Introduction to Xamarin Framework

한양대학교

한양대학교 학술타운 LION Hanyang FMS	
이름	학과
권나현	컴퓨터전공
구장회	컴퓨터전공
김동현	정보시스템
문재석	컴퓨터전공
신정아	정보시스템

1) Xamarin Framework?

a. Xamarin 탄생 배경

b. C#과 .NET 기반의 솔루션

c. Code Sharing

d. Xamarin.Forms

e. Xaml의 지원

f. 프레임워크 개발 환경

g. Xamarin forms, 플랫폼별 구현

2) Android / iOS 플랫폼 특성 비교

a. Android?

b. iOS?

c. Android와 iOS의 차이

3) C# - 언어적 특성 파악하기

a. C#의 탄생

b.객체지향 프로그래밍

c. 기본 구성요소

d. C#을 사용한 예제

4) 실제 어플리케이션 개발

a. 전체 UI

b. Main 구현 code

c. Market List 구현 코드

d. Market Detail 구현 코드

e. DeLion with Xamarin

5) 다양한 API 적용 - Cognitive Service

a. Xamarin의 API 호환성 검증

b. Cognitive API - Xamarin에의 적용

c. API 적용 결과

1. Xamarin Framework?

a. Xamarin 탄생배경

Xamarin이 탄생하고 주목받게 된 계기는 다양한 모바일 환경이 주요 원인이다. iPhone, Android 그리고 Window Phone까지, 세 가지 모바일 환경이 있다. 이 각기 다른 모바일 환경은 다음과 같은 문제점을 가지고 있다;

1. 다른 UI 패러다임
    A. 각 플랫폼은 페이지마다 이동하는 방법이 다르다.
    B. 데이터를 표시하는 다른 convention이 존재하다.
    C. Display menu 표시 방법이 다르다.
    D. 터치 방법도 다르다.
2. 다른 개발 환경
    A. iOS 개발엔, Xcode, Mac에서만 개발가능
    B. Android 개발엔, Android Studio, 다양한 플랫폼에서 개발 가능
    C. Windows 개발엔, Visual Studio, PC에서 개발 가능
3. 다른 프로그래밍 인터페이스
    A. 세가지 플랫폼은 다른 operating system에서 각기 다른 API에 기반한다.
    B. 비슷한 type의 user-interface object를 적용하지만, 다른 이름들을 갖고 있다.
        i. iPhone에선, 'view'가 'UIswitch' 라고 불린다.
        ii. Android에선, 'widget' 이 'Switch' 라고 불린다.
        iii. Windows Runtime API의 'control'은 'ToggleSwitch' 라고 불린다.

4. 다른 프로그래밍 언어

   A. iPhone은, Objective-C

   B. Android는 Java

   C. Windows는 C#

다음과 같이 통합되지 못한, 플랫폼은 개발적 측면에서 너무나도 많은 문제점을 갖고 있다. 같은 기능을 함에도 한 명 이상의 개발자가 필요하거나 혼자서 더 많은 기술이 필요하다. 그래서, 대안으로 등장한 것이 Cross-Platform을 지향하는 Xamarin 이라는 언어이다. 다양한 플랫폼 문제를 모두 해결 하지는 못하지만, 적어도 같은 언어로 코드를 공유하고, 한가지 개발 환경을 가질 수 있다.

b. Xamarin - C# 과 .NET 솔루션

2000년에 Microsoft 사에 의해 개발된 언어인 C#은 다른 프로그래밍 언어에 비해, 새로운 언어이다. C++에서 영향을 받아 탄생한 C#은 객체지향의 장점을 갖고 있으면서도 C++보다 직관적이다. .NET Framework 클래스 라이브러리는 프로그래밍의 기본적인 기능들을 제공한다: Math, Debugging, Reflection, Collections, Globalization, File I/O, Networking, Security, Threading, Web services, Data handling, XML and JSON reading and writing.

Microsoft가 .NET을 2000년 6월에 발표하고, Ximian 회사 (Miguel de Icaza 와 Nat Friedman에 의해 창립) Mono라는 오픈소스 프로젝트를 시작했다. 이는 C# 컴파일러와 .NET Framework를 Linux 에 적용시키기 위한 프로젝트였다. 10년 후 2011년에, Ximian의 창업자들은 Xamarin을 만들었는데, 이는 Mono에서 많은 영향을 받았고, 크로스 모바일 솔루션의 기초가 되었다.

