifying Label for Rotation -->
        <Slider x:Name="rotationSlider"
            BindingContext="{x:Reference label}"
            Grid.Row="2" Grid.Column="1"
            Maximum="360"
            Value="{Binding Rotation, Mode=OneWayToSource}" />

        <Label BindingContext="{x:Reference rotationSlider}"
            Text="{Binding Value, StringFormat='Rotation = {0:F0}'}"

```

```

        Grid.Row="2" Grid.Column="0"
        VerticalTextAlignment="Center" />

    <!-- Slider and identifying Label for RotationX -->
    <Slider x:Name="rotationXSlider"
        BindingContext="{x:Reference label}"
        Grid.Row="3" Grid.Column="1"
        Maximum="360"
        Value="{Binding RotationX, Mode=OneWayToSource}" />

    <Label BindingContext="{x:Reference rotationXSlider}"
        Text="{Binding Value, StringFormat='RotationX = {0:F0}'}"
        Grid.Row="3" Grid.Column="0"
        VerticalTextAlignment="Center" />

    <!-- Slider and identifying Label for RotationY -->
    <Slider x:Name="rotationYSlider"
        BindingContext="{x:Reference label}"
        Grid.Row="4" Grid.Column="1"
        Maximum="360"
        Value="{Binding RotationY, Mode=OneWayToSource}" />

    <Label BindingContext="{x:Reference rotationYSlider}"
        Text="{Binding Value, StringFormat='RotationY = {0:F0}'}"
        Grid.Row="4" Grid.Column="0"
        VerticalTextAlignment="Center" />

```

</Grid>

</ContentPage>

4.7.5 MVVM에서 데이터 바인딩 사용하기

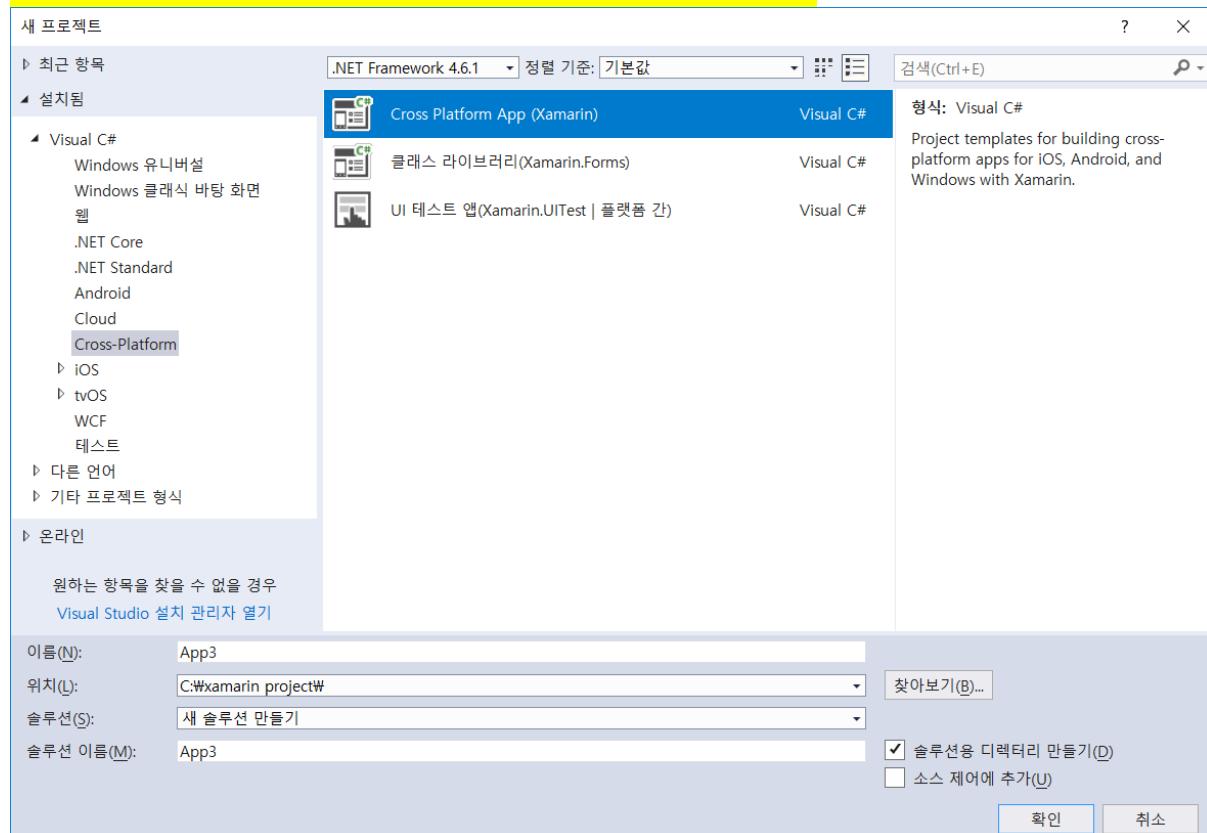
- Model-View-ViewModel (MVVM) 아키텍처 패턴은 XAML을 염두에 두고 고안되었는데 이 패턴은 기본 데이터(모델)에서 View와 모델 (ViewModel) 사이의 중간 역할을 하는 클래스를 통해 XAML 사용자 인터페이스(View)를 분리한다. View와 ViewModel은 주로 XAML 파일에 정의된 데이터 바인딩을 통해 연결되고 View의 BindingContext는 일반적으로 ViewModel의 인스턴스이다.
- 우선 **ViewModel이 없는 간단한 예제를 만들어 보는데** System 네임스페이스에 대한 XML 네임스페이스(sys)를 다음과 같이 정의하여 DateTime 구조체를 사용해 보자.

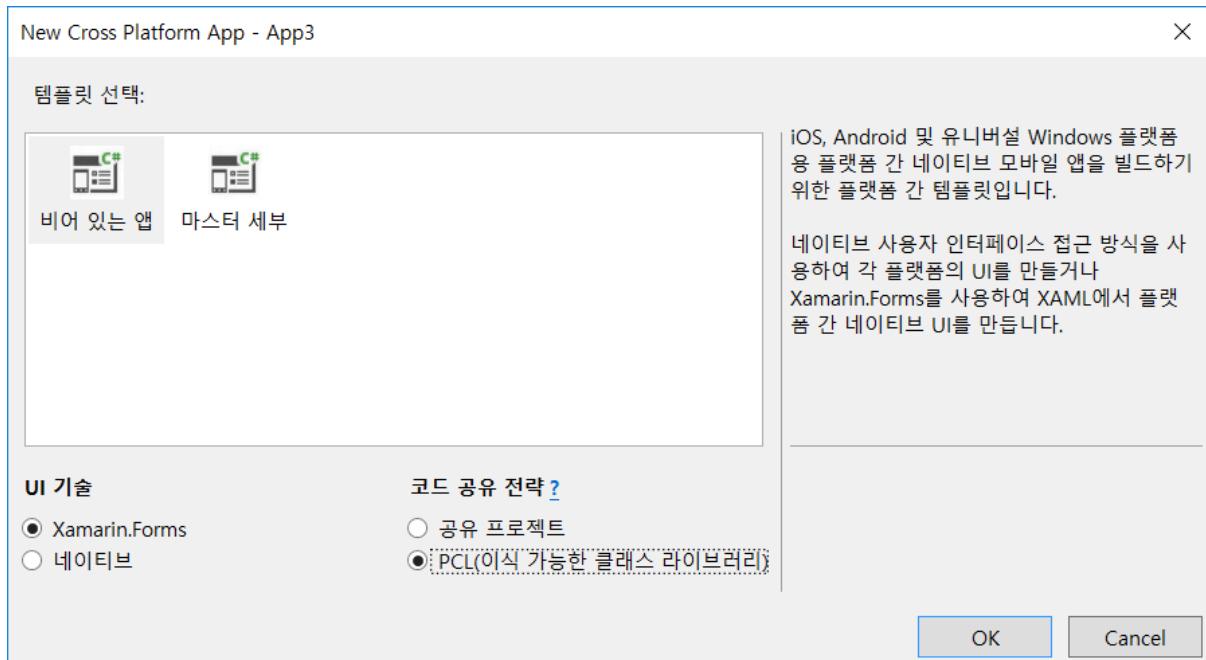
```
xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

- DateTime.Now 속성의 Year, Month, Day, 시간 속성을 각각의 Label 컨트롤에 바인딩 시키고 TimeOfDay 속성을 TimePicker 컨트롤에 바인딩 시키는 예제이다.
- Grid의 BindingContext로 x:Static 키워드를 통해 DateTime.Now 속성을 정의하자. 이 경우 Grid의 자식컨트롤로 BindingContext 속성은 상속되며 이 속성을 통해 바인딩 시킬 소스 데이터를 지정한다.

```
<Grid BindingContext="{x:Static sys:DateTime.Now}" ...>
```

■ 비주얼 스튜디오에서 크로스 플랫폼 앱 프로젝트를 생성하자.





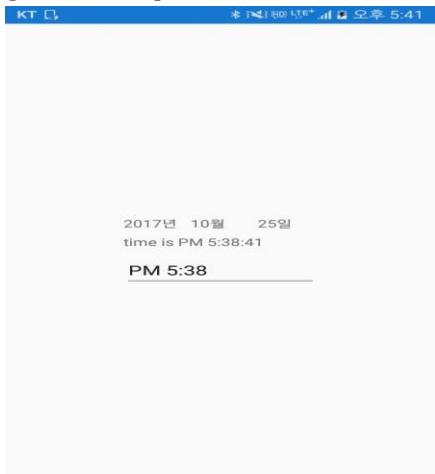
■ MainPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:App3"
    xmlns:sys="clr-namespace:System;assembly=mscorlib"
    x:Class="App3.MainPage">
    <Grid BindingContext="{x:Static sys:DateTime.Now}"
        HorizontalOptions="Center" VerticalOptions="Center">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="50"/>
            <ColumnDefinition Width="50"/>
            <ColumnDefinition Width="50"/>
        </Grid.ColumnDefinitions>

        <Label Text="{Binding Year, StringFormat='{0}년'}" Grid.Row="0"
            Grid.Column="0"/>
        <Label Text="{Binding Month, StringFormat='{0}월'}"
            Grid.Row="0" Grid.Column="1"/>
        <Label Text="{Binding Day, StringFormat='{0}일'}" Grid.Row="0"
            Grid.Column="2"/>
    </Grid>
</ContentPage>
```

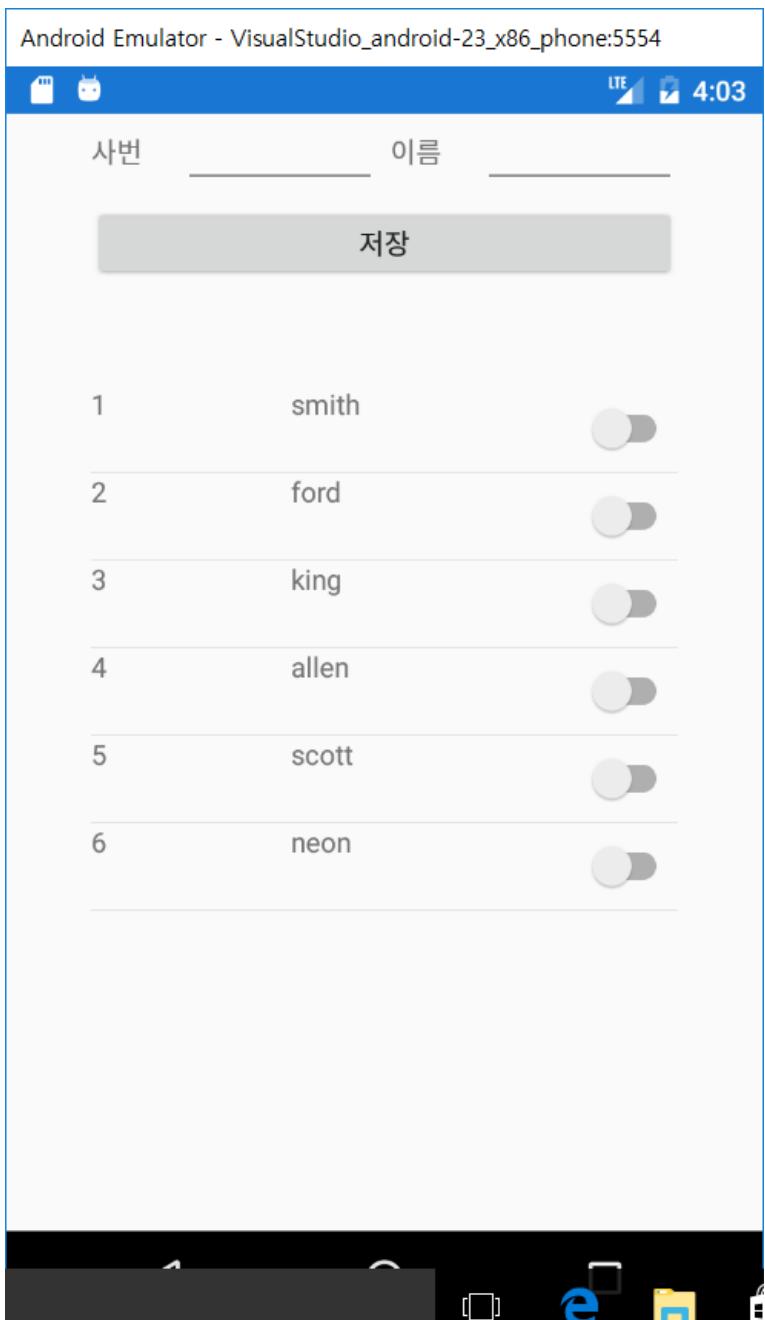
```
Grid.Column="2"/>
    <Label Text="{Binding StringFormat='time is {0:T}'}"
Grid.Row="1" Grid.ColumnSpan="3"/>
    <TimePicker x:Name="timepicker" Time="{Binding TimeOfDay}"
Grid.Row="2" Grid.ColumnSpan="3"/>
</Grid>
</ContentPage>
```

[실행결과]



4.7.6 MVVM, ViewModel을 이용한 ListView 데이터 바인딩

- Model, ViewModel을 간단히 만들고 ViewModel을 ListView에 바인딩하는 간단한 예제를 작성해 보자.
- 사원의 사번과 이름을 입력하면 아래쪽 ListView에 바인딩되서 데이터가 뿌려지는 간단한 예제로 실행화면은 다음과 같다.



■ Emp.cs

```
namespace DataBindingTest
{
    class Emp
    {
        public string Empno { get; set; }
        public string Ename { get; set; }
        public bool IsChecked { get; set; }
    }
}
```

■ EmpViewModel.cs

```
using System.Collections.ObjectModel;

namespace DataBindingTest
{
    class EmpViewModel
    {
        // Collection의 변화(Add/Delete)를 자동으로 감지하여 UI화면을
        // 자동갱신
        // ObservableCollection은 INotifyChanged 인터페이스를 구현했다.
        // ListView를 자동으로 업데이트 하기위해 ObservableCollection을
        // 사용.
        private ObservableCollection<Emp> emps = new
        ObservableCollection<Emp>();
        public ObservableCollection<Emp> Emps
        {
            get
            {
                return emps;
            }
            set
            {
                emps = value;
            }
        }

        public EmpViewModel()
        {
            Emps = new ObservableCollection<Emp>();
        }
    }
}
```

■ MainPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:DataBindingTest"
```

```

    x:Class="DataBindingTest.MainPage">
<Grid HorizontalOptions="Center">
    <Grid.RowDefinitions>
        <RowDefinition Height="40"/>
        <RowDefinition Height="40"/>
        <RowDefinition Height="40"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="40"/>
        <ColumnDefinition Width="100"/>
        <ColumnDefinition Width="40"/>
        <ColumnDefinition Width="100"/>
    </Grid.ColumnDefinitions>

    <Label Text="사번" Grid.Row="0" Grid.Column="0"
VerticalOptions="Center"/>
        <Entry x:Name="txtEmpno" Grid.Row="0" Grid.Column="1"
VerticalOptions="Center"/>
            <Label Text="이름" Grid.Row="0" Grid.Column="2"
VerticalOptions="Center"/>
                <Entry x:Name="txtEname" Grid.Row="0" Grid.Column="3"
VerticalOptions="Center"/>

            <Button x:Name="btnSave" Text="저장" Grid.Row="1"
Grid.ColumnSpan="4"
VerticalOptions="Center" Clicked="btnSave_Click"/>

        <ListView x:Name="listView"
            Grid.Row="3" Grid.ColumnSpan="4"
VerticalOptions="Center">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <ViewCell.View>
                            <Grid>
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition Width="1*"/>
                                    <ColumnDefinition Width="1*"/>
                                    <ColumnDefinition Width="1*"/>
                                </Grid.ColumnDefinitions>
                                <Label Grid.Column="0" Text="{Binding
Empno}"></Label>
                            </Grid>
                        </ViewCell.View>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>

```

```

        <Label Grid.Column="1" Text="{Binding
Ename}"></Label>
        <Switch Grid.Column="2" IsToggled="{Binding
IsChecked}"></Switch>
    </Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</Grid>
</ContentPage>

```

■ MainPage.xaml.cs

```

using System;
using Xamarin.Forms;

namespace DataBindingTest
{
    public partial class MainPage : ContentPage
    {
        EmpViewModel viewModel = new EmpViewModel();

        public MainPage()
        {
            InitializeComponent();
            listView.ItemsSource = viewModel.Emps;
            //BindingContext = viewModel;
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            viewModel.Emps.Add(new Emp() { Empno = txtEmpno.Text,
                Ename = txtEname.Text,
                IsChecked = false });

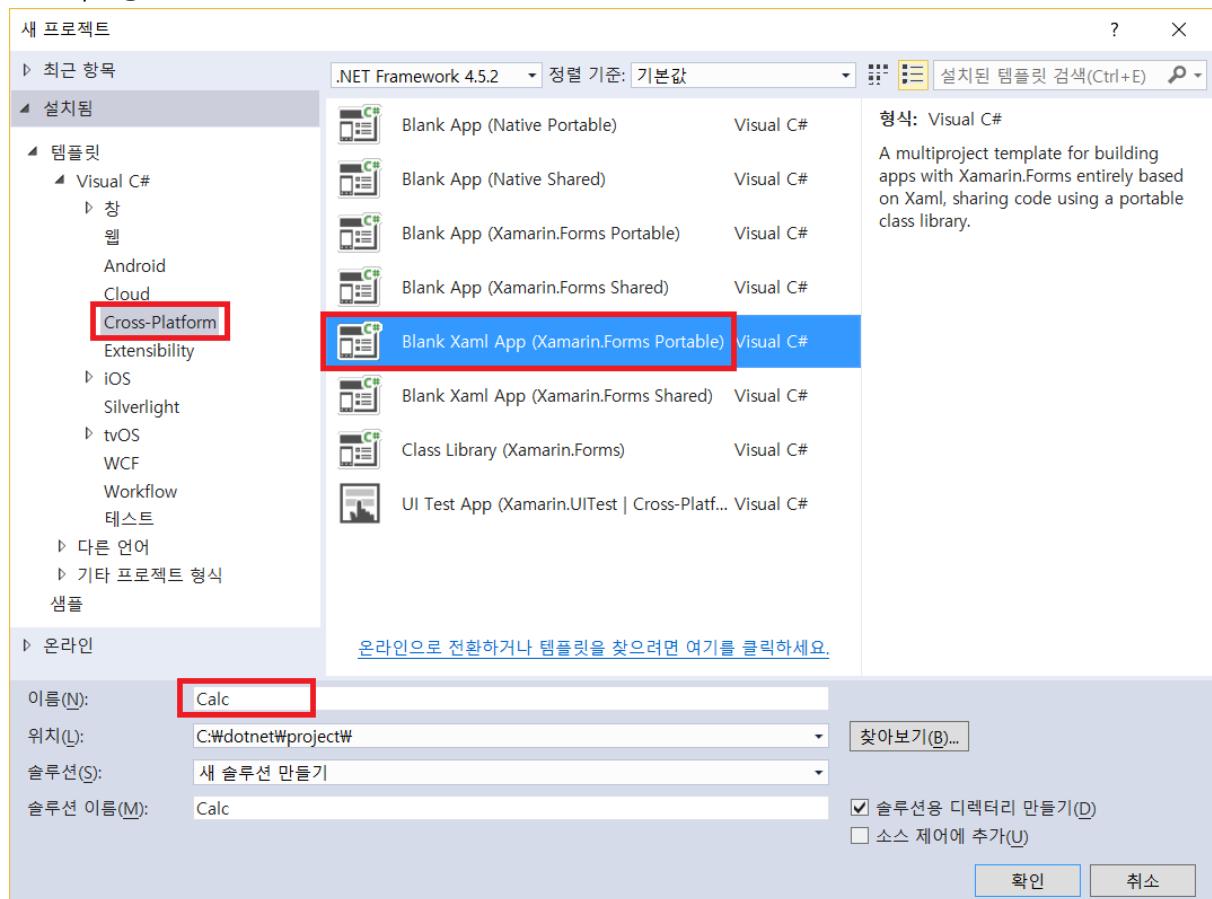
            txtEmpno.Text = "";
            txtEname.Text = "";
        }
    }
}

```

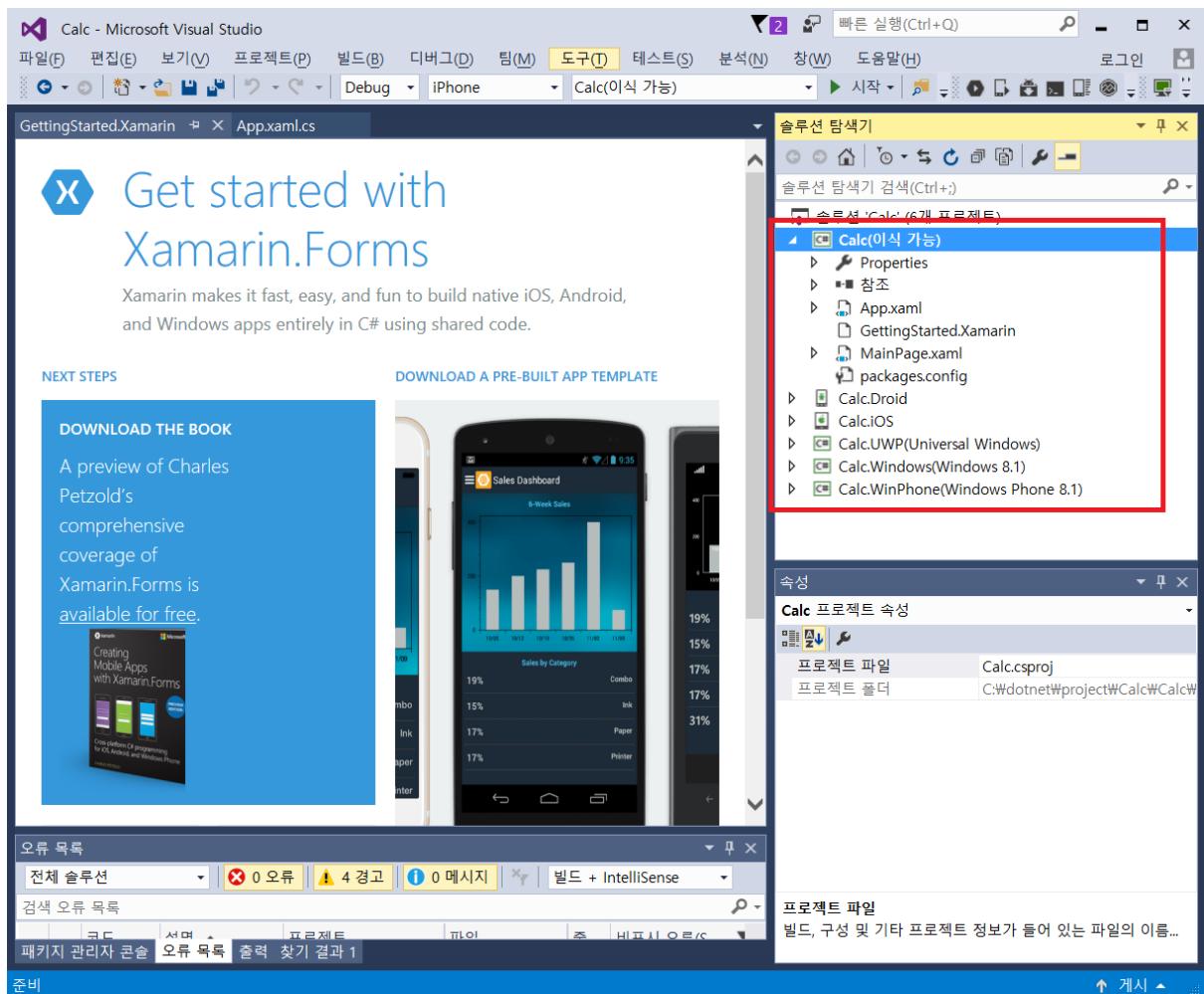
4.7.7 MVVM, XAML을 이용한 간단한 계산기 구현

- MVVM, XAML 패턴을 이해하기 위한 사칙연산만 수행하는 간단한 계산기를 만들어 보자. 본 예제를 기본으로 복잡한 계산기 기능을 완성해 보세요.
- 프로젝트 생성(Xamarin Cross-Platform, Blank Xaml App)
 - iPhone App을 위해서 Mac 장비가 있어야 한다.