그들은 꾸준히 Xamarin 프로젝트를 연구해왔고, .NET Framework Platform이라 불리는 C# compiler가 2014년 출시되면서, .NET 기술이 발전했고, 이는 Xamarin에서 더 큰 비중을 맡는 계기가 되었다고 한다.

2016년 3월, Microsoft가 Xamain 을 모바일 크로스 플랫폼 개발 환경 구축을 위한 기술

로써 인수 합병하며, 더 많은 Microsoft 개발자들을 얻게 되었다. 그리고, Xamarin Forms는 Visual Studio에서 무료로 사용 가능하다.

c. Sharing code

Xamarin의 가장 큰 장점은 C# 만으로 다른 플랫폼에서의 개발이 가능하다는 것이다. 각 플랫폼 별로 오직 C#으로도 개발이 가능하다는 것이다.

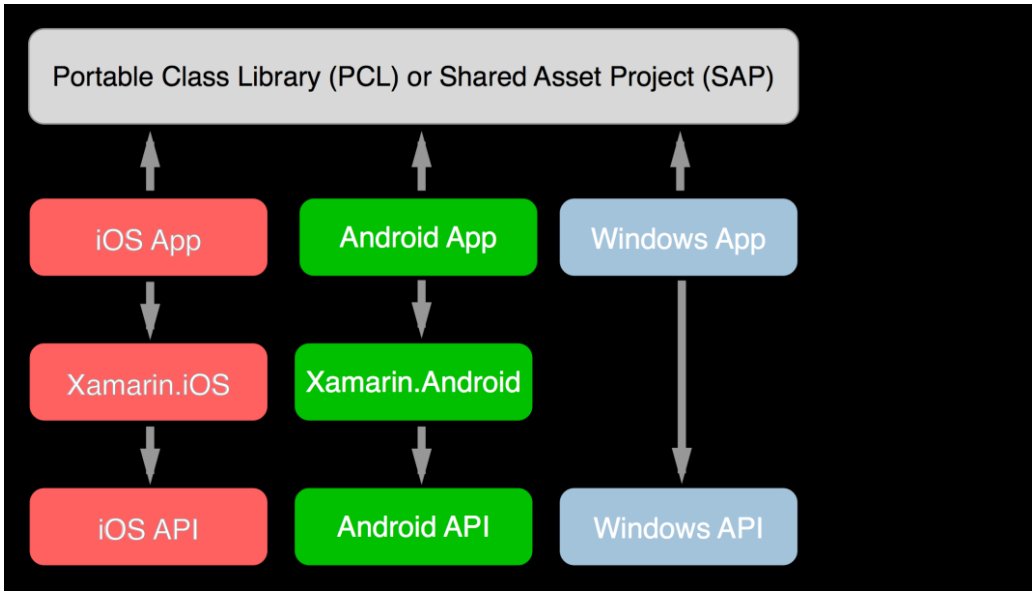
보통, 개발자들은 Model-View-Controller 즉, MVC 구조로 코드를 나누어서 작성한다. 하지만 최근에, MVVM (Model-View-ViewModel) 구조로 넘어오는 추세이다. MVVM 구조는 코드를 Model (깔려있는 데이터), View (유저 인터페이스, visual과 input), 그리고 ViewModel(Model 과 View 사이에 data passing을 관리) 으로 나눈다. 그래서, 개발자가 여러 모바일 플랫폼을 타겟팅하고, 플랫폼 별 다른 view, API, model 등이 어떻게 나누어지는지를 안내할 수 있다.

platform-independent 코드는 파일이나 resource, use collection, threading 처리를 필요로 한다. 보통 이런 일들은 operating system API의 일로써 간주되는데, .NET Framework class library가 할 수 있는 일이기도 하다.

그래서, 이 공통의 코드들은 .NET Framework class library에 접근할 수 있어서, 각기 다른 플랫폼에서도 파일 I/O, globalization 다루기, 웹 서비스, XML 등을 다룰 수 있는 것이다.