프로젝트명 : Calc



- Portable, Android, iOS, Window UWP, Window8 App, Windows Phone8을 위한 6개의 프로젝트가 생성되며 프로젝트 생성 후 모습은 다음과 같다.



- Calc(이식 가능) 프로젝트에 App.xaml 파일이 응용프로그램 시작점이며 그안에서 MainPage를 호출하면서 프로그램이 시작된다. 계산기 예제의 경우 **Calc(이식 가능)** 프로젝트 쪽에만 공통 코드를 작성하면 안드로이드, 아이폰, 윈도우등 크로스 플랫폼으로 배포가능 하다.
- App.xaml.cs 파일은 수정할 필요는 없고 MainPage를 호출하는 부분은 아래와 같으니 확인만 하자.

```
public App()
{
    InitializeComponent();

    MainPage = new Calc.MainPage();
}
```

- Calc(이식 가능) 프로젝트의 MainPage.xaml(UI담당, VIEW) 파일을 아래와 같이 만들자. MainPage.xaml.cs 파일은 특별히 수정할 필요는 없다.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:Calc;assembly=Calc"
    x:Class="Calc.MainPage"
    Title="Calculator Page">

    <Grid HorizontalOptions="Center"
        VerticalOptions="Center">

        <!-- BindingContext에 ViewModel 클래스의 이름을 기술 -->
        <Grid.BindingContext>
            <local:CalcViewModel />
        </Grid.BindingContext>

        <!-- Grid의 행과 열을 정의 -->
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <!-- 최상단 행을 위한 내부 그리드 정의 -->
        <Grid Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="4">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>

            <Frame Grid.Column="0">
```

```

<Label Text="Xamarin Calculator"
    HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
    IsVisible="true"
    FontAttributes="Bold" TextColor="Black" />
</Frame>
</Grid>

<!-- 두번째 행을 위한 내부 그리드 정의, 출력텍스트박스, BACK버튼, Clear버튼 -->
<Grid Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="4">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>

    <!-- ViewModel 클래스의 DisplayText 속성과 바인딩 -->
    <Frame Grid.Column="0"
        OutlineColor="Accent">
        <Label Text="{Binding DisplayText}" />
    </Frame>

    <!-- 버튼의 Command 이벤트에 Command명을 지정하고 바인딩-->
    <!-- ViewModel 클래스에서 DeleteCharCommand 속성이 정의되어 하고 -->
    <!-- 실제 버튼이 눌러졌을 때 Command에 대한 이벤트 핸들러를 정의해야 한다.-->
    <Button Text="BACK"
        Command="{Binding DeleteCharCommand}"
        Grid.Column="1"
        BorderWidth="0" />
    <Button Text="Clear"
        Command="{Binding ClearCommand}"
        Grid.Column="2"
        BorderWidth="0" />
</Grid>

<Button Text="1"
    Command="{Binding AddCharCommand}"
    CommandParameter="1"
    Grid.Row="2" Grid.Column="0" />

```

```
<Button Text="2"
    Command="{Binding AddCharCommand}"
    CommandParameter="2"
    Grid.Row="2" Grid.Column="1" />

<Button Text="3"
    Command="{Binding AddCharCommand}"
    CommandParameter="3"
    Grid.Row="2" Grid.Column="2" />
<Button Text="+"
    Command="{Binding OperationCommand}"
    CommandParameter="+"
    Grid.Row="2" Grid.Column="3" />

<Button Text="4"
    Command="{Binding AddCharCommand}"
    CommandParameter="4"
    Grid.Row="3" Grid.Column="0" />

<Button Text="5"
    Command="{Binding AddCharCommand}"
    CommandParameter="5"
    Grid.Row="3" Grid.Column="1" />

<Button Text="6"
    Command="{Binding AddCharCommand}"
    CommandParameter="6"
    Grid.Row="3" Grid.Column="2" />
<Button Text="-"
    Command="{Binding OperationCommand}"
    CommandParameter="-"
    Grid.Row="3" Grid.Column="3" />

<Button Text="7"
    Command="{Binding AddCharCommand}"
    CommandParameter="7"
    Grid.Row="4" Grid.Column="0" />

<Button Text="8"
```

```

        Command="{Binding AddCharCommand}"
        CommandParameter="8"
        Grid.Row="4" Grid.Column="1" />

<Button Text="9"
        Command="{Binding AddCharCommand}"
        CommandParameter="9"
        Grid.Row="4" Grid.Column="2" />
<Button Text="*"
        Command="{Binding OperationCommand}"
        CommandParameter="*"
        Grid.Row="4" Grid.Column="3" />

<Button Text="0"
        Command="{Binding AddCharCommand}"
        CommandParameter="0"
        Grid.Row="5" Grid.Column="0" />

<Button Text="."
        Command="{Binding AddCharCommand}"
        CommandParameter="."
        Grid.Row="5" Grid.Column="1" />

<Button Text="="
        Command="{Binding CalcCommand}"
        CommandParameter="="
        Grid.Row="5" Grid.Column="2" />
<Button Text="/"
        Command="{Binding OperationCommand}"
        CommandParameter="/"
        Grid.Row="5" Grid.Column="3" />

```

</Grid>

</ContentPage>

■ Calc(이식가능, PCL) 프로젝트에 VIEW를 표현하고 Command를 구현한 ViewModel을 만들자.
(CalcViewModel.cs)

```

using System;
using System.ComponentModel;

```

```
using System.Windows.Input;
using Xamarin.Forms;

namespace Calc
{
    class CalcViewModel : INotifyPropertyChanged
    {
        //아래 두 필드는 속성으로 구현되어 있다.
        //출력될 문자들을 담아둘 변수
        string inputString = "";

        //계산기화면의 출력 텍스트박스에 대응되는 필드
        string displayText = "";

        //속성이 바뀔때 이벤트 발생하도록 이벤트 정의
        public event PropertyChangedEventHandler PropertyChanged;

        //생성자
        public CalcViewModel()
        {
            //이벤트 핸들러 정의
            //숫자 버튼을 클릭할 때 실행
            this.AddCharCommand = new Command<string>((key) =>
            {
                this.InputString += key;
            });

            //백스페이스 버튼을 클릭할 때 실행, 한글자 삭제
            this.DeleteCharCommand = new Command((nothing) =>
            {
                this.InputString = this.InputString.Substring(0,
                    this.InputString.Length - 1);
            },
            (nothing) =>
            {
                // Return true if there's something to delete.
                return this.InputString.Length > 0;
            });
        }
    }
}
```

```

//Clear 버튼을 클릭할 때 실행, 출력창을 전부 지운다.
this.ClearCommand = new Command((nothing) =>
{
    // Clear
    this.InputString = "";
});

//+,-,*/ 버튼을 클릭할 때 실행
//현재출력창의 숫자를 Op1 속성에 저장, Op속성에 클릭한 연산자 저장
//계산기의 출력화면을 Clear
this.OperationCommand = new Command<string>((key) =>
{
    this.Op = key;
    this.Op1 = Convert.ToDouble(this.InputString);
    this.InputString = "";
});

// =버튼을 클릭시 실행
this.CalcCommand = new Command<string>((nothing) =>
{
    this.Op2 = Convert.ToDouble(this.InputString);

    switch (this.Op)
    {
        case "+": this.InputString = (this.Op1 + this.Op2).ToString(); break;
        case "-": this.InputString = (this.Op1 - this.Op2).ToString(); break;
        case "*": this.InputString = (this.Op1 * this.Op2).ToString(); break;
        case "/": this.InputString = (this.Op1 / this.Op2).ToString(); break;
    }
});

// Public 속성들을 정의
public string InputString
{
    protected set
    {
        if (inputString != value)
        {

```

```

        inputString = value;
        OnPropertyChanged("InputString");
        this.DisplayText = inputString;

        // 숫자버튼을 클릭하면 백스페이스 버튼을 활성화 시킨다.
        ((Command)this.DeleteCharCommand).ChangeCanExecute();
    }
}

get { return inputString; }

}

//계산기의 출력창과 바인딩된 속성
public string DisplayText
{
    protected set
    {
        if (displayText != value)
        {
            displayText = value;
            OnPropertyChanged("DisplayText");
        }
    }
    get { return displayText; }
}

public string Op { get; set; }
public double Op1 { get; set; }
public double Op2 { get; set; }

// 숫자 클릭
public ICommand AddCharCommand { protected set; get; }

// <- 클릭, 한글자씩 삭제
public ICommand DeleteCharCommand { protected set; get; }

// C 클릭시 DisplayText 전체를 지운다.
public ICommand ClearCommand { protected set; get; }

// +, -, *, / 클릭

```

```

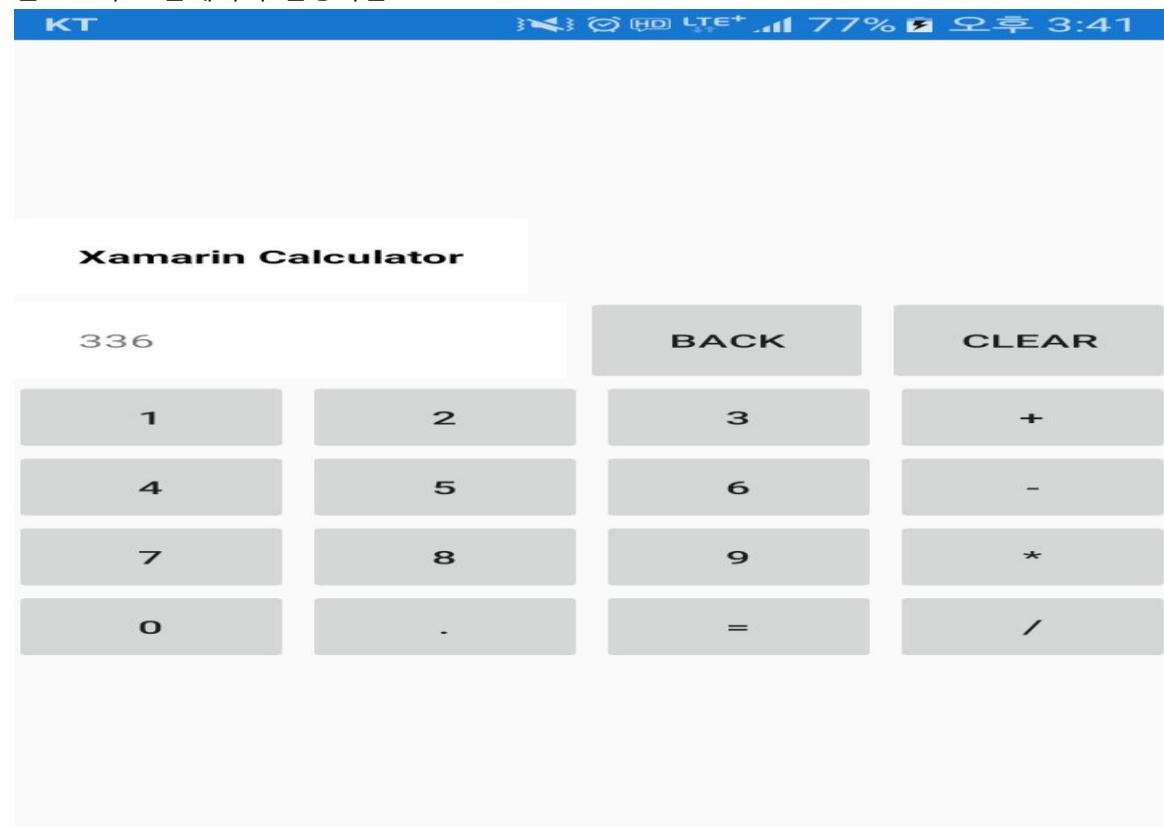
public ICommand OperationCommand { protected set; get; }

// = 클릭
public ICommand CalcCommand { protected set; get; }

protected void OnPropertyChanged(string propertyName)
{
    //이벤트 가입자가 있다면 이벤트를 발생시킨다.
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
}

```

- Calc.Android 프로젝트에서 우측 마우스 클릭 후 시작프로젝트로 설정 후 안드로이드폰을 연결하거나 에뮬레이터로 실행화면 된다.
- 안드로이드 폰에서의 실행화면



- 안드로이드 에뮬레이터에서의 실행화면

Xamarin Calculator

6

BACK

CLEAR

1

2

3

+

4

5

6

-

7

8

9

*

0

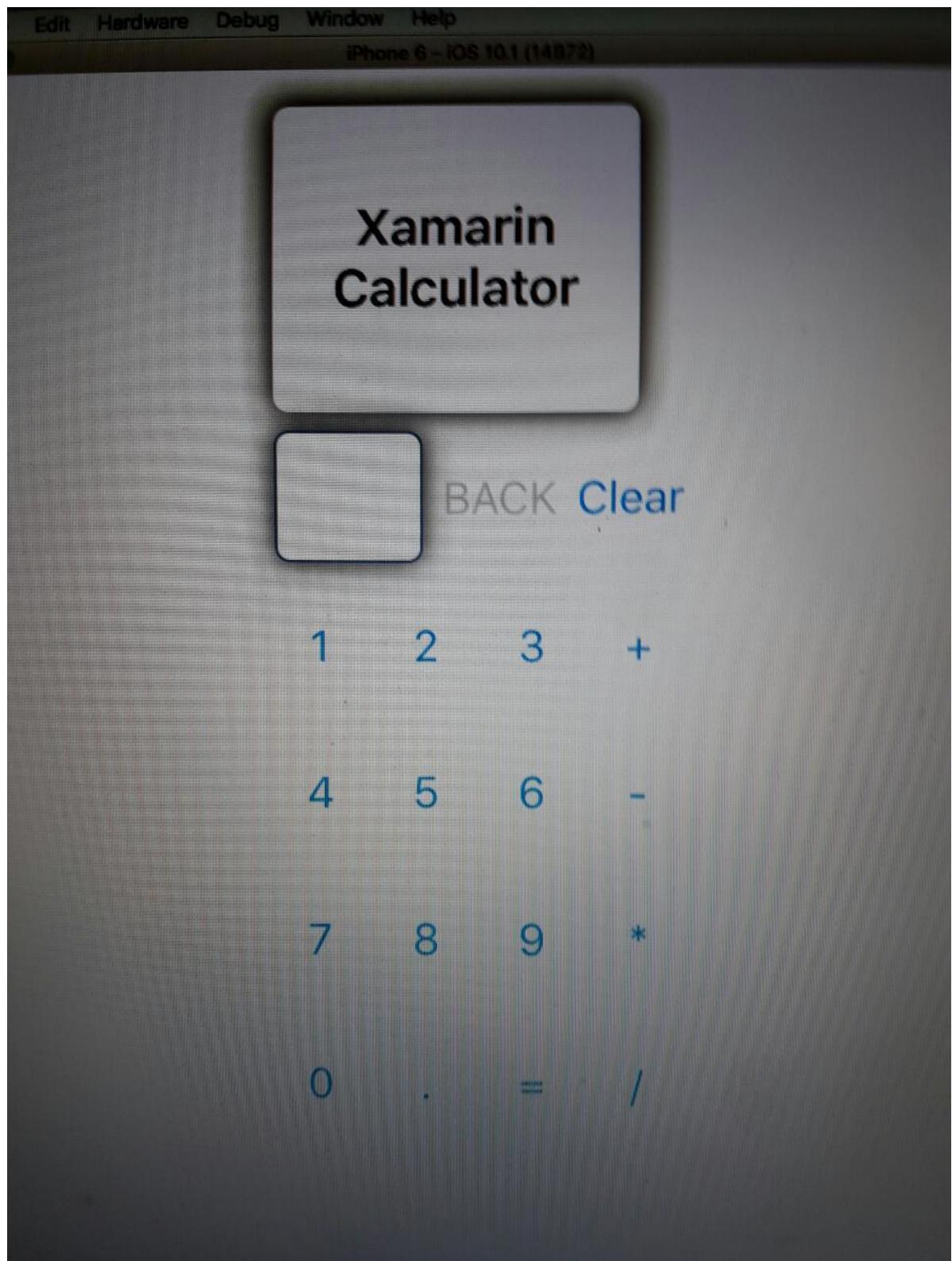
.

=

/



- Calc.iOS 프로젝트에서 우측 마우스 버튼 클릭 후 시작프로젝트로 설정 후 아이폰 시뮬레이터로 실행하면 된다.
- 아이폰 시뮬레이터에서의 실행화면

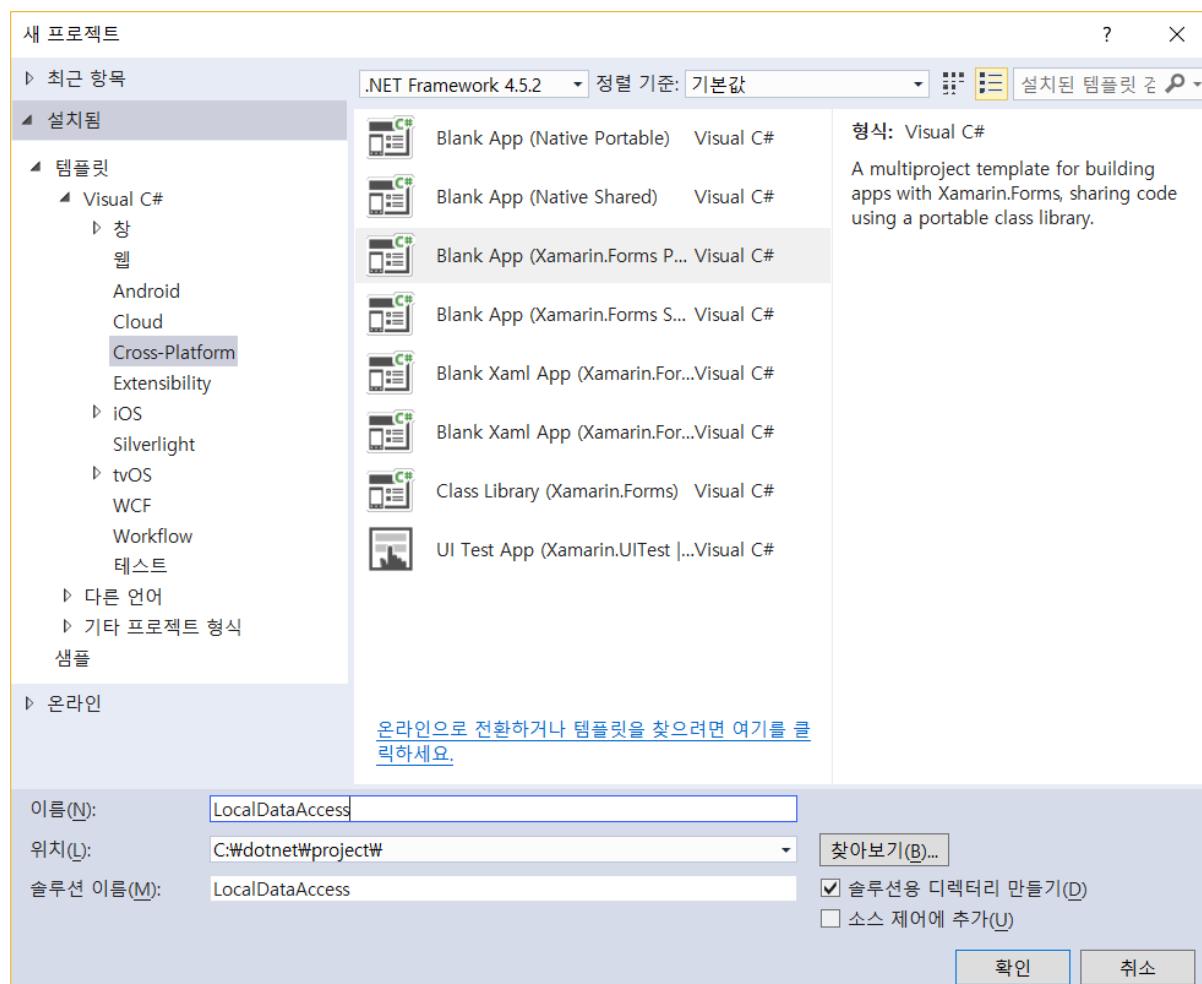


4.8 SQLite.Net with Xamarin.Forms

- Xamarin.Forms 앱은 쉽게 SQLite.NET 계층을 통해 Local SQLite DB에 액세스 가능하다
- SQLite는 오픈소스로 가볍고, 서버가 필요없는 로컬 DB용이며 C#, Linq쿼리로 접근이 가능하다.
- Portable DB이므로 크로스플랫폼에 더욱 강점이 있다. 안드로이드, 아이폰에 있는 DB가 쉽게 Windows등에 이식이 가능하다.
- SQLite 쿠어 엔진은 이미 iOS와 Android에 포함되어 있지만 Windows에는 포함되어 있지 않다. 그러므로 윈도우 개발인 경우 app 패키지에 SQLite 바이너리를 포함해야 한다.

4.8.1 Local SQLite Access Example

- 파일 -> 새로만들기 -> 프로젝트 -> Cross Platform 프로젝트, Blank App(Xamarin.Forms Portable)
- 프로젝트명 : LocalDataAccess



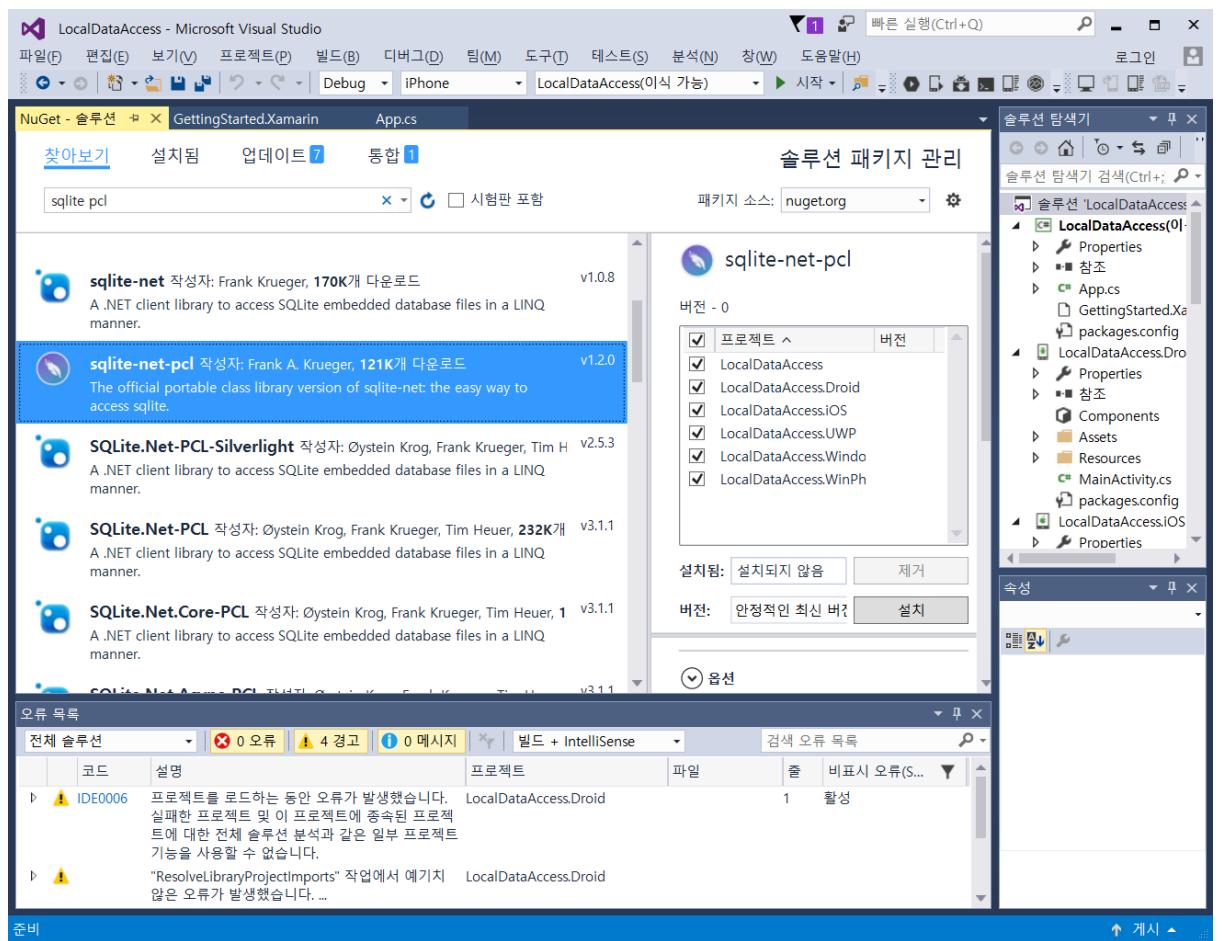
확인 버튼을 클릭하면 비주얼스튜디오는 iOS, Android, UWP, Windows Runtime(WinRT), Windows

Phone, Windows Portable Class Library (PCL) 프로젝트를 생성한다.

■ SQLite NuGet 패키지 인스톨

프로젝트 생성을 했으면 SQLite를 접근하기 위한 방법이 필요한데 .NET 프레임워크에서 DB에 접근하기 여러 라이브러리가 있는데 Xamarin에서 접근해야 하므로 SQLite-Net을 통해 접근할 수 있다. (오픈소스 라이브러리, sqlite-net-pcl이며 NuGet 패키지를 통해 설치)

- 솔루션에서 마우스 우측버튼 -> NuGet 패키지 관리 클릭 **1.X 다운로드**



혹시 Xamarin.Forms 종속성 오류가 나오면 버전을 좀 낮추자. NuGet 관리자 콘솔에서 다음을 실행

```
Install-Package sqlite-net-pcl -Version 1.1.0
```

■ DB접속을 위한 연결 문자열 설정(Connection String)

- Portable Project에서 **IDatabase-Connection.cs** 인터페이스 생성

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LocalDataAccess
{
    public interface IDatabaseConnection
    {
        SQLite.SQLiteConnection DbConnection();
    }
}

```

- `DbConnection`이라는 추상메소드가 선언 되었는데 이 인터페이스 메소드는 각각의 플랫폼에서 구현하고 적절한 접속 문자열을 리턴한다.
- 다음 과정은 각각의 플랫폼 프로젝트에서 클래스를 추가하고 이 인터페이스를 구현하여 적절한 접속 문자열을 리턴하면 된다. (샘플DB : CustomersDb.db3)
- `LocalDataAccess.Android` 프로젝트에 `DatabaseConnection_Android.cs` 파일을 추가하자.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using LocalDataAccess.Android;
using SQLite;

[assembly: Xamarin.Forms.Dependency(typeof(DatabaseConnection_Android))]
namespace LocalDataAccess.Android
{
    public class DatabaseConnection_Android : IDatabaseConnection
    {
        public SQLiteConnection DbConnection()
        {
            var dbName = "CustomersDb.db3";
            string personalFolder = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
            string libraryFolder = Path.Combine(personalFolder, "..", "Library");
        }
    }
}

```

```

        var path = Path.Combine(libraryFolder, dbName);

        if (!Directory.Exists(libraryFolder))
        {
            Directory.CreateDirectory(libraryFolder);
        }
        if (!File.Exists(path))
        {
            File.Create(path);
        }

        return new SQLiteConnection(path);
    }
}

```

Xamarin.Forms.Dependency Attribute는 클래스가 적절한 인터페이스를 구현했음을 나타내고 assembly 키워드와 같이 사용된다. 안드로이드에서는 데이터베이스 파일은 Personal 폴더에 저장되어야 하므로 데이터베이스 경로이름이 파일이름으로 만들어 진다.

아래는 iOS의 예이다.

[LocalDataAccess.iOS\ DataBaseConnection_iOS.cs]

```

using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using SQLite;
using LocalDataAccess.iOS;

[assembly: Xamarin.Forms.Dependency(typeof(DataBaseConnection_iOS))]
namespace LocalDataAccess.iOS
{
    public class DataBaseConnection_iOS : IDatabaseConnection
    {
        public SQLiteConnection DbConnection()
        {
            var dbName = "CustomersDb.db3";
            string personalFolder = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
            string libraryFolder = Path.Combine(personalFolder, "..", "Library");

```

```

var path = Path.Combine(libraryFolder, dbName);

if (!Directory.Exists(libraryFolder))
{
    Directory.CreateDirectory(libraryFolder);
}
if (!File.Exists(path))
{
    File.Create(path);
}

return new SQLiteConnection(path);
}
}
}

```

■ 데이터 모델(Data Model)의 생성

Portable 프로젝트에 Customer.cs 파일을 만들자. (테이블에 매핑되는 모델 클래스)

DB의 데이터가 변했음을 Caller에게 알리기 위해 INotifyPropertyChanged 인터페이스를 상속 받고 [Table("Customers")]이 생략되면 클래스 이름이 테이블 이름이 된다.

```

using SQLite;
using System.ComponentModel;

namespace LocalDataAccess
{
    [Table("Customers")]
    public class Customer : INotifyPropertyChanged
    {
        private int _id;
        [PrimaryKey, AutoIncrement]
        public int Id
        {
            get
            {
                return _id;
            }
            set
            {

```

```
        this._id = value;
        OnPropertyChanged(nameof(Id));
    }
}

private string _companyName;
[NotNull]
public string CompanyName
{
    get
    {
        return _companyName;
    }
    set
    {
        this._companyName = value;
        OnPropertyChanged(nameof(CompanyName));
    }
}

private string _physicalAddress;
[MaxLength(50)]
public string PhysicalAddress
{
    get
    {
        return _physicalAddress;
    }
    set
    {
        this._physicalAddress = value;
        OnPropertyChanged(nameof(PhysicalAddress));
    }
}

private string _country;
public string Country
{
    get
    {
        return _country;
    }
}
```

```

        set
    {
        _country = value;
        OnPropertyChanged(nameof(Country));
    }
}

public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
    this.PropertyChanged?.Invoke(this,
        new PropertyChangedEventArgs(propertyName));
}
}
}

```

■ Data Access 구현

DB 테이블의 데이터를 조작하는 클래스를 생성하자.

먼저 접속문자열을 저장할 변수와 데이터 충돌을 피하기 위해 데이터 조작 시 DB Lock을 구현한 멤버를 추가한다.

```

private SQLiteConnection database;
private static object collisionLock = new object();

```

Xamarin.Forms에서 XAML을 사용하여 UI를 빌드하고, 데이터를 입력/출력할 때 데이터 바인딩으로 인해 이점이 있는데 이 때문에 응용프로그램은 XAML에게 이러한 일을 하도록 데이터를 드러낼 필요가 있고 변화에 대한 통지의 가장 좋은 방법은 ObservableCollection<Customer>을 사용하는 것이다. 즉 XAML 기반의 플랫폼에서 데이터 바인딩을 위한 가장 좋은 방법이다.

```
public ObservableCollection<Customer> Customers { get; set; }
```

생성자에서는 플랫폼 기반의 DbConenction 메소드를 호출한다.

```

public CustomersDataAccess()
{
    database = DependencyService.Get<IDatabaseConnection>().

```

```
    DbConnection();
    database.CreateTable<Customer>();
    this.Customers =
        new ObservableCollection<Customer>(database.Table<Customer>());
    // If the table is empty, initialize the collection
    if (!database.Table<Customer>().Any())
    {
        AddNewCustomer();
    }
}
```

다음은 AddCustomer 메소드의 구현이다.

```
public void AddNewCustomer()
{
    this.Customers.
        Add(new Customer
    {
        CompanyName = "Xamarin Corp...",
        PhysicalAddress = "SANGAM IN SEOUL",
        Country = "KOREA"
    });
}
```

다음은 데이터를 쿼리하는 부분에 대해 살펴보자.

```
public IEnumerable<Customer> GetFilteredCustomers(string countryName)
{
    lock(collisionLock)
    {
        var query = from cust in database.Table<Customer>()
                   where cust.Country == countryName
                   select cust;
        return query.AsEnumerable();
    }
}
public IEnumerable<Customer> GetFilteredCustomers()
{
    lock(collisionLock)
    {
```

```

        return database.Query<Customer>(
            "SELECT * FROM Item WHERE Country = 'Italy'").AsEnumerable();
    }
}

public IEnumerable<Customer> GetFilteredCustomers(string countryName)
{
    lock (collisionLock)
    {
        var query = from cust in database.Table<Customer>()
                    where cust.Country == countryName
                    select cust;
        return query.AsEnumerable();
    }
}
public IEnumerable<Customer> GetFilteredCustomers()
{
    lock (collisionLock)
    {
        return database.Query<Customer>(
            "SELECT * FROM Item WHERE Country = 'KOREA'").AsEnumerable();
    }
}

public Customer GetCustomer(int id)
{
    lock (collisionLock)
    {
        return database.Table<Customer>().
            FirstOrDefault(customer => customer.Id == id);
    }
}

```

CRUD Operation은 다음과 같다.

```

public int SaveCustomer(Customer customerInstance)
{
    lock(collisionLock)

```

```
{  
    if (customerInstance.Id != 0)  
    {  
        database.Update(customerInstance);  
        return customerInstance.Id;  
    }  
    else  
    {  
        database.Insert(customerInstance);  
        return customerInstance.Id;  
    }  
}  
  
public void SaveAllCustomers()  
{  
    lock (collisionLock)  
    {  
        foreach (var customerInstance in this.Customers)  
        {  
            if (customerInstance.Id != 0)  
            {  
                database.Update(customerInstance);  
            }  
            else  
            {  
                database.Insert(customerInstance);  
            }  
        }  
    }  
}  
  
public int DeleteCustomer(Customer customerInstance)  
{  
    var id = customerInstance.Id;  
    if (id != 0)  
    {  
        lock(collisionLock)  
        {  

```

```
        database.Delete<Customer>(id);
    }
}

this.Customers.Remove(customerInstance);
return id;
}

public void DeleteAllCustomers()
{
    lock(collisionLock)
    {
        database.DropTable<Customer>();
        database.CreateTable<Customer>();
    }

    this.Customers = null;
    this.Customers = new ObservableCollection<Customer>
        (database.Table<Customer>());
}
}
```

이식가능(Portable) 프로젝트에 **CustomersDataAccess.cs** 파일을 만들자.

[CustomersDataAccess.cs]

```
using SQLite;

using System.Collections.Generic;
using System.Linq;
using Xamarin.Forms;
using System.Collections.ObjectModel;
namespace LocalDataAccess
{
    public class CustomersDataAccess
    {
        private SQLiteConnection database;
        private static object collisionLock = new object();
        public ObservableCollection<Customer> Customers { get; set; }
        public CustomersDataAccess()
        {
            database =

```

```
DependencyService.Get<IDatabaseConnection>().  
DbConnection();  
database.CreateTable<Customer>();  
this.Customers =  
    new ObservableCollection<Customer>(database.Table<Customer>());  
// If the table is empty, initialize the collection  
if (!database.Table<Customer>().Any())  
{  
    AddNewCustomer();  
}  
}  
public void AddNewCustomer()  
{  
    this.Customers.  
        Add(new Customer  
    {  
        CompanyName = "Company name...",  
        PhysicalAddress = "Address...",  
        Country = "Country..."  
    });  
}  
// Use LINQ to query and filter data  
public IEnumerable<Customer> GetFilteredCustomers(string countryName)  
{  
    // Use locks to avoid database collisions  
    lock(collisionLock)  
    {  
        var query = from cust in database.Table<Customer>()  
                   where cust.Country == countryName  
                   select cust;  
        return query.AsEnumerable();  
    }  
}  
// Use SQL queries against data  
public IEnumerable<Customer> GetFilteredCustomers()  
{  
    lock(collisionLock)  
    {  
        return database.
```

```
Query<Customer>
("SELECT * FROM Item WHERE Country = 'Italy'").AsEnumerable();
}

}

public Customer GetCustomer(int id)
{
    lock(collisionLock)
    {
        return database.Table<Customer>().
            FirstOrDefault(customer => customer.Id == id);
    }
}

public int SaveCustomer(Customer customerInstance)
{
    lock(collisionLock)
    {
        if (customerInstance.Id != 0)
        {
            database.Update(customerInstance);
            return customerInstance.Id;
        }
        else
        {
            database.Insert(customerInstance);
            return customerInstance.Id;
        }
    }
}

public void SaveAllCustomers()
{
    lock(collisionLock)
    {
        foreach (var customerInstance in this.Customers)
        {
            if (customerInstance.Id != 0)
            {
                database.Update(customerInstance);
            }
            else

```

```

        {
            database.Insert(customerInstance);
        }
    }
}

public int DeleteCustomer(Customer customerInstance)
{
    var id = customerInstance.Id;
    if (id != 0)
    {
        lock(collisionLock)
        {
            database.Delete<Customer>(id);
        }
    }
    this.Customers.Remove(customerInstance);
    return id;
}

public void DeleteAllCustomers()
{
    lock(collisionLock)
    {
        database.DropTable<Customer>();
        database.CreateTable<Customer>();
    }
    this.Customers = null;
    this.Customers = new ObservableCollection<Customer>
        (database.Table<Customer>());
}
}
}

```

- 이번에는 UI를 구현하는데 Portable Project에 Forms XAML PAGE로 CustomersPage.xaml 파일을 생성하자.

[CustomersPage.xaml]

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

```

```

    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="LocalDataAccess.CustomersPage">
<ListView x:Name="CustomersView"
    ItemsSource="{Binding Path=Customers}"
    ListView.RowHeight="200">
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout Orientation="Vertical">
<Entry Text="{Binding Id}" IsEnabled="False"/>
<Entry Text="{Binding CompanyName}"/>
<Entry Text="{Binding PhysicalAddress}"/>
<Entry Text="{Binding Country}"/>
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
<ContentPage.ToolbarItems>
<ToolbarItem Name="Add" Activated="OnAddClick"
    Priority="0" Order="Secondary"/>
<ToolbarItem Name="Remove" Activated="OnRemoveClick"
    Priority="1" Order="Secondary"/>
<ToolbarItem Name="Remove all" Activated="OnRemoveAllClick"
    Priority="2" Order="Secondary"/>
<ToolbarItem Name="Save" Activated="OnSaveClick"
    Priority="3" Order="Secondary"/>
</ContentPage.ToolbarItems>
</ContentPage>

```

DataTemplate은 4개의 Entry 컨트롤로 구성되었으며 각각은 Customer 모델 클래스에 바운드 된다. Customer.Id 는 비활성화 되어 있으며 하단의 ToolbarItem은 모든 플랫폼에 가능하며 사용자들의 선택을 편하도록 하며 Priority 속성은 ToolbarItem의 나타나는 순서이다.

CustomerPage.xaml의 코드 비하인드

[CustomerPage.xaml.cs]

```

using System;
using System.Linq;
using Xamarin.Forms;

```

```
namespace LocalDataAccess
{
    public partial class CustomersPage : ContentPage
    {
        private CustomersDataAccess dataAccess;
        public CustomersPage()
        {
            InitializeComponent();
            // An instance of the CustomersDataAccessClass
            // that is used for data-binding and data access
            this.dataAccess = new CustomersDataAccess();
        }
        // An event that is raised when the page is shown
        protected override void OnAppearing()
        {
            base.OnAppearing();
            // The instance of CustomersDataAccess
            // is the data binding source
            this.BindingContext = this.dataAccess;
        }
        // Save any pending changes
        private void OnSaveClick(object sender, EventArgs e)
        {
            this.dataAccess.SaveAllCustomers();
        }
        // Add a new customer to the Customers collection
        private void OnAddClick(object sender, EventArgs e)
        {
            this.dataAccess.AddNewCustomer();
        }
        // Remove the current customer
        // If it exist in the database, it will be removed
        // from there too
        private void OnRemoveClick(object sender, EventArgs e)
        {
            var currentCustomer =
                this.CustomersView.SelectedItem as Customer;
            if (currentCustomer!=null)
            {
```

```

        this.dataAccess.DeleteCustomer(currentCustomer);
    }
}

// Remove all customers
// Use a DisplayAlert object to ask the user's confirmation
private async void OnRemoveAllClick(object sender, EventArgs e)
{
    if (this.dataAccess.Customers.Any())
    {
        var result =
            await DisplayAlert("Confirmation",
                "Are you sure? This cannot be undone",
                "OK", "Cancel");
        if (result == true)
        {
            this.dataAccess.DeleteAllCustomers();
            this.BindingContext = this.dataAccess;
        }
    }
}
}
}

```

- 이식가능(Portable) 프로젝트의 App.cs 파일을 열어 생성자에서 시작 페이지를 수정하자.