요약하자면, 각기 다른 플랫폼에서는, 특정 기능들을 수행할 때, 이를 처리하는 os API가 달라서 문제가 되었지만, 이를 .NET Framework가 수행하면서, 각기 다른 플랫폼들을 .NET Framework에 접근 시켜서 Code Sharing을 가능하게 한 것이다. 그래서, 기존의 C#으로 만들어지는 Window 어플리케이션 기능들을 똑같이 다른 플랫폼에서 구현을 하면서, 충돌이 일어나는 부분들을 .NET Framework에서 처리해주는 것이다.

그래서, Visual Studio에서 솔루션을 새로 만들면, 네 개의 C# 프로젝트가 만들어진다. Xamarin 라이브러리인 PCL 또는 SAP가 하나이고, iOS, Android, 그리고 Windows 파일이 만들어진다.



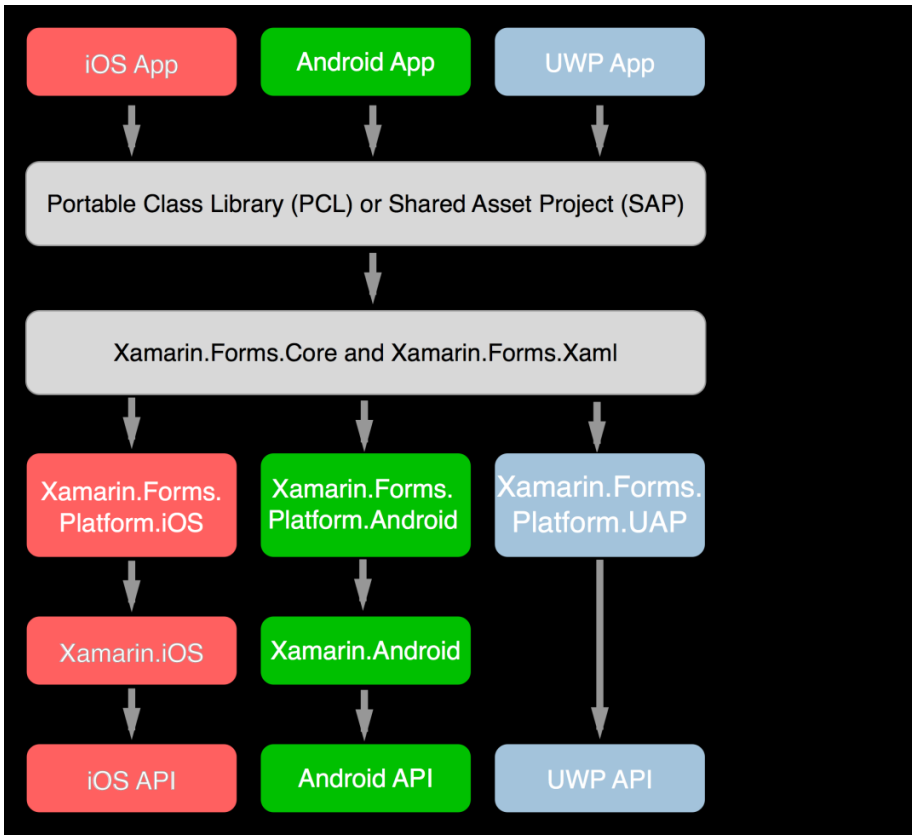
그림을 보면, 두번째 행의 각 플랫폼들이 Xamarin 라이브러리에 접근해서, Xamarin native platform API를 사용한다. 그리고, 사진에는 나와있지 않지만, 이들이 .NET Framework 라이브러리에 접근한다.

iOS 에서 코딩을 하면, Xamarin.iOS에서 언어 바인딩을 해주고, Xamarin C# 컴파일러가 C# Intermediate Language(IL) 을 생성한다. 그리고, 이것은 Objective-C compiler 처럼 Mac의 Apple 컴파일러를 사용해서 어플리케이션을 생성한다.

Android에서는, Xamarin C# 컴파일러가 IL을 만들고, Java engine에서 컴파일을 해서, 어플리케이션을 생성한다.