,

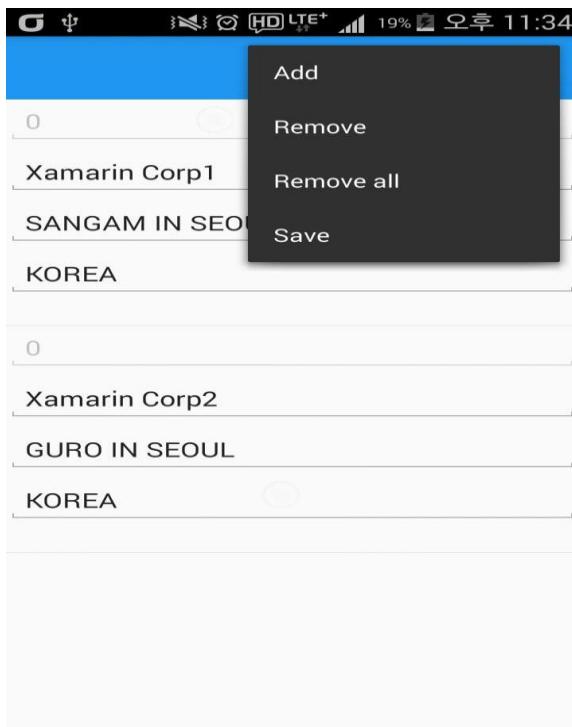
```

MainPage = new NavigationPage(new CustomersPage());
}

protected override void OnStart()
{
    // Handle when your app starts
}

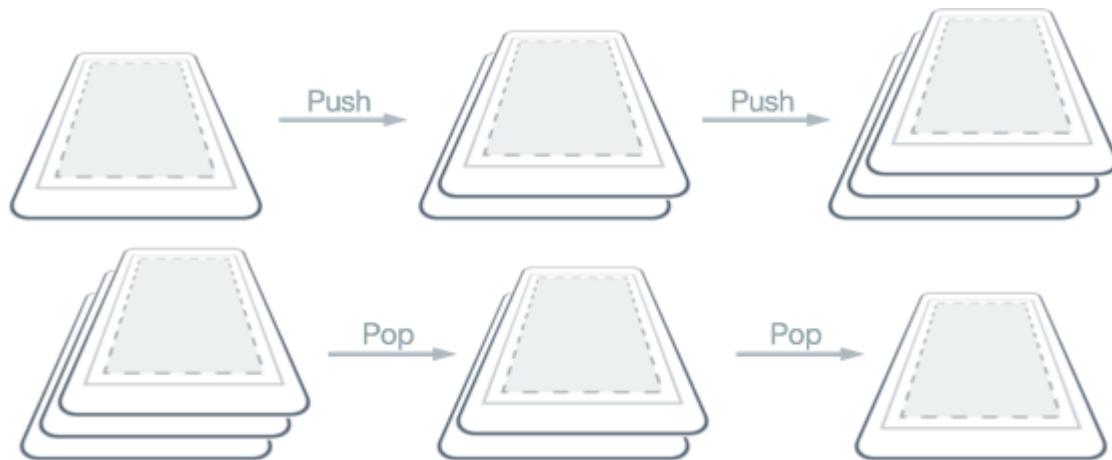
```

- 실행화면

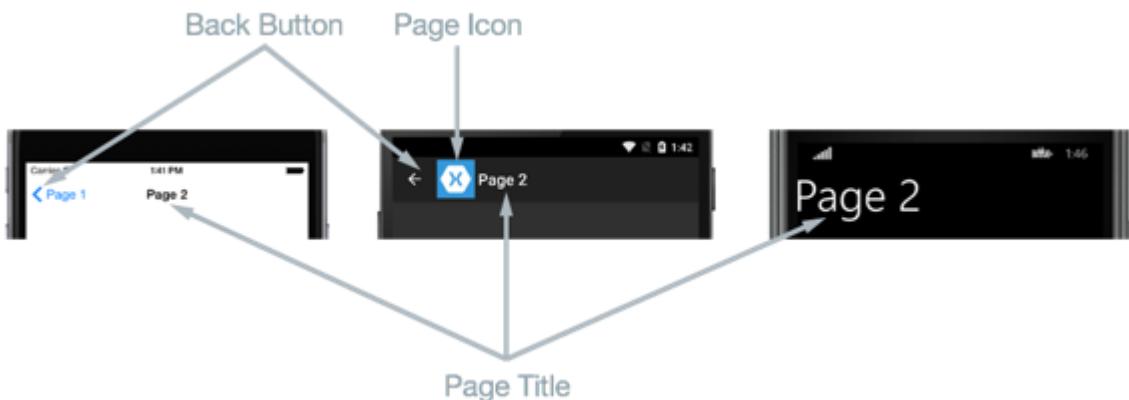


4.8 Hierarchical Navigation

- `NavigationPage` 클래스는 사용자들이 앞, 뒤로 계층적인 page 네비게이션을 할 수 있도록 지원하는데 `Page` 객체의 `Stack(LiFo 구조)`을 구현했다.
- 한 페이지에서 다른 페이지로 이동하기 위해서 `Navigation Stack`에 푸시하면 되고 이 전 페이지로 돌아가기 위해서는 응용프로그램은 `Navigation Stack`에서 현재 페이지를 `Pop`하면 된다.



- 계층적인 네비게이션에서 `NavigationPage` 클래스는 `ContentPage` 객체의 스택을 통해 탐색을 하는데 다음 그림은 각 플랫폼의 `NavigationPage`의 구성요소를 보여준다. 모든 플랫폼에서 `Page.Title` 속성으로 페이지 상단 타이틀을 표시한다.



■ 루트 페이지 생성하기

맨 처음 페이지는 App.cs의 다음 코드를 통해 생성되는데 Page1Xaml ContentPage의 인스턴스가 Navigation Stack의 상단에 푸시되면서 현재의 활성화된 페이지가 되고 또한 현재 응용 프로그램의 Root 페이지가 된다.

참고로 Xamarin.Forms 응용프로그램에서 계층적인 네비게이션을 수행하기 위해서 NavigationPage의 모든 인스턴스는 ContentPage 인스턴스가 되어야 한다.

```
public App ()
{
    MainPage = new NavigationPage (new Page1Xaml ());
}
```



4.8.1 Pushing Pages to the Navigation Stack

Page2Xaml 페이지를 탐색하기 위해서는 현재 페이지 Navigation 속성의 PushAsync 메소드를 통해 Page2Xaml 인스턴스가 전달되어야 한다.

```
async void OnNextPageButtonClicked (object sender, EventArgs e)
{
    // Page2Xaml 페이지가 스택의 상단에 뿌시되면서 Active Page가 된다.
    await Navigation.PushAsync (new Page2Xaml ());
}

// PushAsync 메소드가 호출되면 두개의 이벤트가 발생되는데
// 현재 페이지는 OnDisappearing, 호출되는 페이지는 OnAppearing 이벤트가 발생한다.
// 두 이벤트가 발생된 후 PushAsync Task가 종료된다.
// 두 이벤트는 별로 이점이 없는데 iOS에서 OnDisappearing 이벤트는
// 응용프로그램이 종료될 때 호출된다.
```



4.8.2 Popping Pages from the Navigation Stack

Active Page는 디바이스 또는 스크린의 Back 버튼을 이용해서 Navigation Stack에서 Pop될 수 있는데 프로그래밍적으로 원래 페이지로 돌아가기 위해서 Page2Xaml 인스턴스는 PopAsync 메소드에 의해 호출되어야 한다.

```

// 아래 코드에 의해 Page2Xaml 인스턴스는 스택의 상단에서 Pop되고
// 이전의 Page1Xaml 인스턴스가 Active Page가 된다.
// PopAsync 메소드가 호출되면 두개의 이벤트가 발생되는데
// 현재 Pop 되는 페이지는 OnDisappearing, 스택의 상단으로 올라오는 리턴되는
// 페이지는 OnAppearing 이벤트가 발생한다. 그다음 PopAsync Task가 종료된다.
async void OnPreviousPageButtonClicked (object sender, EventArgs e)
{
    await Navigation.PopAsync ();
}

```

- Navigation 프로퍼티의 PushAsync 및 PopAsync 메소드는 Root 페이지를 제외한 모든 페이지를 Pop 시키기 위해 PopToRootAsync 메소드를 제공하는데 다음 예문을 보자.

```

async void OnRootPageButtonClicked (object sender, EventArgs e)
{
    await Navigation.PopToRootAsync ();
}

```

4.8.3 Passing Data when Navigating

- 한 페이지에서 다른 페이지로 데이터를 넘겨야 하는 경우가 있는데 페이지의 생성자를 통해 넘기거나 새 페이지의 BindingContext에 데이터를 설정하는 방법이 있다.
- 생성자를 통해 넘기기

```

public App ()
{
    MainPage = new NavigationPage (new MainPage (DateTime.Now.ToString ("u")));
}

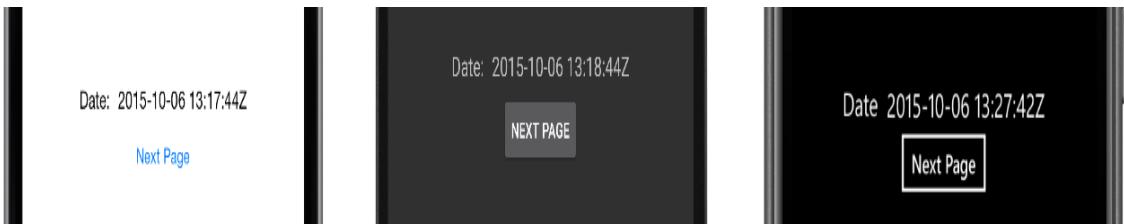
```

데이터를 받는 MainPage에서는 다음과 같이 생성자에서 처리하면 된다.

```

public MainPage (string date)
{
    InitializeComponent ();
    dateLabel.Text = date;
}

```



■ BindingContext를 통해 넘기기

아래 코드는 SecondPage의 BindingContext에 Contact 인스턴스를 세팅 했다.

```
async void OnNavigateButtonClicked (object sender, EventArgs e)
{
    var contact = new Contact()
    {
        Name = "Jane Doe",
        Age = 30,
        Occupation = "Developer",
        Country = "USA"
    };

    var secondPage = new SecondPage ();
    secondPage.BindingContext = contact;
    await Navigation.PushAsync (secondPage);
}
```

아래는 SecondPage에서 넘어오는 Contact 데이터를 출력하기 위해서 XAML코드를 통해 처리한 예문이다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="PassingData.SecondPage"
    Title="Second Page">
<ContentPage.Content>
    <StackLayout HorizontalOptions="Center" VerticalOptions="Center">
        <StackLayout Orientation="Horizontal">
            <Label Text="Name:" HorizontalOptions="FillAndExpand" />
            <Label Text="{Binding Name}" FontSize="Medium" FontAttributes="Bold" />
        </StackLayout>
        ...
        <Button x:Name="navigateButton" Text="Previous Page" />
    </StackLayout>
</ContentPage.Content>
</ContentPage>
```

```
        Clicked="OnNavigateButtonClicked" />
    </StackLayout>
</ContentPage.Content>
</ContentPage>
```

C#코드로 BindingContext의 데이터를 받기 위해서는 다음과 같이 하면 된다.

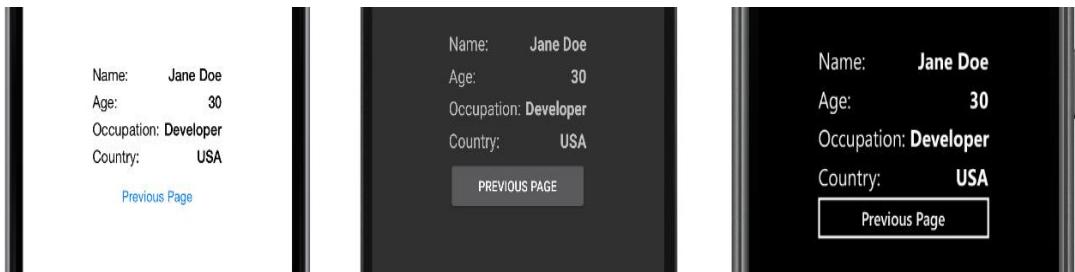
```
public class SecondPageCS : ContentPage
{
    public SecondPageCS ()
    {
        var nameLabel = new Label() {
            FontSize = Device.GetNamedSize (NamedSize.Medium, typeof(Label)),
            FontAttributes = FontAttributes.Bold
        };
        nameLabel.SetBinding (Label.TextProperty, "Name");
        ...
        var navigateButton = new Button { Text = "Previous Page" };
        navigateButton.Clicked += OnNavigateButtonClicked;

        Padding = new Thickness (0, Device.OnPlatform (20, 0, 0), 0, 0);
        Content = new StackLayout {
            HorizontalOptions = LayoutOptions.Center,
            VerticalOptions = LayoutOptions.Center,
            Children = {
                new StackLayout {
                    Orientation = StackOrientation.Horizontal,
                    Children = {
                        new Label{ Text = "Name:", FontSize = Device.GetNamedSize
(NamedSize.Medium, typeof(Label)), HorizontalOptions = LayoutOptions.FillAndExpand },
                        nameLabel
                    }
                },
                ...
                navigateButton
            }
        };
    }
}
```

```

async void OnNavigateButtonClicked (object sender, EventArgs e)
{
    await Navigation.PopAsync ();
}

```



4.8.4 Hierarchical Navigation Example

- Xamarin.Forms를 이용하여 이전에 학습한 페이지 네비게이션 및 Navigation Stack에 대해 이해하도록 하자.
(<https://developer.xamarin.com/samples/xamarin-forms/Navigation/Hierarchical/>)
- Cross-Platform Xamarin.Forms Blank 프로젝트 생성, 프로젝트명을 “WorkingWithNavigation”으로 설정
- MainPage.xaml, MainPage.xaml.cs 파일 삭제
- [App.xaml.cs 수정]

```

using Xamarin.Forms;

namespace WorkingWithNavigation
{
    public partial class App : Application
    {
        public App ()
        {
            MainPage = new NavigationPage (new Page1());
        }
    }
}

```

```
}
```

```
.....
```

```
.....
```

■ [Page1.xaml] : Page1 XAML 코드

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="WorkingWithNavigation.Page1"
              Title="Page 1">
    <ContentPage.Content>
        <StackLayout>
            <Button x:Name="button" Text="Next Page"
                   Clicked="OnNextPageButtonClicked" VerticalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

■ [Page1.xaml.cs] : Page1 XAML의 코드 비하인드

```
using System;
using Xamarin.Forms;

namespace WorkingWithNavigation
{
    public partial class Page1 : ContentPage
    {
        public Page1()
        {
            InitializeComponent();
        }

        async void OnNextPageButtonClicked (object sender, EventArgs e)
        {
            await Navigation.PushAsync (new Page2());
        }
    }
}
```

```
}
```

■ [Page2.xaml] : Page2의 XAML 코드

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="WorkingWithNavigation.Page2"
              Title="Page 2">

    <ContentPage.Content>
        <StackLayout>
            <Button x:Name="nextButton" Text="Next" Page"
                   Clicked="OnNextPageButtonClicked" VerticalOptions="CenterAndExpand" />
            <Button x:Name="previousButton" Text="Previous" Page"
                   Clicked="OnPreviousPageButtonClicked" VerticalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

■ [Page2.xaml.cs] : Page2의 XAML의 코드 비하인드

```
using System;
using Xamarin.Forms;

namespace WorkingWithNavigation
{
    public partial class Page2 : ContentPage
    {
        public Page2 ()
        {
            InitializeComponent ();
        }

        async void OnNextPageButtonClicked (object sender, EventArgs e)
        {
            await Navigation.PushAsync (new Page3());
        }

        async void OnPreviousPageButtonClicked (object sender, EventArgs e)
```

```
        {
            await Navigation.PopAsync ();
        }
    }
}
```

■ [Page2a.xaml] : Page2a의 XAML 코드

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="WorkingWithNavigation.Page2a"
    Title="Page 2a">
    <ContentPage.Content>
        <StackLayout>
            <Button x:Name="nextButton" Text="Next" Page"
Clicked="OnNextPageButtonClicked" VerticalOptions="CenterAndExpand" />
            <Button x:Name="previousButton" Text="Previous" Page"
Clicked="OnPreviousPageButtonClicked" VerticalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

■ [Page2a.xaml.cs] : Page2a의 XAML의 코드 비하인드

```
using System;
using Xamarin.Forms;

namespace WorkingWithNavigation
{
    public partial class Page2a : ContentPage
    {
        public Page2a ()
        {
            InitializeComponent ();
        }

        async void OnNextPageButtonClicked (object sender, EventArgs e)
        {
```

```

        await Navigation.PushAsync (new Page3());
    }

    async void OnPreviousPageButtonClicked (object sender, EventArgs e)
    {
        await Navigation.PopAsync ();
    }
}

```

■ [Page3.xaml] : Page3의 XAML 코드

```

<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="WorkingWithNavigation.Page3"
              Title="Page 3">
    <ContentPage.Content>
        <StackLayout>
            <Button x:Name="previousButton" Text="Previous Page"
                   Clicked="OnPreviousPageButtonClicked" VerticalOptions="CenterAndExpand" />
            <Button x:Name="rootButton" Text="Return to Root Page"
                   Clicked="OnRootPageButtonClicked" VerticalOptions="CenterAndExpand" />
            <Button x:Name="insertButton" Text="Insert Page 2a Before Page 3"
                   Clicked="OnInsertPageButtonClicked" VerticalOptions="CenterAndExpand" />
            <Button x:Name="removeButton" Text="Remove Page 2"
                   Clicked="OnRemovePageButtonClicked" VerticalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>

```

■ [Page3.xaml.cs] : Page3의 XAML의 코드 비하인드

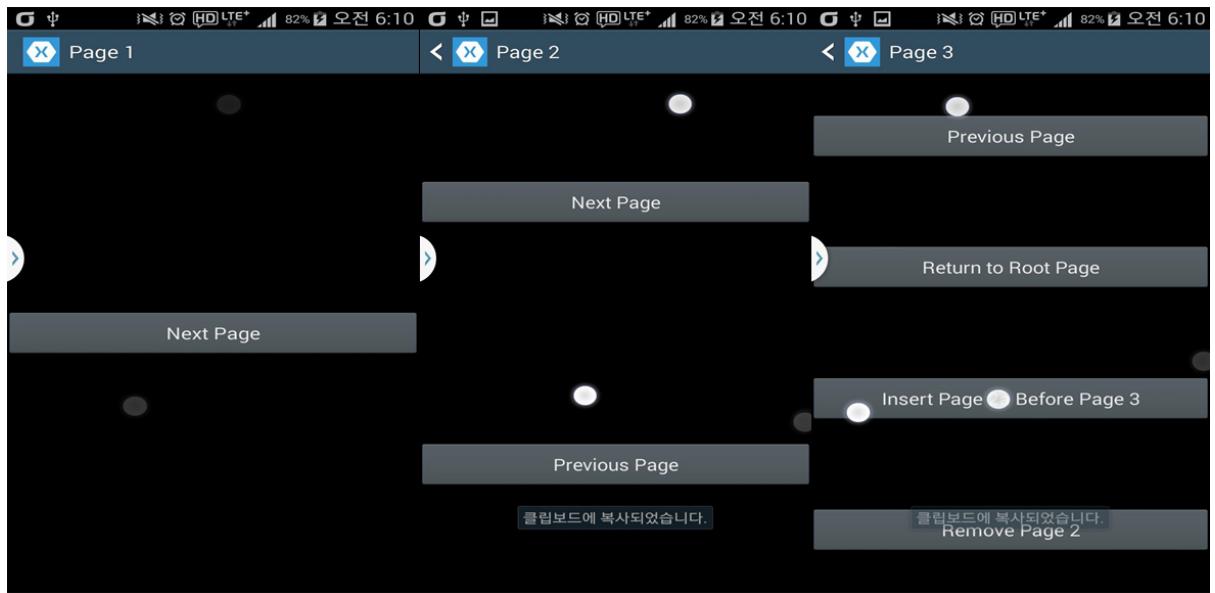
```

using System;
using System.Linq;
using Xamarin.Forms;

namespace WorkingWithNavigation
{
    public partial class Page3 : ContentPage

```

```
{  
    public Page3 ()  
    {  
        InitializeComponent ();  
    }  
  
    async void OnPreviousPageButtonClicked (object sender, EventArgs e)  
    {  
        await Navigation.PopAsync ();  
    }  
  
    async void OnRootPageButtonClicked (object sender, EventArgs e)  
    {  
        await Navigation.PopToRootAsync ();  
    }  
  
    void OnInsertPageButtonClicked (object sender, EventArgs e)  
    {  
        var page2a = Navigation.NavigationStack.FirstOrDefault (p => p.Title  
== "Page 2a");  
        if (page2a == null) {  
            Navigation.InsertPageBefore (new Page2a(), this);  
        }  
    }  
  
    void OnRemovePageButtonClicked (object sender, EventArgs e)  
    {  
        var page2 = Navigation.NavigationStack.FirstOrDefault (p => p.Title ==  
"Page 2");  
        if (page2 != null) {  
            Navigation.RemovePage (page2);  
        }  
    }  
}
```



4.8.5 Login Flow Example

- 로그인 절차, 네비게이션을 간단히 이해할 수 있도록 구성한 프로젝트
(<https://github.com/xamarin/xamarin-forms-samples/tree/master/Navigation/LoginFlow>)
- Cross-Platform Xamarin.Forms Blank 프로젝트 생성, 프로젝트명을 LoginNavigation로 설정
- 시작프로젝트를 LoginNavigation.Android로 설정
- [App.xaml.cs 파일 수정] : 로그인된 경우와 안된 경우에 따라 다른 페이지로 분기

```
using Xamarin.Forms;

namespace LoginNavigation
{
    public partial class App : Application
    {
```

```

public static bool IsUserLoggedIn { get; set; }

public App () {
{
    if (!IsUserLoggedIn) {
        MainPage = new NavigationPage (new LoginPage ());
    } else {
        MainPage = new NavigationPage
LoginNavigation.MainPage ());
//MainPage = new NavigationPage (new LoginNavigation.MainPageCS ());

```

.....
.....

- [Constants.cs] : 로그인을 위한 ID, Password를 상수값으로 보관

```

namespace LoginNavigation
{
    public static class Constants
    {
        public static string Username = "xamarin";
        public static string Password = "password";
    }
}

```

- [LoginPage.xaml] : 로그인이 안되어 있는 경우 Login화면 UI - XAML

```

<?xml version="1.0" encoding="UTF-8"?>
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="LoginNavigation.LoginPage"
    Title="Login">
    <ContentPage.ToolbarItems>
        <ToolbarItem Text="Sign Up" Clicked="OnSignUpButtonClicked" />
    </ContentPage.ToolbarItems>
    <ContentPage.Content>
        <StackLayout VerticalOptions="StartAndExpand">
            <Label Text="Username" />
            <Entry x:Name="usernameEntry" Placeholder="username" />
            <Label Text="Password" />

```

```
<Entry x:Name="passwordEntry" IsPassword="true" />
<Button Text="Login" Clicked="OnLoginButtonClicked" />
<Label x:Name="messageLabel" />
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

■ [LoginPage.xaml.cs] :

await Navigation.PushAsync (new SignUpPage ()) : 회원가입 페이지가 잠시후 비동기적으로 네비게이션 스택의 맨위에 푸시된다.
Navigation.InsertPageBefore (new MainPage (), this) : 스택에 존재하는 현재페이지 앞에 MainPage가 삽입된다.
await Navigation.PopAsync() : 스택 최상단의 페이지를 비동기적으로 꺼낸다.

```
using System;
using Xamarin.Forms;

namespace LoginNavigation
{
    public partial class LoginPage : ContentPage
    {
        public LoginPage ()
        {
            InitializeComponent ();
        }

        async void OnSignUpButtonClicked (object sender, EventArgs e)
        {
            await Navigation.PushAsync (new SignUpPage ());
        }

        async void OnLoginButtonClicked (object sender, EventArgs e)
        {
            var user = new User {
                Username = usernameEntry.Text,
                Password = passwordEntry.Text
            };

            var isValid = AreCredentialsCorrect (user);
        }
    }
}
```

```

        if (isValid) {
            App.IsUserLoggedIn = true;
            //로그인되면 현재 페이지(Login 페이지) 앞에 Main 페이지
            //삽입하고 현재페이지를 꺼내버림, 메인페이지가 스크린에 나타남
            Navigation.InsertPageBefore (new MainPage (), this);
            await Navigation.PopAsync();
        } else {
            messageLabel.Text = "Login failed";
            passwordEntry.Text = string.Empty;
        }
    }

    bool AreCredentialsCorrect (User user)
    {
        return user.Username == Constants.Username && user.Password ==
    Constants.Password;
    }
}

```

- [LoginPageCS.cs] : 로그인이 안되어 있는 경우 Login화면 UI – C# 코드로 구현(안해도 됨)

```

using System;
using Xamarin.Forms;

namespace LoginNavigation
{
    public class LoginPageCS : ContentPage
    {
        Entry usernameEntry, passwordEntry;
        Label messageLabel;

        public LoginPageCS ()
        {
            var toolbarItem = new ToolbarItem {
                Text = "Sign Up"
            };
            toolbarItem.Clicked += OnSignUpButtonClicked;
            ToolbarItems.Add (toolbarItem);

            messageLabel = new Label ();
            usernameEntry = new Entry {
                Placeholder = "username"
            };
            passwordEntry = new Entry {
                IsPassword = true
            };
        }
    }
}

```

```

    };
    var loginButton = new Button {
        Text = "Login"
    };
    loginButton.Clicked += OnLoginButtonClicked;

    Title = "Login";
    Content = new StackLayout {
        VerticalOptions = LayoutOptions.StartAndExpand,
        Children = {
            new Label { Text = "Username" },
            usernameEntry,
            new Label { Text = "Password" },
            passwordEntry,
            loginButton,
            messageLabel
        }
    };
}

async void OnSignUpButtonClicked (object sender, EventArgs e)
{
    await Navigation.PushAsync (new SignUpPageCS ());
}

async void OnLoginButtonClicked (object sender, EventArgs e)
{
    var user = new User {
        Username = usernameEntry.Text,
        Password = passwordEntry.Text
    };

    var isValid = AreCredentialsCorrect (user);
    if (isValid) {
        App.IsUserLoggedIn = true;
        Navigation.InsertPageBefore (new MainPageCS (), this);
        await Navigation.PopAsync ();
    } else {
        messageLabel.Text = "Login failed";
        passwordEntry.Text = string.Empty;
    }
}

bool AreCredentialsCorrect (User user)
{
    return user.Username == Constants.Username &&
    user.Password == Constants.Password;
}
}
}

```

- [MainPage.xaml] : 처음 뜨는 페이지 – XAML로 구현

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="LoginNavigation.MainPage"
    Title="Main Page">
    <ContentPage.ToolbarItems>
        <ToolbarItem Text="Logout" Clicked="OnLogoutButtonClicked" />
    </ContentPage.ToolbarItems>
    <ContentPage.Content>
        <StackLayout>
            <Label Text="Main app content goes here" HorizontalOptions="Center"
                VerticalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

■ [MainPage.xaml.cs]

```
using System;
using Xamarin.Forms;

namespace LoginNavigation
{
    public partial class MainPage : ContentPage
    {
        public MainPage ()
        {
            InitializeComponent ();
        }

        async void OnLogoutButtonClicked (object sender, EventArgs e)
        {
            App.IsUserLoggedIn = false;
            Navigation.InsertPageBefore (new LoginPage (), this);
            await Navigation.PopAsync ();
        }
    }
}
```

■ [SignUpPage.xaml] : 회원가입 – XAML로 구현

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="LoginNavigation.SignUpPage"
    Title="Sign Up">
    <ContentPage.Content>
        <StackLayout VerticalOptions="StartAndExpand">
            <Label Text="Username" />
            <Entry x:Name="usernameEntry" Placeholder="username" />
            <Label Text="Password" />
            <Entry x:Name="passwordEntry" IsPassword="true" />
            <Label Text="Email address" />
            <Entry x:Name="emailEntry" />
            <Button Text="Sign Up" Clicked="OnSignUpButtonClicked" />
            <Label x:Name="messageLabel" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

■ [SignUpPage.xaml.cs]

```
using System;
using System.Linq;
using Xamarin.Forms;

namespace LoginNavigation
{
    public partial class SignUpPage : ContentPage
    {
        public SignUpPage ()
        {
            InitializeComponent ();
        }

        async void OnSignUpButtonClicked (object sender, EventArgs e)
        {
            var user = new User ()
            {
                Username = usernameEntry.Text,
                Password = passwordEntry.Text,
            };
        }
    }
}
```

```

        Email = emailEntry.Text
    };

    // Sign up logic goes here

    var signUpSucceeded = AreDetailsValid (user);
    if (signUpSucceeded) {
        var rootPage = Navigation.NavigationStack.FirstOrDefault ();
        if (rootPage != null) {
            App.IsUserLoggedIn = true;
            Navigation.InsertPageBefore (new MainPage (), 
Navigation.NavigationStack.First ());
            await Navigation.PopToRootAsync ();
        }
    } else {
        messageLabel.Text = "Sign up failed";
    }
}

bool AreDetailsValid (User user)
{
    return (!string.IsNullOrWhiteSpace (user.Username) && !string.IsNullOrWhiteSpace (user.Password) && !string.IsNullOrWhiteSpace (user.Email) && user.Email.Contains ("@"));
}
}

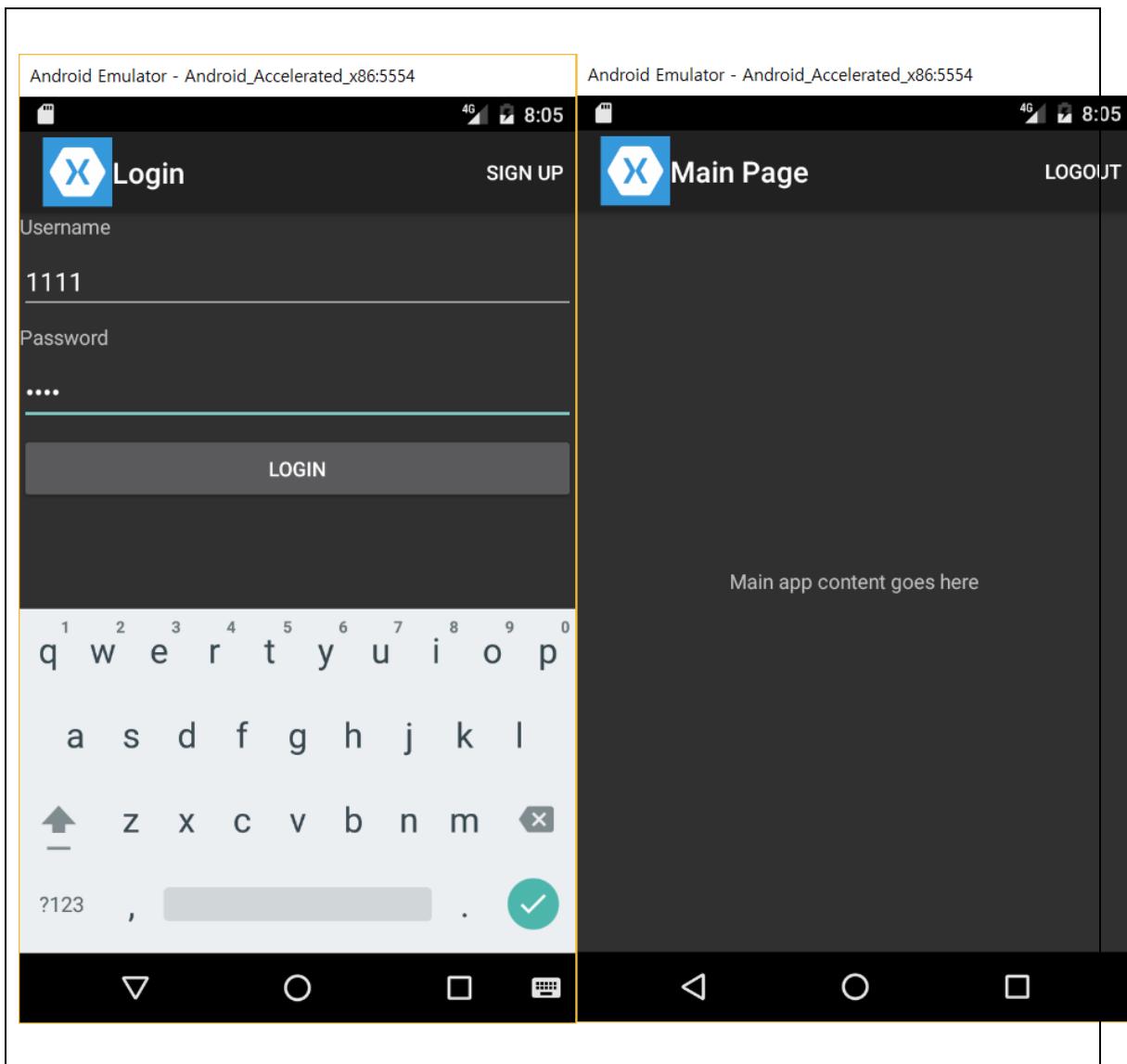
```

■ [User.cs]

```

namespace LoginNavigation
{
    public class User
    {
        public string Username { get; set; }
        public string Password { get; set; }
        public string Email { get; set; }
    }
}

```



Android Emulator - Android_Accelerated_x86:5554



4G



8:06



Sign Up

Username

username

Password

Email address

SIGN UP



5. Xamarin.Forms & REST WebService

5.1 Rest service를 위한 클래스(HttpClient, HttpResponseMessage, HttpContent, HttpWebRequest)

- HttpClient는 Http를 통해 요청을 주고 받을 수 있는 기능을 제공한다. 즉 URL로 제공되는 리소스로 HTTP 요청을 보내고 HTTP 응답을 수신하는 기능을 제공한다.
- 각 요청은 비동기로 전송되며 HttpResponseMessage은 응답으로 받은 HTTP 응답메시지를 나타낸다. 여기에는 상태코드, 헤더, 본문등이 포함되어 있다.
- HttpContent 클래스는 Content-Type 및 Content-Encoding과 같은 HTTP 본문 및 내용 헤더를 나타낸다.
- 데이터의 형식에 따라 ReadAsStringAsync 및 ReadAsByteArrayAsync와 같은 ReadAs 메서드 중 하나를 사용하여 내용을 읽을 수 있다.
- 웹 서비스를 호출하기 위해서는 HttpWebRequest를 사용하는데 특정 URI에 대한 요청 인스턴스를 만들거나 요청 인스턴스에 다양한 HTTP 등록 정보 설정하며 요청에서 HttpWebResponse를 검색하고 응답에서 데이터를 읽을 수 있다.
- 예문

```
var empno= "8877";
var request =
HttpWebRequest.Create(string.Format(@"http://localhost:8080
/emp/{0}", empno));
request.ContentType = "application/json";
request.Method = "GET";

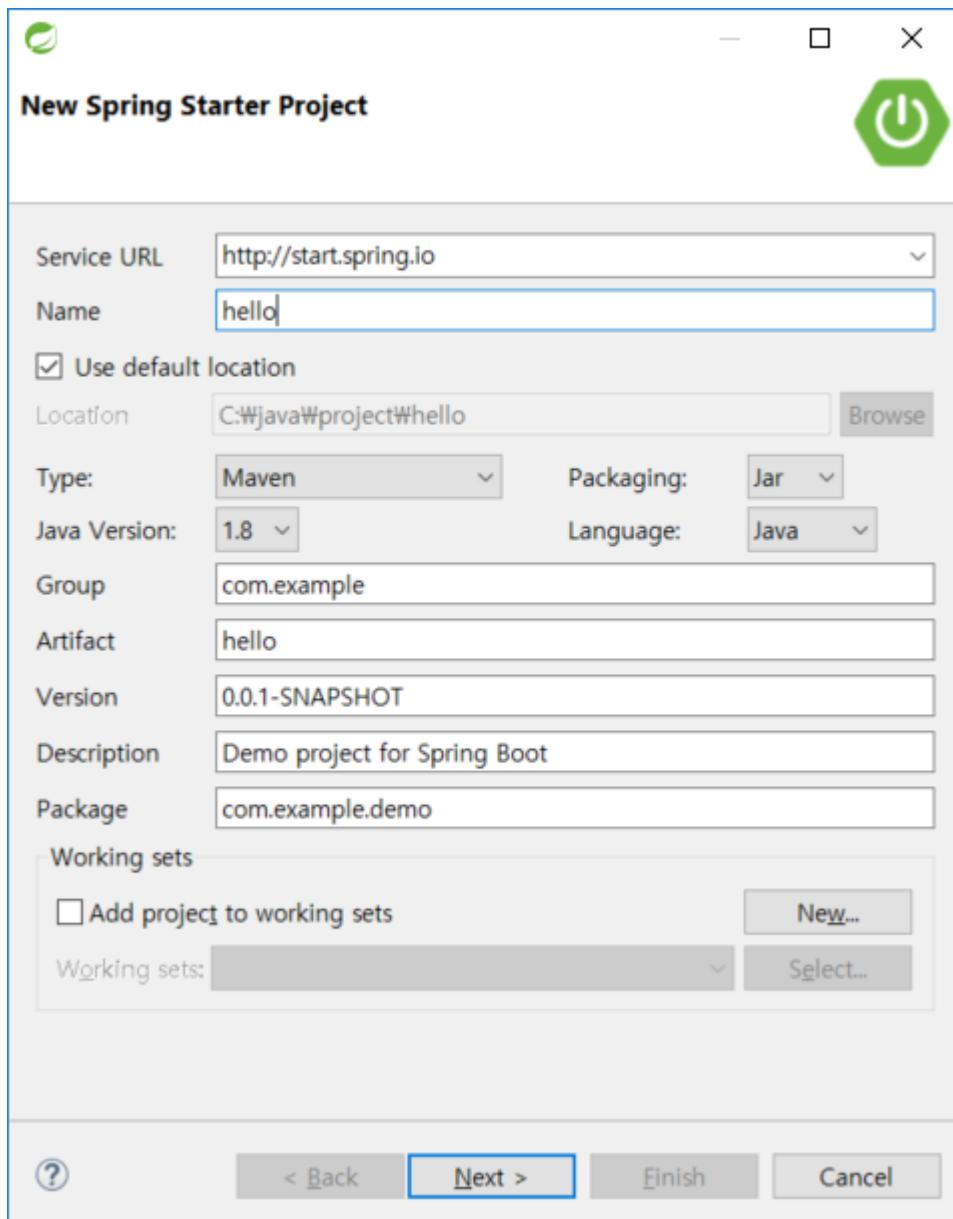
using (HttpWebResponse response = request.GetResponse() as
HttpWebResponse)
{
    if (response.StatusCode != HttpStatusCode.OK)
        Console.Out.WriteLine("Error fetching data. Server
returned status code: {0}", response.StatusCode);
    using (StreamReader reader = new
StreamReader(response.GetResponseStream()))
    {
        var content = reader.ReadToEnd();
        if(string.IsNullOrWhiteSpace(content)) {
            Console.Out.WriteLine("Response
contained empty body...");
        }
        else {
            Console.Out.WriteLine("Response Body:
\r\n {0}", content);
        }
    }

    Assert.NotNull(content);
}
```

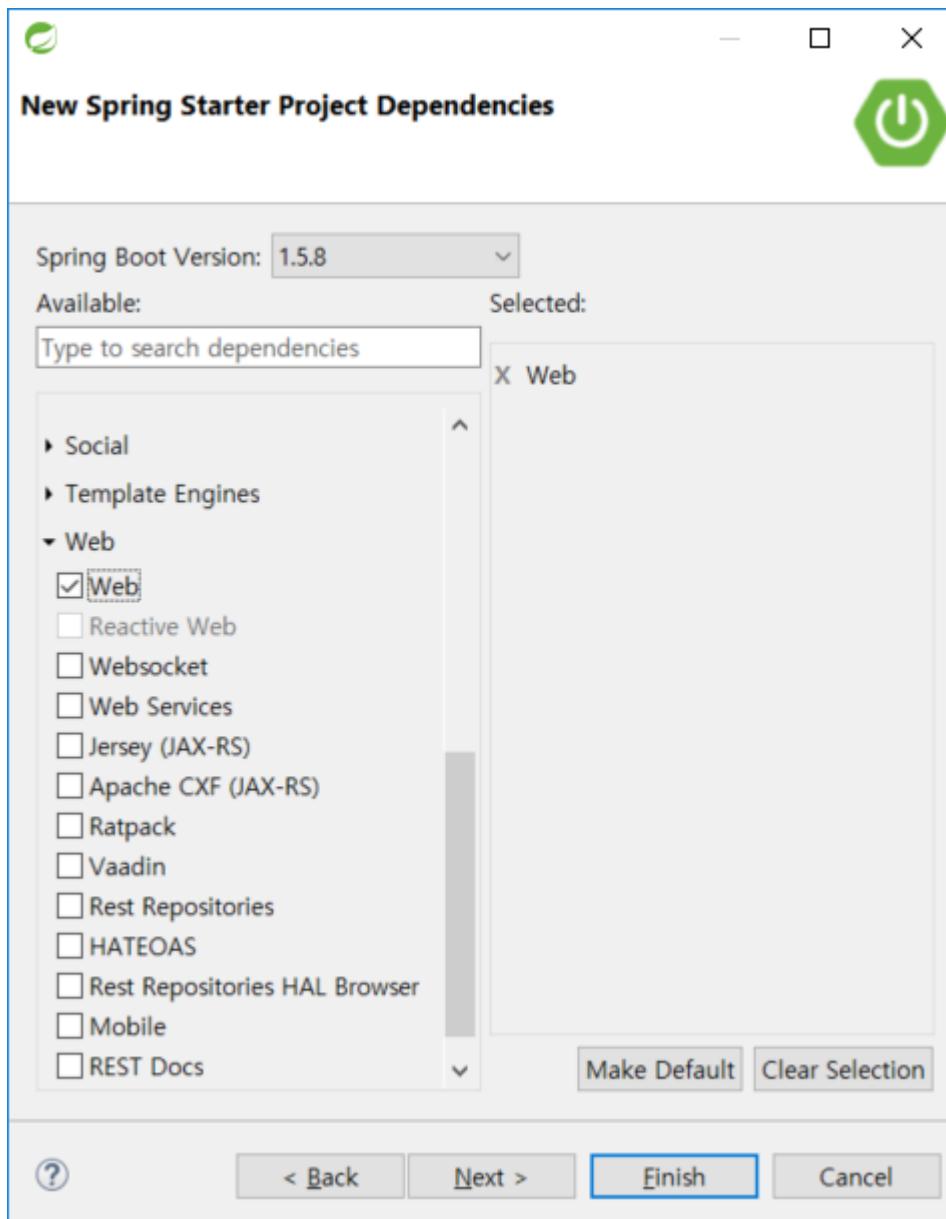
}

5.2 Xamarin.Forms 안드로이드에서 자바기반 스프링 프레임워크(스프링 부트)로 작성한 웹서비스 호출 실습.

- 자바, 스프링 프레임워크(스프링부트) 기반으로 간단히 레스트컨트롤러(RestController)를 만들어 요청을 처리할 hello() 메소드를 만들고 브라우저(실습 노트북의 사설 IP는 192.168.0.189, http://192.168.10.189:8080/hello?name=홍길동)에서 먼저 테스트를 한다. 이것이 확인 되면 Xamarin.Forms로 앱을 만들어 안드로이드 폰(또는 에뮬레이터)을 통해 웹서비스를 호출하는 예제 실습이다.
- 휴대폰에서 실습을 위해 웹서비스를 실행하는 노트북과 같은 네트워크로 되어야 사설 IP로 호출가능 하므로 “휴대폰을 노트북과 같은 네트워크의 Wifi로 설정”하고 테스트 하면 폰에서 스프링 기반 웹 서비스를 호출할 수 있다.
- HttpClient는 Http를 통해 요청을 주고 받을 수 있는 기능을 제공한다. 즉 URL로 제공되는 리소스로 HTTP 요청을 보내고 HTTP 응답을 수신하는 기능을 제공한다. 본 예제에서는 HttpClient를 이용하여 요청을 보내고 ReadAsStringAsync 메소드로 응답을 문자열로 읽어서 휴대폰 화면에 출력한다.
- STS 또는 이클립스에서 Spring MVC Rest 형식으로 간단히 스프링 컨트롤러를 만들자.
- STS를 오픈하여 File >> New >> Project >> Spring Stater Project(스프링 부트)를 선택



- 다음 화면에서 Web >> Web 선택 후 Finish 클릭



- com.example.demo 패키지에서 스프링 컨트롤러를 작성하자.