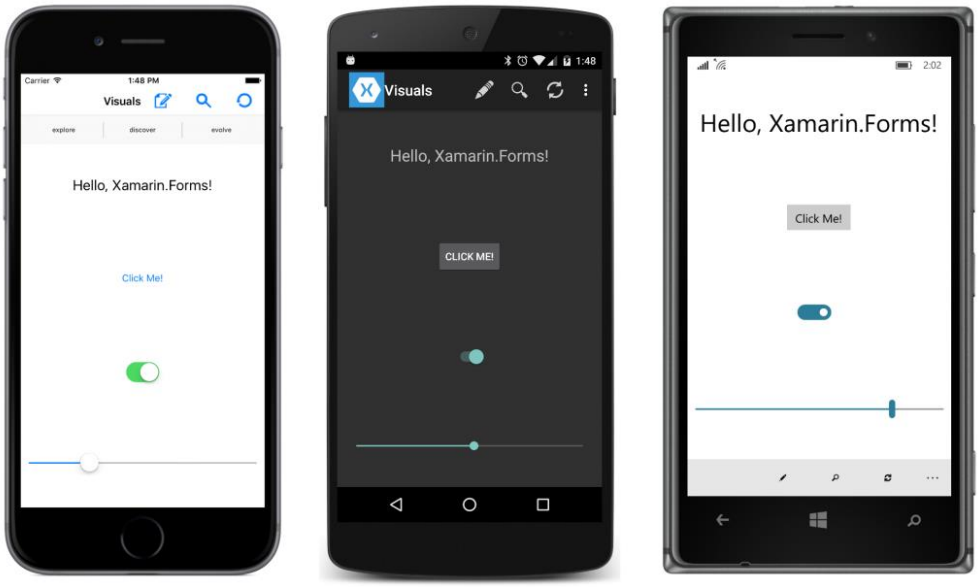
d. Xamarin.Forms 소개

기존의 Xamarin이 각 플랫폼 별로 C#으로 각자 코딩을 할 수 있었다는 것이 주요 장점이었다면, Xamarin.Forms는 오직 한 플랫폼에서 코딩을 해서, 세가지 플랫폼으로 애플리케이션을 생성할 수 있다는 마법 같은 기능을 제공한다.



위 그림은, Xamarin.Forms의 작동 원리를 설명한다. 위의 Xamarin과의 차별점은 중간에 Xamarin.Forms.Core 와 Xamarin.Forms.Xaml 이 있고, 그들이 각각 Xamarin.Forms.Platofm 과정으로 넘어간다는 것이다. Core와 Xaml은 Xamarin.Forms의 코딩 파일 및 라이브러리 이다. 그리고, Xamarin.Forms.Platform은 위에서 한 코드들을 각각의 Xamarin 라이브러리로 변형해주는 기능을 하는 것이다. 즉, Xamarin.Forms에서 Switch라는 코드를 사용했으면, Xamarin.Forms.Platorm에서는 Switch가 iPhone에선, UISwitch, Android에서는, Switch, Windows에서는, ToggleSwitch로 매핑해준다는 것이다.

Xamarin.Forms에서 코딩을 하고 컴파일을 시켜, 결과를 출력하면, 다음 그림과 같이 세가지 플랫폼에서 같은 어플리케이션을 만들 수 있다.



e. XAML 지원

Xamarin.Forms는 XAML이라는 XML 기반 Extensible Application Markup 언어를 지원한다. 이는 오브젝트와 View들을 초기화하는데 사용되는 언어이다. C# 코드와 접목이 되며, Java의 XML과 같은 기능으로서 사용된다. XAML은 다음과 같이 생겼다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="PlatformVisuals.PlatformVisualsPage"
             Title="Visuals">

    <StackLayout Padding="10,0">
        <Label Text="Hello, Xamarin.Forms!"
               FontSize="Large"
               VerticalOptions="CenterAndExpand"
               HorizontalOptions="Center" />

        <Button Text = "Click Me!"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="Center" />

        <Switch VerticalOptions="CenterAndExpand"
                HorizontalOptions="Center" />