- [HelloController.java]

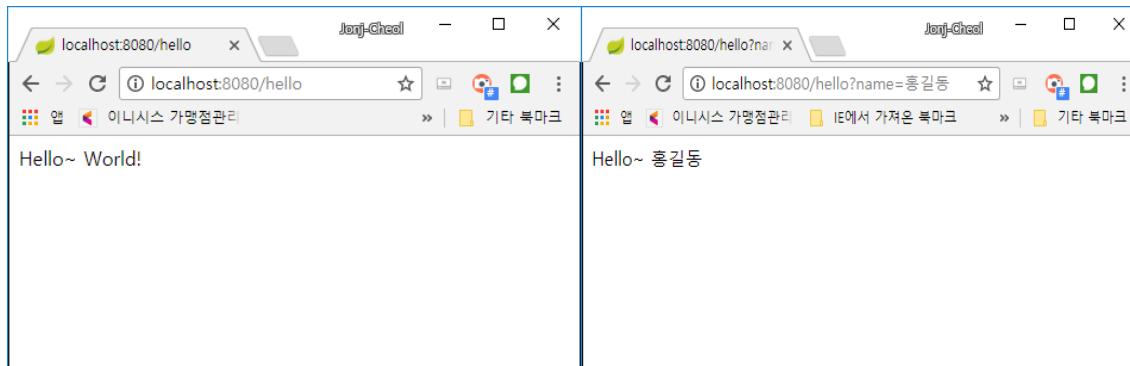
```
package com.example.demo;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

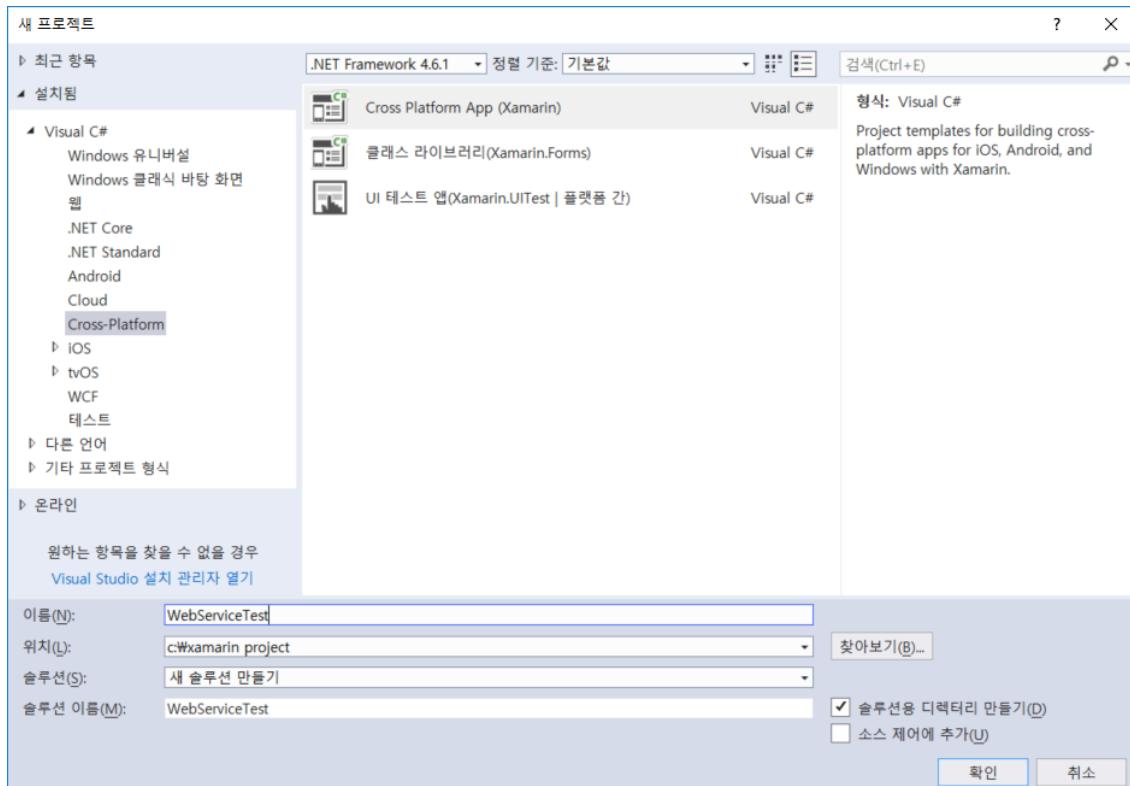
@RestController
public class HelloController {
    @RequestMapping("/hello")
    public String hello(@RequestParam(value="name",
defaultValue="World!") String name) {
        return "Hello~ " + name;
    }
}
```

```
}
```

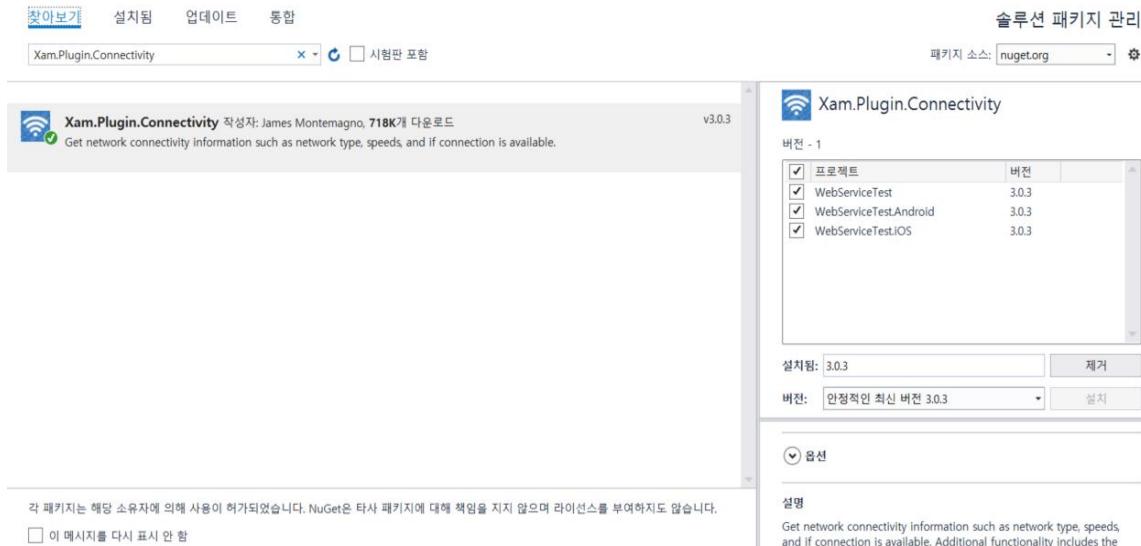
■ 실행 후 브라우저에서 확인



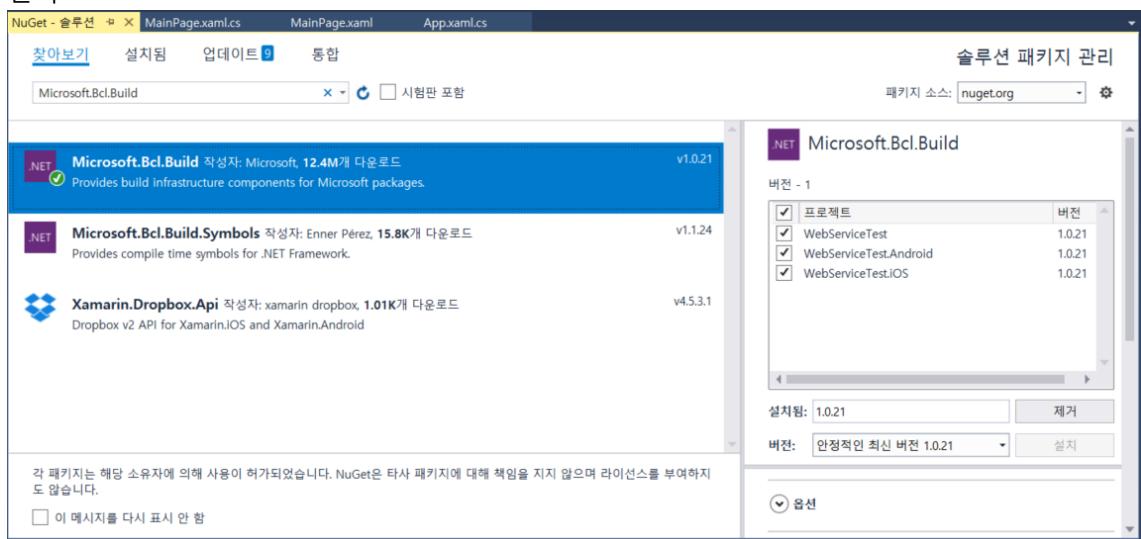
■ 비주얼 스튜디오 2017 커뮤니티에서 Xamarin.Forms 크로스 플랫폼 앱, PCL 프로젝트를 생성하자.



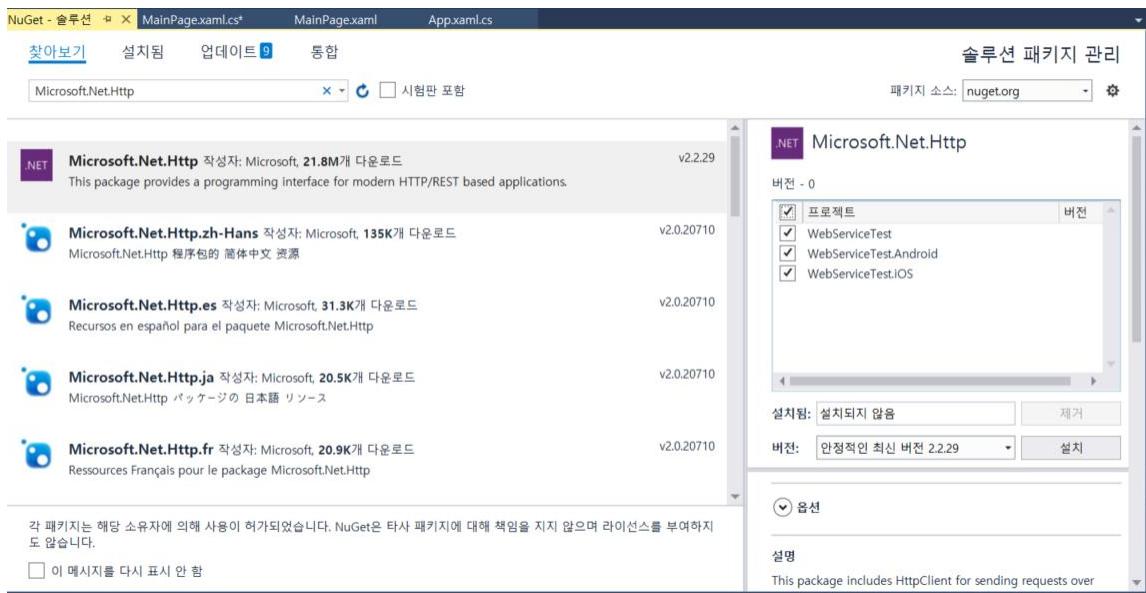
■ Xamarin.Forms에서 인터넷 연결을 체크할 수 있는 "Xam.Plugin.Connectivity" Nuget 패키지를 설치하자.



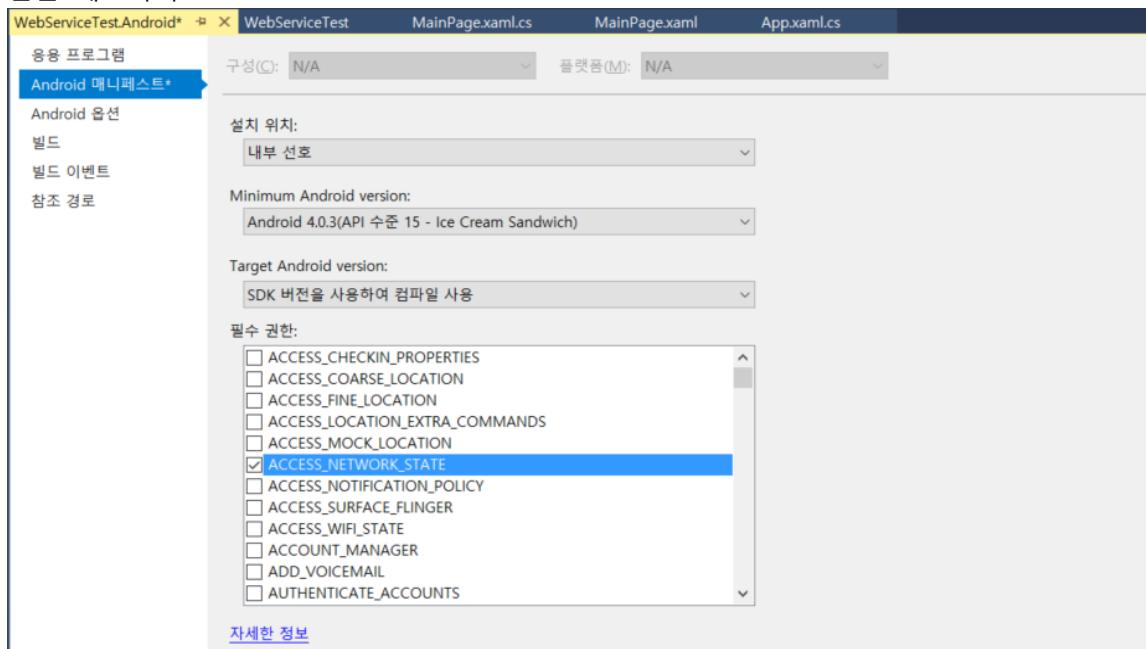
- Xamarin.Forms에서 HttpClient를 사용할 예정인데 직접 다를 수는 없으므로 "Microsoft.Net.Http" Nuget 패키지를 추가해야 한다. "Microsoft.Net.Http"를 포함시키기 위해 솔루션 >> 우측마우스 >> 솔루션용 NuGet 패키지 관리에서 "Microsoft.Bcl.Build"를 추가 한다.



- 솔루션 >> 우측마우스 >> 솔루션용 NuGet 패키지 관리에서 "Microsoft.Net.Http"를 추가 한다.



- 안드로이드 플랫폼에서 테스트를 위해 안드로이드쪽 프로젝트에서 Properties를 더블클릭 하여 “Android 매니페스트옵션” >> “필수권한”에서 “ACCESS_NETWORK_STATE”, “INTERNET” 권한을 체크하자.



- 휴대폰쪽의 UI를 작성하자. [MainPage.xaml]

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:WebServiceTest"
    x:Class="WebServiceTest.MainPage">
    <StackLayout>
        <Label Text="이름을 입력하세요?" />

```

```

<Entry x:Name="txtName" Text="" />
<Button x:Name="button1" Text="스프링 웹서비스 호출하기"
Clicked="Button1_Click"/>
<Label x:Name="label1"/>
</StackLayout>

</ContentPage>

```

- 버튼클릭 이벤트에서 자바, 스프링쪽의 웹서비스를 호출하고 결과를 받는 로직을 추가하자.

- [MainPage.xaml.cs]

```

using Plugin.Connectivity;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace WebServiceTest
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private async void Button1_Click(object sender, EventArgs e)
        {
            string url = "http://192.168.0.189:8080/hello";
            if (CrossConnectivity.Current.IsConnected)
            {
                var client = new System.Net.Http.HttpClient();

                if (!string.IsNullOrEmpty(txtName.Text)) url += "?name=" + txtName.Text;
                var response = await client.GetAsync(url);

                string helloString = await response.Content.ReadAsStringAsync();

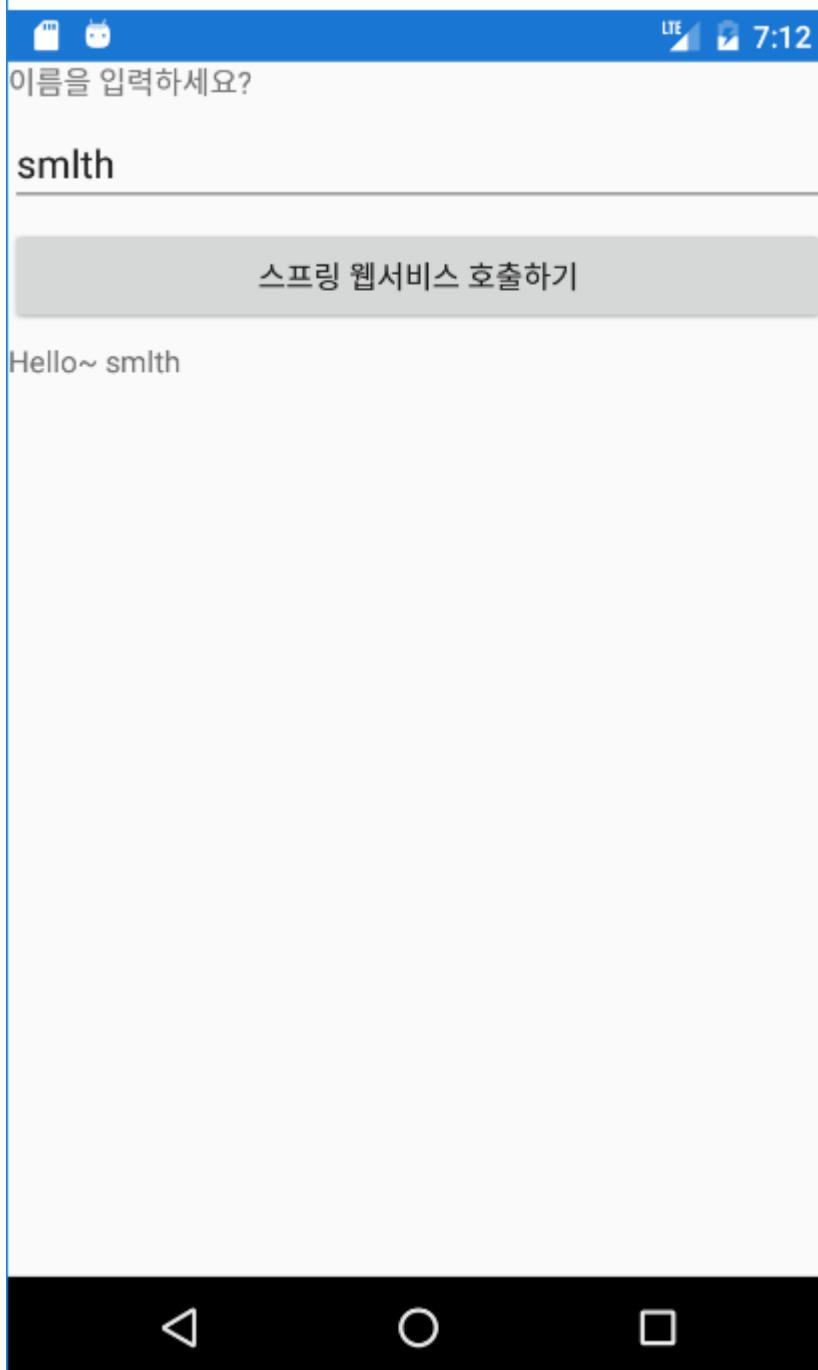
                if (helloString != "") 
                {

```

```
        label1.Text = helloString;
    }
else
{
    await DisplayAlert("경고", "데이터가 없습니다.", "OK");
}
else
{
    await DisplayAlert("경고", "네트워크 에러! 인터넷 연결 확인 바랍니다.", "OK");
}
}
```

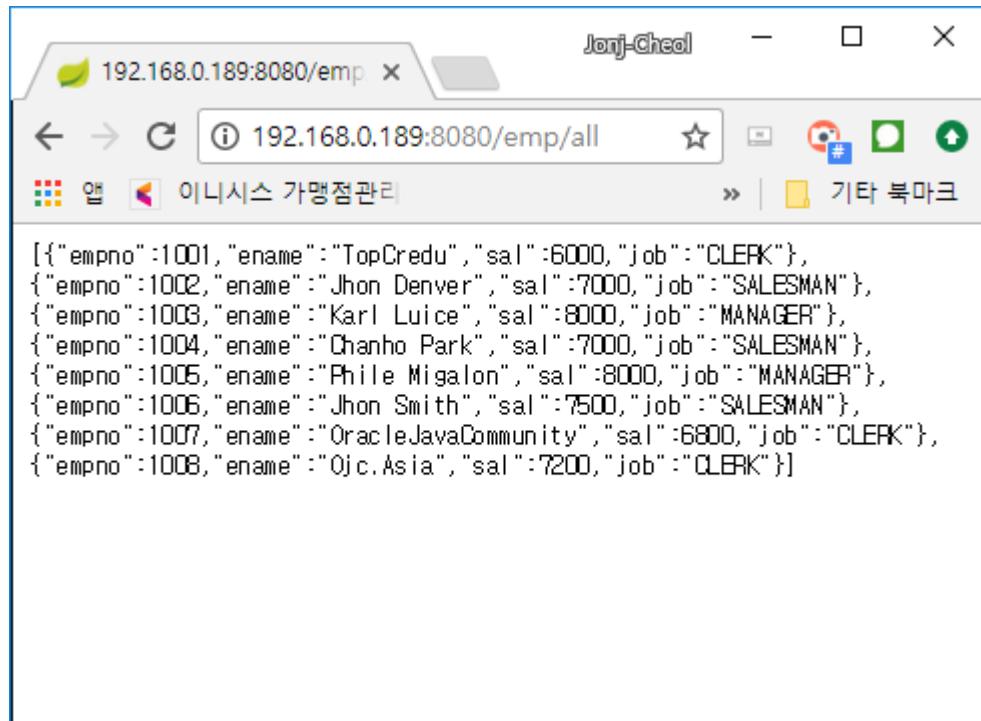
■ 실행화면

Android Emulator - VisualStudio_android-23_x86_phone:5...



5.3 자마린 앱에서 스프링프레임워크/스프링부트 RESTful기반 웹서비스 Call 실습, JSON 파싱하기[웹서비스는 자바,스프링으로 모바일 앱은 자마린으로!]

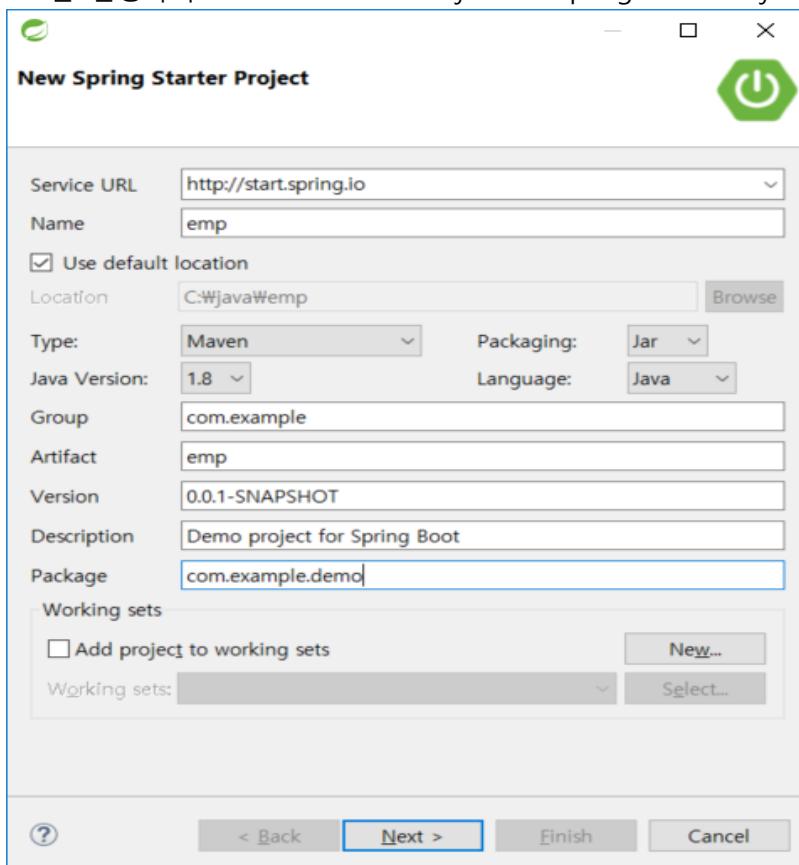
- 웹은 자바, 스프링 프레임워크를 기반으로 구축된 곳이 많다. 왜냐하면 성능도 좋고 데이터베이스를 다루기에는 용이하기 때문이다. (스프링 프레임워크 + 마이바티스, Spring Data JPA, ORM 기술 등)
- 자마린 앱에서 원격의 오라클, MS-SQL, MySQL등의 DB에 직접 접속하여 개발하고 싶은 개발자도 있겠지만 이는 적절하지 못한 방법이며 지원 역시 빈약하기 때문에 문제에 직면할 확률이 높다. 웹서비스를 만들고 이를 자마린 앱에서 호출하는 것이 현명할 것 같다.
- 단넷 기반의 자마린 앱 실습 이지만 스프링프레임워크(스프링 부트) 기반으로 RESTful WebService를 만들고 이를 자마린 앱 응용프로그램에서 호출하여 결과(응답)를 JSON을 받아 파싱하여 데이터 바인딩을 이용하여 휴대폰 화면에 출력해보자. (단넷 개발자 이지만 이번 회에 자바 스프링 프레임워크 MVC기반의 RESTful WebService를 경험해 보자. 별것 아니다.)
- 자바, 스프링 프레임워크(스프링부트) 기반으로 간단히 RESTful 기반 웹서비스를 만드는데, 스프링의 레스트컨트롤러(RestController)를 이용하여 CRUD 기능의 컨트롤러, DAO클래스를 만들고 브라우저에서 먼저 테스트를 한다. 이것이 확인되면 Xamarin.Forms로 앱을 만들어 안드로이드 폰(또는 에뮬레이터)을 통해 웹서비스를 호출하는 CRUD를 테스트 하는 실습이다.
- CRUD 전체 기능을 테스트 하는 것은 여러분들께 맡기고 본 실습에서는 전체 사원데이터 검색, 한명의 사원 검색하는 정도를 구현해 보기로 한다.
- 스프링 프레임워크(스프링부트)에서 RESTFule기반으로 구현한 실행화면을 먼저 보자.
- [전체 사원 조회]



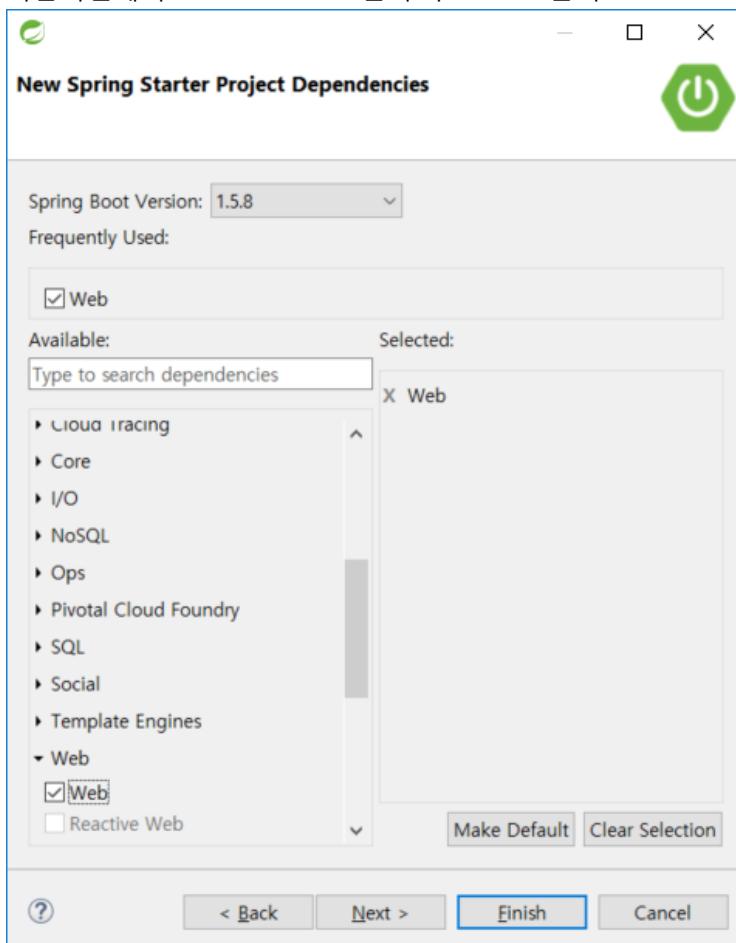
- 한명의 사원 조회



- 자바, 스프링쪽 실습은 Spring Tool Suite(STS)에서 하는데 JDK가 설치되어야 하며, 구글에서 “STS Download”라고 입력하면 다운받아서 압축풀고 간단하게 실행 할 수 있다.
- STS를 실행하여 File >> New >> Project >> Spring Starter Project(스프링 부트)를 선택.



- 다음화면에서 Web >> Web 선택 후 Finish 클릭.



- com.example.demo 패키지에 Emp.java, EmpDAO.java, EmpController.java 3개의 파일을 생성하자.

- [Emp.java]