        <Slider VerticalOptions="CenterAndExpand" />
    </StackLayout>

    <ContentPage.ToolbarItems>
        <ToolbarItem Text="edit" Order="Primary">
            <ToolbarItem.Icon>
                <OnPlatform x:TypeArguments="FileImageSource"
                            iOS="edit.png"
                            Android="ic_action_edit.png"
                            WinPhone="Images/edit.png" />
            </ToolbarItem.Icon>
        </ToolbarItem>

        <ToolbarItem Text="search" Order="Primary">
            <ToolbarItem.Icon>
                <OnPlatform x:TypeArguments="FileImageSource"
                            iOS="search.png"
                            Android="ic_action_search.png"
                            WinPhone="Images/feature.search.png" />
            </ToolbarItem.Icon>
        </ToolbarItem>

        <ToolbarItem Text="refresh" Order="Primary">
            <ToolbarItem.Icon>
                <OnPlatform x:TypeArguments="FileImageSource"
                            iOS="reload.png"
                            Android="ic_action_refresh.png"
                            WinPhone="Images/refresh.png" />
            </ToolbarItem.Icon>
        </ToolbarItem>

        <ToolbarItem Text="explore" Order="Secondary" />
        <ToolbarItem Text="discover" Order="Secondary" />
        <ToolbarItem Text="evolve" Order="Secondary" />
    </ContentPage.ToolbarItems>
</ContentPage>
```

f. 개발환경

Xamarin 개발을 위해서는 Mac의 Xamarin Studio나 PC의 Visual Studio 프로그램을 통해, 개발이 가능하다. 모두 Xamarin Library를 다운받는 것이 필수이다. 단, Visual Studio에서 iOS 결과를 확인하려면, Mac과 연결을 해야 가능하다. 즉, PC 와 Mac 둘 다 필요하다는 것이다.

실제 폰을 USB 케이블과 연결해서, 어플리케이션을 다운받아서 결과를 확인할 수 있다. Xamarin Studio나 Visual Studio에 내장된 에뮬레이터를 사용해도 되고, 다른 에뮬레이터 프로그램을 받아서 사용해도 된다.

g. Welcome to Xamarin Forms! 띄우기

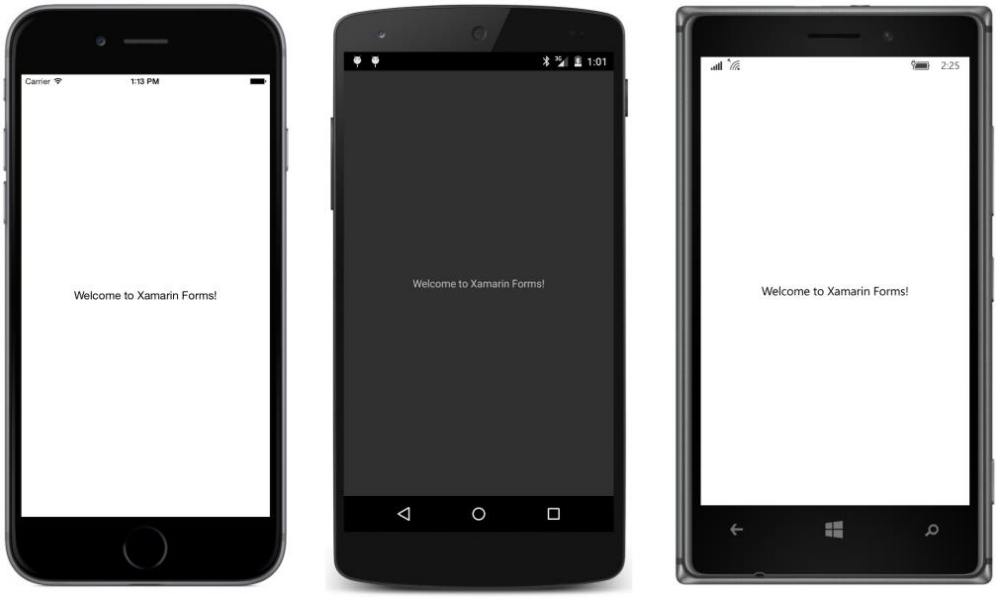
Visual Studio와 Xamarin Studio는 거의 비슷하니, Visual Studio에서의 예제 설명을 하겠다.

먼저, 메뉴 옵션에 File > New > Proejct로 들어가서 New Project를 누른다. 그리고, 세가지 옵션 중 Blank App(Xamarin.Forms Portable)를 선택한다. PCL을 사용 하나, SAP를 사용 하나의 차이지만, 큰 상관은 없다고 한다. Hello를 파일이름으로 지정하고 프로젝트를 만들면, 6개의 프로젝트가 생겨난다; Hello, Hello.Driod, Hello.iOS, Hello.UWP, Hello.Windows, Hello.WinPhone. 우린 Android, iOS, Window 어플리케이션만 생성할 것이니, UWP, Windows는 삭제해도 상관없다.