```
package com.example.demo;

public class Emp {
    private Long empno;
    private String ename;
    private Long sal;
    private String job;

    public Emp(Long empno, String ename, Long sal, String job) {
        this.empno = empno;
        this.ename = ename;
        this.sal = sal;
        this.job = job;
    }

    public Long getEmpno() {
        return empno;
    }
}
```

```

}

public void setEmpno(Long empno) {
    this.empno = empno;
}

public String getEname() {
    return ename;
}

public void setEname(String ename) {
    this.ename = ename;
}

public Long getSal() {
    return sal;
}

public void setSal(Long sal) {
    this.sal = sal;
}

public String getJob() {
    return job;
}

public void setJob(String job) {
    this.job = job;
}
}

```

■ [EmpDAO.java]

```

package com.example.demo;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;

@Repository
public class EmpDAO {
    private static List<Emp> emps;
    {

        emps = new ArrayList<Emp>();
        emps.add(new Emp(1001L, "TopCredu", 6000L, "CLERK"));
        emps.add(new Emp(1002L, "Jhon Denver", 7000L,
"SALESMAN"));
    }
}

```

```

        emps.add(new Emp(1003L, "Karl Luice", 8000L,
"MANAGER"));
        emps.add(new Emp(1004L, "Chanho Park", 7000L,
"SALESMAN"));
        emps.add(new Emp(1005L, "Phile Migalon", 8000L,
"MANAGER"));
        emps.add(new Emp(1006L, "Jhon Smith", 7500L,
"SALESMAN"));
        emps.add(new Emp(1007L, "OracleJavaCommunity", 6800L,
"CLERK"));
        emps.add(new Emp(1008L, "Ojc.Asia", 7200L, "CLERK"));
    }

    public List<Emp> getEmps() {
        return emps;
    }

    public Emp getEmp(Long empno) {
        for(Emp e : emps) {
            if (e.getEmpno().equals(empno)) return e;
        }
        return null;
    }

    public Emp createEmp(Emp e) {
        emps.add(e);
        return e;
    }

    public Long deleteEmp(Long empno) {
        for(Emp e : emps) {
            if (e.getEmpno().equals(empno)) {
                emps.remove(e);
                return empno;
            }
        }
        return null;
    }

    public Emp updateEmp(Long empno, Emp e) {
        for (Emp emp : emps) {
            if (emp.getEmpno().equals(empno)) {
                e.setEmpno(emp.getEmpno());
                emps.remove(emp);
                emps.add(e);
                return e;
            }
        }
    }
}

```

```
        return null;
    }
}
```

■ [EmpController.java]

```
package com.example.demo;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmpController {

    @Autowired
    private EmpDAO empDAO;

    // http://localhost:8080/emp/all
    @GetMapping("/emp/all")
    public List<Emp> getEmps() {
        System.out.println();
        return empDAO.getEmps();
    }

    // http://localhost:8080/emp/1
    @GetMapping("/emp/{empno}")
    public ResponseEntity<Emp> getEmp(@PathVariable("empno") Long empno) {
        Emp e = empDAO.getEmp(empno);
        if (e == null) {
            return new
ResponseEntity<Emp>(HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<Emp>(e, HttpStatus.OK);
    }

    // http://localhost:8080/emp/create
```

```

    @PostMapping(value="/emp/create")
    public ResponseEntity<Emp> createEmp(@RequestBody Emp e) {
        empDAO.createEmp(e);
        return new ResponseEntity<Emp>(e, HttpStatus.OK);
    }

    // http://localhost:8080/emp/delete/1
    @DeleteMapping("/emp/delete/{empno}")
    public ResponseEntity<Long> deleteEmp(@PathVariable Long empno) {
        if (null == empDAO.deleteEmp(empno)) {
            return new
        ResponseEntity<Long>(HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<Long>(empno, HttpStatus.OK);
    }

    // http://localhost:8080/emp/update/1
    @PutMapping("/emp/update/{empno}")
    public ResponseEntity<Emp> updateCustomer(@PathVariable Long empno, @RequestBody Emp e) {

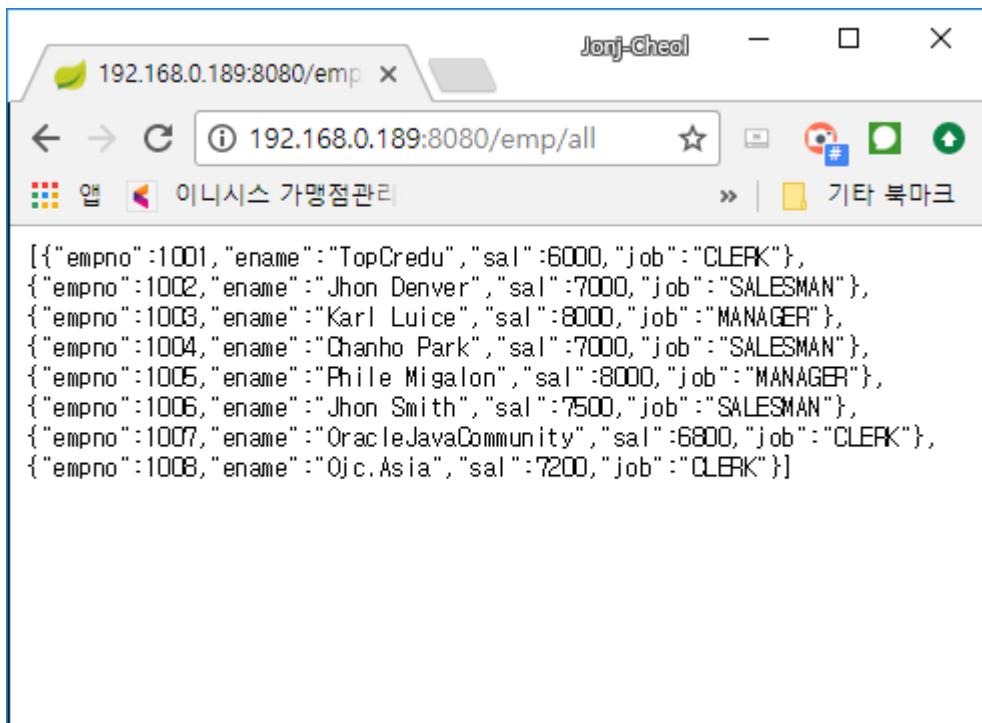
        e = empDAO.updateEmp(empno, e);

        if (null == e) return new
        ResponseEntity<Emp>(HttpStatus.NOT_FOUND);

        return new ResponseEntity<Emp>(e, HttpStatus.OK);
    }
}

```

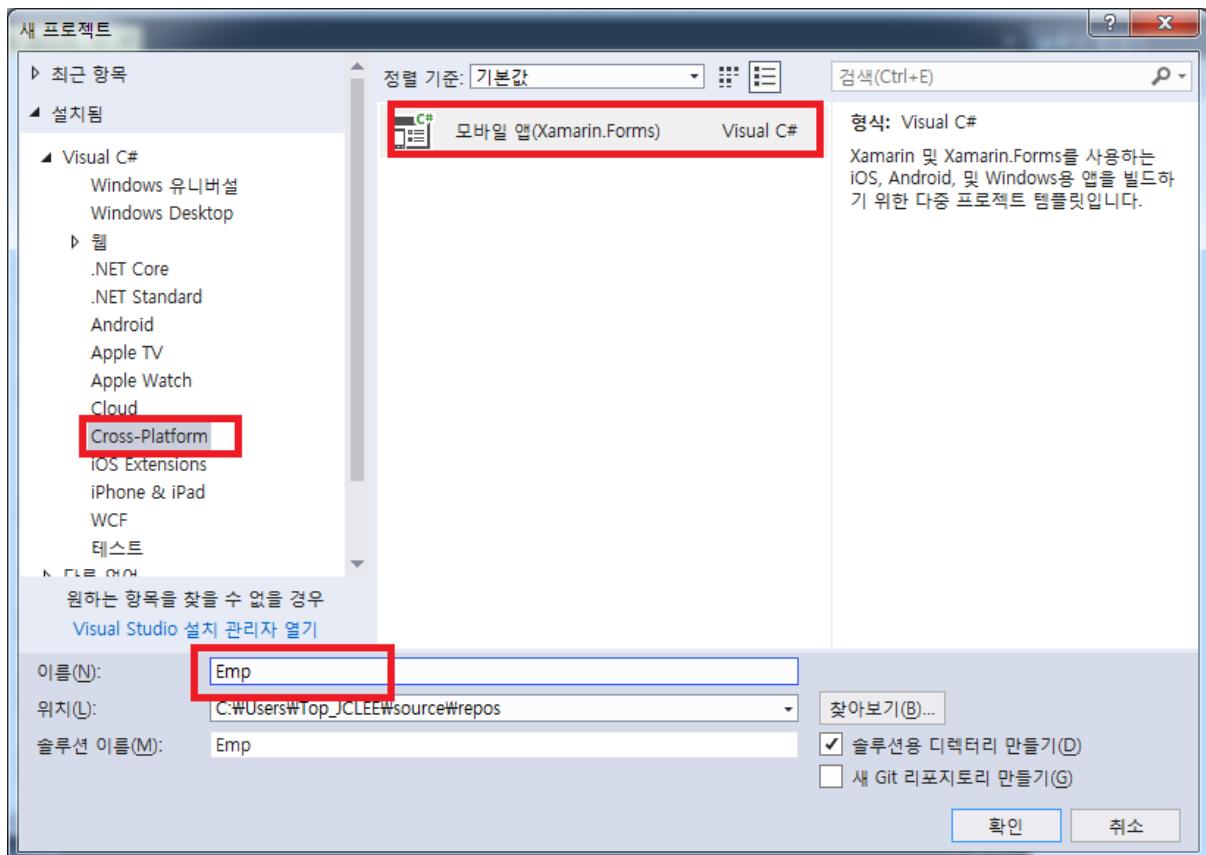
- EmpApplication.java에서 마우스 우측버튼 >> Run As >> Java Application을 클릭 하여 실행 후 브라우저에서 확인하자.
- [전체 사원 조회]



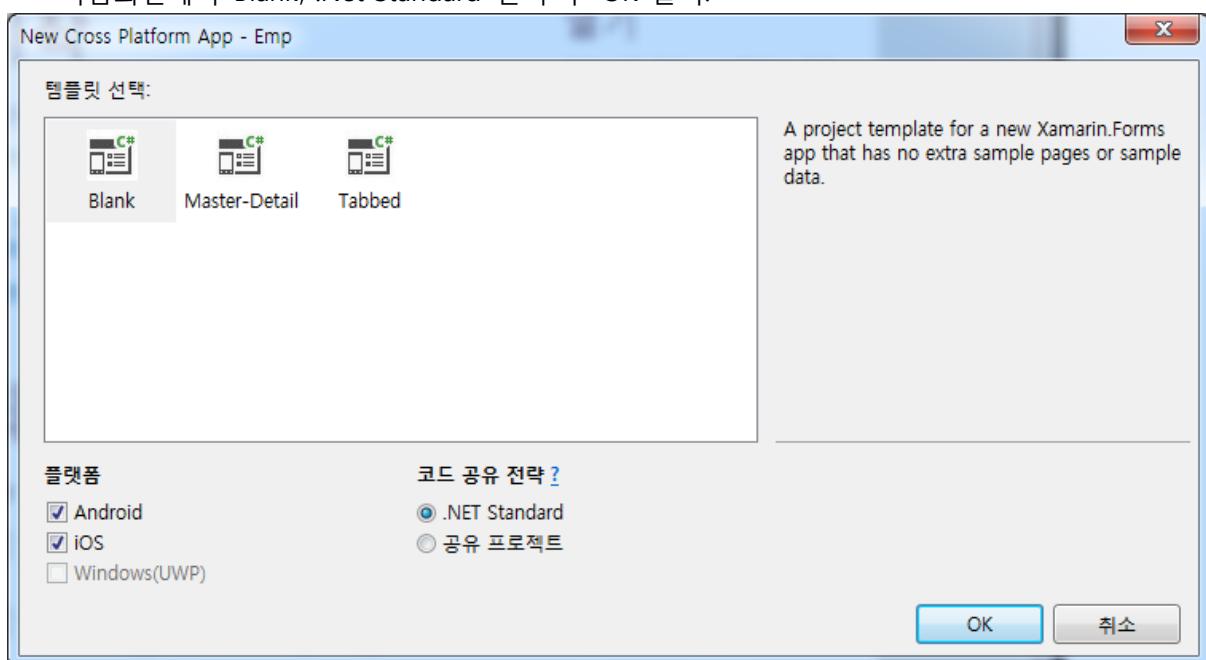
■ 한명의 사원 조회



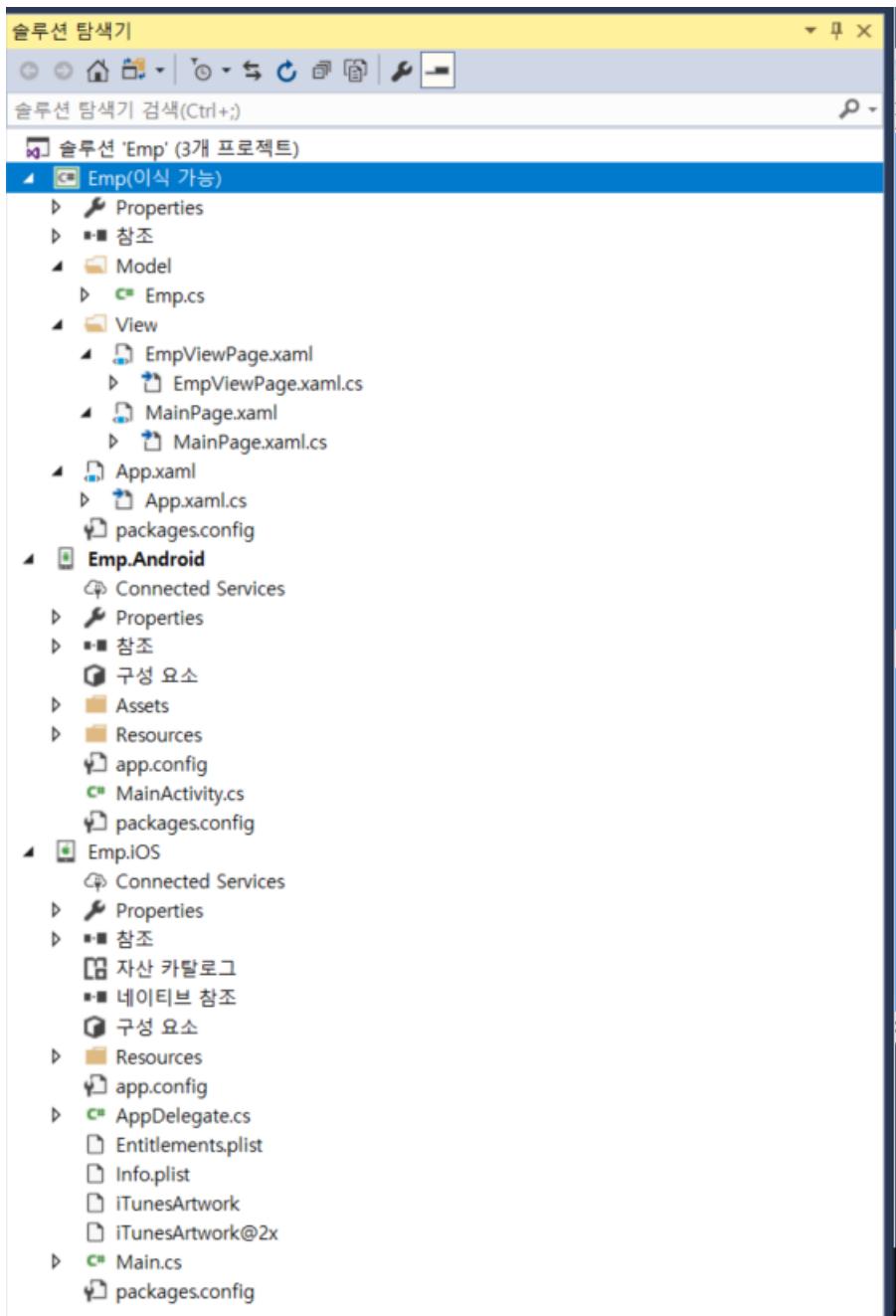
- 이제는 Xamarin.Forms 응용프로그램에서 모바일 UI를 만들어서 테스트해보자.
- 비주얼 스튜디오 2017 커뮤니티를 실행 후 Xamarin.Forms 크로스 플랫폼 앱, PCL 프로젝트를 생성하자.



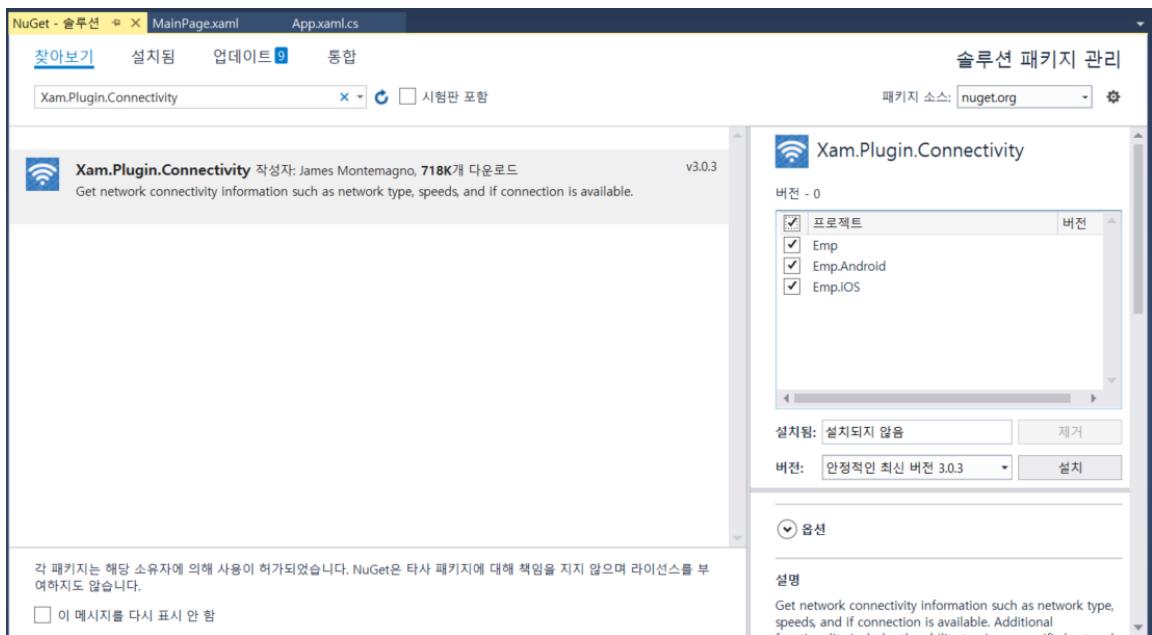
■ 다음화면에서 Blank, .Net Standard 선택 후 OK 클릭.



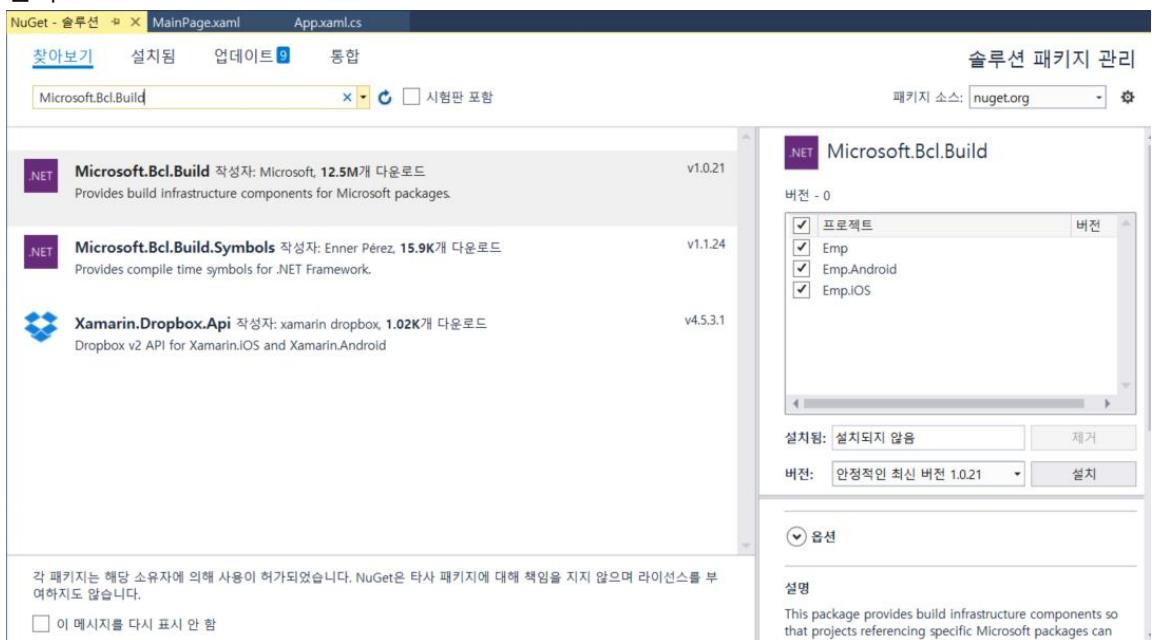
전체 프로젝트 구조



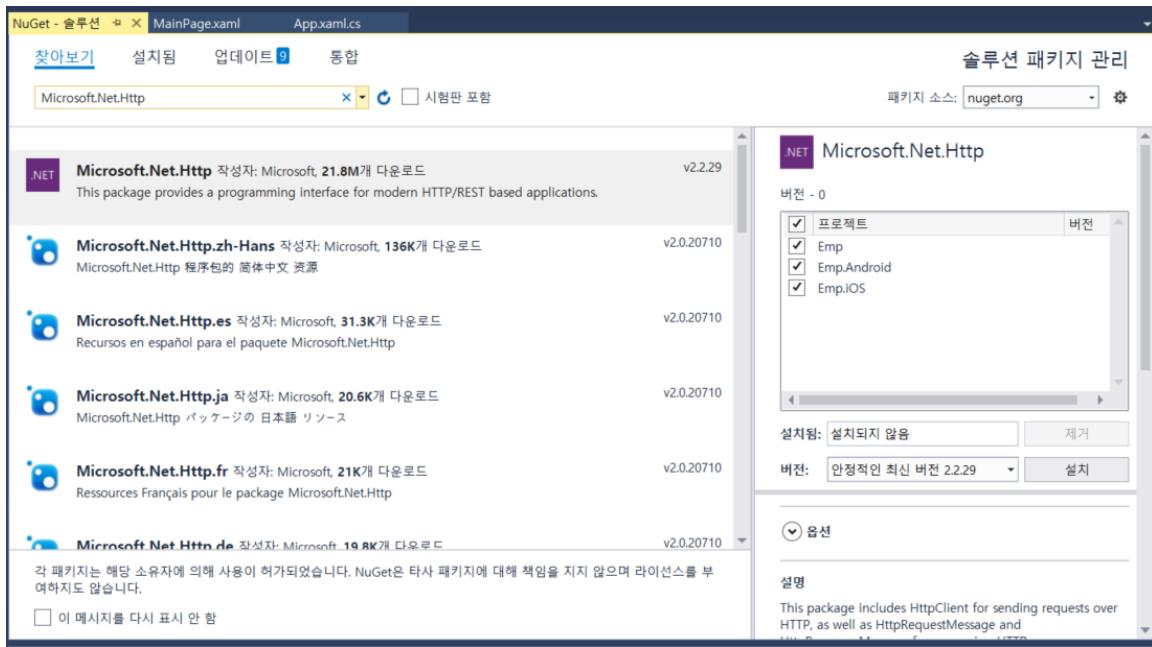
- Xamarin.Forms에서 인터넷 연결을 체크할 수 있는 "Xam.Plugin.Connectivity" Nuget 패키지를 설치하자. 솔루션에서 마우스 우측버튼 >> 솔루션용 NuGet 패키지 관리 >> 찾아보기 에서 "Xam.Plugin.Connectivity" 입력 후 설치.



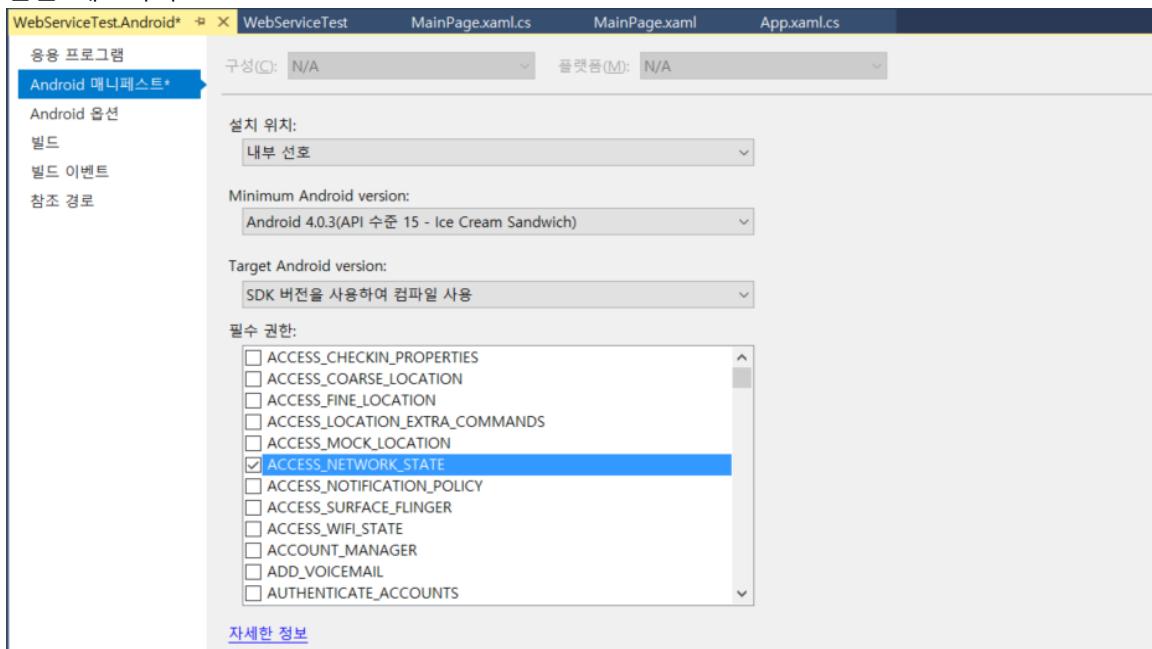
- Xamarin.Forms에서 HttpClient를 사용할 예정인데 직접 다룰 수는 없으므로 "Microsoft.Net.Http" Nuget 패키지를 추가해야 한다. "Microsoft.Net.Http"를 포함시키기 위해 솔루션 >> 우측마우스 >> 솔루션용 NuGet 패키지 관리에서 "Microsoft.Bcl.Build"를 추가 한다.



- 솔루션 >> 우측마우스 >> 솔루션용 NuGet 패키지 관리에서 "Microsoft.Net.Http"를 추가 한다.



- 안드로이드 플랫폼에서 테스트를 위해 안드로이드쪽 프로젝트에서 Properties를 더블클릭 하여 “Android 매니페스트옵션” >> “필수권한”에서 “ACCESS_NETWORK_STATE”, “INTERNET” 권한을 체크하자.



- 웹서비스 쪽에서 응답으로 넘어오는 JSON 형식의 문자열에 대응하는 C# 클래스(Emp.cs)를 Model 폴더에 만들자. <http://json2csharp.com/> 사이트에 JSON 문자열을 C# 클래스로 매핑하는 솔루션이 있어 이를 사용하기로 한다.

- [Emp.cs] <<= 이식가능 프로젝트에 Model을 만들고 그 안에 위치