새 프로젝트를 생성할 때, NuGet 패키지도 자동으로 다운받아진다. 하지만, 특정 패키지는 솔루션 템플릿에 따라서, 이전 버전이 다운받아질 수도 있다. 이렇게 되면, 에러가 발생할 수 있기 때문에, Nuget Package Manager 에서 모두 업데이트하기 바란다. 이 문제 때문에, 우리 팀원들은 오랜 시간 동안 Xamarin 개발을 시작할 수 없었다.

계속하기 전에, 설정이 제대로 되있는지를 확인하는 것이 좋다. Build > Configuration Manager 메뉴로 들어간다. Build 박스가 모든 플랫폼 별로 체크가 되어있어야 하고, Deploy 박스도 역시 모두 체크되어있어야 한다. 그리고, Hello 프로젝트가 있으면, Any CPU로서 flag 되어있어야 한다. Hello.Droid, Hello.iOS 프로젝트도 역시 Any CPU로 되어있어야 한다.

모든 준비를 마쳤으니, Debug를 하면 다음과 같이 어플리케이션이 생성된다.



보기와 같이, 플랫폼 별로 UI가 상이하다는 것을 알 수 있다. 지금은 간단한 차이이지만, 많은 UI가 추가될수록 차이는 더 확연해 진다. 이 차이를 다루는 곳은 각각의 Droid, iOS, WinPhone 파일들이다. 즉, Hello 프로젝트에서 Xamarin.Forms의 역할을 수행해서, 기본적인 어플리케이션 빌드를 하는 곳이다. 그리고 나머지 파일에서는 세부적인 차이점을 고쳐주는 역할을 한다.

다음 코드는 화면에 텍스트를 그린 것에 대한 코드이다. 이것은 Hello 프로젝트에 App.cs라는 클래스로 선언되어있다. Xamarin Studio에는, Hello.cs라고 되어있다. 이것이 Xamarin.Forms의 C#으로 만든 코드라고 볼 수 있다.

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

using Xamarin.Forms;

namespace Hello
{
    public class App : Application
    {
        public App()
        {
            // The root page of your application
            MainPage = new ContentPage
            {
                Content = new StackLayout
                {
                    VerticalOptions = LayoutOptions.Center,
                    Children = {
                        new Label {
                            HorizontalTextAlignment = TextAlignment.Center,
                            Text = "Welcome to Xamarin Forms!"
                        }
                    }
                }
            };
        }

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

        protected override void OnSleep()
        {
            // Handle when your app sleeps
        }

        protected override void OnResume()



        {
            // Handle when your app resumes
        }
    }
}
```

namespace는 프로젝트 이름과 같다는 것을 알 수 있다. 그리고, App 클래스는 public으로 선언되어 있고, Xamarin.Forms Application 클래스로부터 왔다고 정의 되어있다. constructor에는 MainPage라는 Application의 property 밖에 가지고 있지않다.

Xamarin.Forms 템플릿은 constructor를 정의하는데 아주 쉬운 방법을 사용한다; ContentPage 클래스는 Xamarin.Forms 어플리케이션의 주요 화면을 나타낸다. 이것은 화면 상단의 status bar 와 하단의 폰 버튼들을 제외한 모든 화면 부분을 다룬다.