```
using System.Collections.Generic;
```

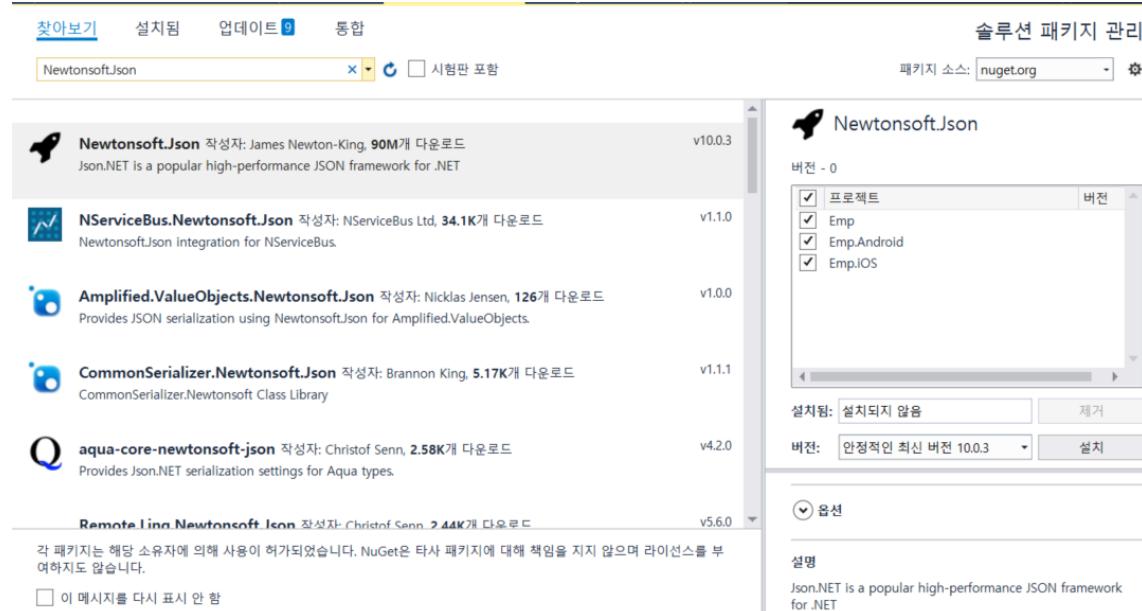
```

namespace Emp.Model
{
    public class Emp
    {
        public int Empno { get; set; }
        public string Ename { get; set; }
        public int Sal { get; set; }
        public string Job { get; set; }
    }

    public class EmpList
    {
        public List<Emp> emps { get; set; }
    }
}

```

- 자마린 프로젝트에 “Newtonsoft.Json” Nuget 패키지를 추가해야 한다. 이전에 추가한 방식대로 설치하자.



- [App.xaml.cs 수정]

```

using Xamarin.Forms;

namespace Emp
{
    public partial class App : Application
    {
}

```

```

public App ()
{
    MainPage = new NavigationPage (new MainPage());
}
.....
.....

```

- 이식 가능 프로젝트에 View 폴더를 만들고 전체 사원 목록을 출력하기 위한 MainPage.xaml 파일을 만들자. (기존에 있는 MainPage.xaml 파일을 위치 이동 시키자.)

■ [MainPage.xaml]

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamar.in.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:Emp"
    x:Class="Emp.MainPage">
    <Grid>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <Button x:Name="Button1" Grid.Row="0" Margin="10"
Text="사원 리스트 보기" Clicked="Button1_Click" />
            <ListView x:Name="listView" Grid.Row="1"
HorizontalOptions="FillAndExpand"
HasUnevenRows="True"
ItemSelected="listView_ItemSelected">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <Grid HorizontalOptions="FillAndExpand"
Padding="10">
                                <Grid.RowDefinitions>
                                    <RowDefinition Height="Auto"/>
                                    <RowDefinition Height="Auto"/>
                                </Grid.RowDefinitions>
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition Width="1*"/>
                                    <ColumnDefinition Width="1*"/>
                                </Grid.ColumnDefinitions>
                                <Label Text="{Binding Empno}"
Grid.Row="0" Grid.Column="0" TextColor="Red"
HorizontalTextAlignment="Center" />

```

```

        <Label Text="{Binding Ename}"
Grid.Row="0" Grid.Column="1" TextColor="Blue"
HorizontalTextAlignment="Center"/>

        <BoxView HeightRequest="1"
Margin="0,5,5,0" BackgroundColor="Gray" Grid.Row="2"
Grid.ColumnSpan="2" HorizontalOptions="FillAndExpand" />
    </Grid>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</Grid>
<ActivityIndicator x:Name="activityIndicator" IsRunning="True"
IsVisible="false" Color="Green"/>
</Grid>
</ContentPage>
```

■ [MainPage.xaml.cs]

```

using Newtonsoft.Json;
using Plugin.Connectivity;
using System.Collections.Generic;
using System.Net.Http;
using Xamarin.Forms;

namespace Emp
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        //사원 전체 목록 출력
        private async void Button1_Click(object sender,
System.EventArgs e)
        {
            /* 응답 JSON 형식
            * [{"empno":1001,"ename":"TopCredu","sal":6000,"job":"CLERK"}, 
            * {"empno":1002,"ename":"Jhon
Denver","sal":7000,"job":"SALESMAN"}, 
            * {"empno":1003,"ename":"Kar I
```

```

        "Luice", "sal":8000, "job":"MANAGER"},  

        * {"empno":1004, "ename":"Chanho  

    Park", "sal":7000, "job":"SALESMAN"},  

        * {"empno":1005, "ename":"Phi le  

    Migalon", "sal":8000, "job":"MANAGER"},  

        * {"empno":1006, "ename":"Jhon  

    Smith", "sal":7500, "job":"SALESMAN"},  

        *  

{"empno":1007, "ename":"OracleJavaCommunity", "sal":6800, "job":"CLERK"},  

        * {"empno":1008, "ename":"0jc.Asia", "sal":7200, "job":"CLERK"}]  

* */

activityIndicator.isVisible = true;
string url = "http://192.168.0.189:8080/emp/all";

// 네트워크 연결 상태 확인
if (CrossConnectivity.Current.isConnected)
{

    var client = new HttpClient();
    var res = await client.GetAsync(url);
    string ResponseStr = await
res.Content.ReadAsStringAsync();

    //EmpList empList = new EmpList();

    if (ResponseStr != "")  

{
        // 웹서비스에서 리턴하는 JSON 형식을 EmpList  

        형식으로 변환
        listView.ItemsSource =
JsonConvert.DeserializeObject<List<Model.Emp>>(ResponseStr);
    }
    // 리스트뷰의 아이템소스로 emps 컬렉션을 할당 화면에  

    출력됨
    // listView.ItemsSource = empList.Emps;
}
else
{
    await DisplayAlert("에러!", "네트워크 연결상태를  

    확인하세요.", "Ok");
}
//응답이 다 넘어온 경우 비활성화 시킴

```

```

        activityIndicator.isVisible = false;
    }

    //선택한 사원을 다음 화면에 출력
    private async void listView_ItemSelected(object sender,
SelectedIndexChangedEventArgs e)
{
    /* 응답 JSON 형식
       {"empno":1001,"ename":"TopCredu","sal":6000,"job":"CLERK"}
    */
    activityIndicator.isVisible = true;

    var Emp = e.SelectedItem as Model.Emp;
    var nextPage = new View.EmpViewPage();

    //웹서비스 호출 URL 생성
    string url = "http://192.168.0.189:8080/emp/" + Emp.Empno;

    // 네트워크 연결 상태 확인
    if (CrossConnectivity.Current.isConnected)
    {

        var client = new HttpClient();
        var res = await client.GetAsync(url);
        string ResponseStr = await
res.Content.ReadAsStringAsync();

        if (ResponseStr != "")
        {
            // 웹서비스에서 리턴하는 JSON 형식을 EmpList
            형식으로 변환
            nextPage.BindingContext =
JsonConvert.DeserializeObject<Model.Emp>(ResponseStr);
        }
    }
    else
    {
        await DisplayAlert("에러!", "네트워크 연결상태를
확인하세요.", "Ok");
    }
    //응답이 다 넘어온 경우 비활성화 시킴
    activityIndicator.isVisible = false;
}

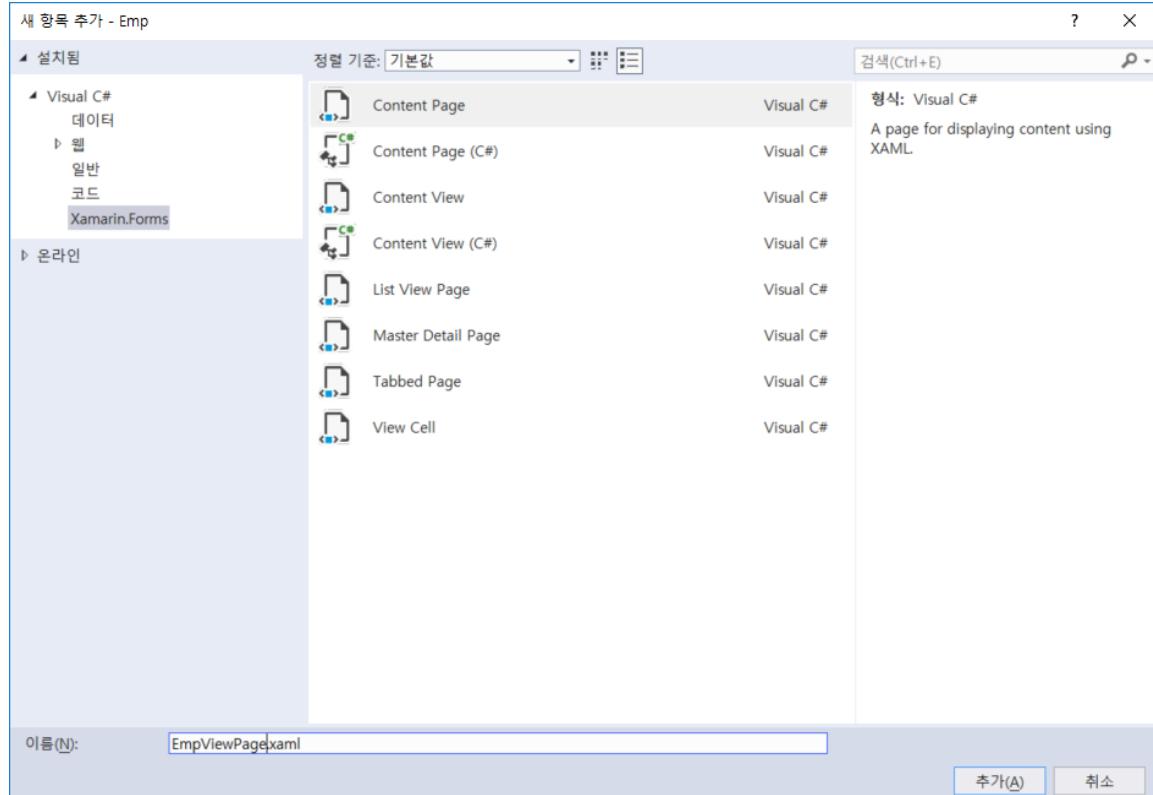
```

```

        await Navigation.PushAsync(nextPage);
    }
}
}

```

- 사원 데이터를 보기 위한 상세 페이지 UI를 작성 하자. (새 창에서 실행됨)
- View 폴더 아래에서 추가 >> 새 항목추가 >> Xamarin.Forms >> Content Page 템플릿으로 EmpViewPage.xaml 을 추가하자.



- EmpViewPage는 전체 사원 목록 보기(ListView)에서 한 사원을 선택했을 때 새로 뜨는 창이며 보통 ListView의 있는 내용을 가지고 뜨지만 본 예제에서는 자바, 스프링쪽의 웹서비스를 호출하면서 사번을 넘겨주고 응답으로 해당 사원객체를 JSON 형식으로 받아서 데이터 바인딩을 이용하여 화면에 출력했다.
- [EmpViewPage.xaml]

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="Emp.View.EmpViewPage">
    <StackLayout>

        <Label Text="사원 상세보기" HorizontalOptions="Center"
               TextColor="Blue" FontSize="30"/>

```

```

<Grid x:Name="GridDetails" VerticalOptions="Center"
HorizontalOptions="Center" Margin="0, 10, 10, 0" >
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*"/>
        <ColumnDefinition Width="1*"/>
    </Grid.ColumnDefinitions>

    <Label Text="사번:" Grid.Row="0" Grid.Column="0"
HorizontalOptions="Center" FontAttributes="Bold"/>
        <Label Text="{Binding Empno}" Grid.Row="0" Grid.Column="1"
HorizontalOptions="Center" />
        <BoxView HeightRequest="2" Margin="10,10,10,0"
BackgroundColor="Gray" Grid.Row="1" Grid.ColumnSpan="2" />

        <Label Text="이름:" Grid.Row="2" Grid.Column="0"
HorizontalOptions="Center" FontAttributes="Bold"/>
        <Label Text="{Binding Ename}" Grid.Row="2" Grid.Column="1"
HorizontalOptions="Center" />
        <BoxView HeightRequest="2" Margin="10,10,10,0"
BackgroundColor="Gray" Grid.Row="3" Grid.ColumnSpan="2" />

        <Label Text="급여($):" Grid.Row="4" Grid.Column="0"
HorizontalOptions="Center" FontAttributes="Bold"/>
        <Label Text="{Binding Sal}" Grid.Row="4" Grid.Column="1"
HorizontalOptions="Center" />
        <BoxView HeightRequest="2" Margin="10,10,10,0"
BackgroundColor="Gray" Grid.Row="5" Grid.ColumnSpan="2" />

        <Label Text="직무:" Grid.Row="6" Grid.Column="0"
HorizontalOptions="Center" FontAttributes="Bold"/>
        <Label Text="{Binding Job}" Grid.Row="6" Grid.Column="1"
HorizontalOptions="Center" />
        <BoxView HeightRequest="2" Margin="10,10,10,0"
BackgroundColor="Gray" Grid.Row="7" Grid.ColumnSpan="2" />

```

```
</Grid>
</StackLayout>
</ContentPage>
```

■ [EmpViewPage.xaml.cs] 파일은 수정할 필요없다.

- 에뮬레이터는 그냥 실행하면 되지만 안드로이드 휴대폰에서 실습을 위해서는 모바일에서 웹 서비스에 접근 가능 해야 하므로 PC의 네트워크와 같은 네트워크의 Wifi에 휴대폰을 연결하고 테스트 하면 된다. 실행화면은 다음과 같다.