그리고 StackLayout이라는 layout 안에, Label을 선언해서 Welcome to Xamarin Forms라는 텍스트를 화면에 띄웠다는 것을 알 수 있다.

iOS 프로젝트

```
using Foundation;
using UIKit;

namespace Hello.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);
        }
    }
}
```

위 코드는 iOS 프로젝트이다. 앞에서 말한 것과 같이, 플랫폼별로 차이가 생겼을 때, 다룰 수 있는 부분이다. 코드는 보기와 같이, 간단하게, LoadApplication 이라는 함수로 App 클래스를 불러오는 행동만 하고 있다.

Android 프로젝트

```
using Android.App;
using Android.Content.PM;
using Android.OS;

namespace Hello.Droid
{
    [Activity(Label = "Hello", 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.Forms.Init(this, bundle);
            LoadApplication(new App());
        }
    }
}
```

Android 프로젝트이다. iOS 프로젝트와 같은 역할을 하고, 비슷한 형태를 갖고 있다.

2. Android / Ios 플랫폼 분석

a. Andriod?

사용 언어 - JAVA

안드로이드는 Java언어를 사용하여 개발을 진행한다. Java의 가장 큰 특징은 Virtual Machine(가상머신)을 통한 범용성의 증가에 있다. 개발자가 입력한 .java 파일은 Window 혹은 mac 등 다양한 환경에 맞는 가상머신을 사용하여 ByteCode로 변형되어진다. 개발자의 입장에서는 하나의 .java 코드만을 생성하면 다양한 환경에서 같은 기능을 수행시킬 수 있다는 큰 장점이 존재한다. 안드로이드는 기존 가상머신을 사용하지 않고 Dalvik머신이라는 별도의 가상머신을 이용하여 그 효율성을 올리고 있다.

Android 동작구조

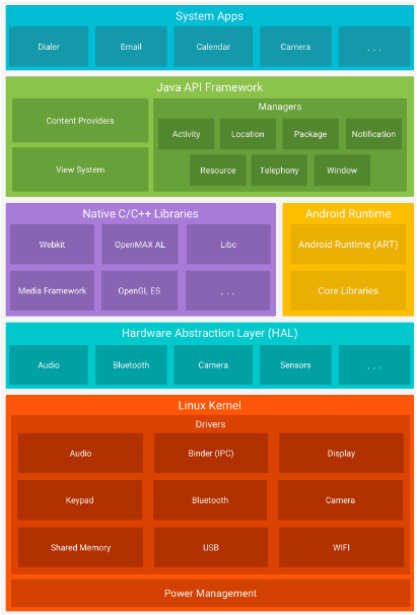


Figure 1) Android 구조 (출처: android developer guide)

안드로이드는 4가지 계층에 의해 동작하는 플랫폼이라고 정의할 수 있다.

- 가장 기본이 되는 Kernel(커널)에서는 메모리 프로세스 관리 모델, 디스플레이 드라이버, IPC Driver 등 스마트폰이 동작하는 데에 있어서 가장 기초가 되는 부분을 처리한다. PC에서 Window와 같은 운영체제 위에서 프로그램을 동작시키듯이 안드로이드 또한 리눅스 위에서 동작한다.
- 하드웨어 추상레이어는 스마트폰에 부착되어있는 카메라, GPS, Audio 등의 하드웨어들을 커널에 있는 드라이버와 매핑 해주는 역할을 수행하는 계층이다.
- 안드로이드 플랫폼에서 사용되는 기본 라이브러리의 경우 C 혹은 C++코드로 작성이 되어있다. 때문에 Java 코드로 작성된 Application계층이 라이브러리를 호출할 수 있도록 Dalvik머신에 있는 JNI(자바 네이티브 인터페이스)를 사용한다.
- 어플리케이션 프레임워크는 어플리케이션을 제작하는데 있어서 대부분의 개발자들이 접하게 되는 계층이다. APP의 구동 및 관리를 담당한다.

b. Ios?

사용 언어 - Object C, Swift

Object-C는 애플사의 '과거' 표준 프로그래밍 언어였다면 Swift는 '현재'프로그래밍 언어라고 표현할 수 있을 것이다. 두 언어의 공통적 철학은 C언어를 객체지향적으로 변형시키자 에 있다. 때문에 Object C의 컴파일러는 모든 C언어로 제작된 코드를 컴파일 가능하다. 이는 가장 고수준 프로그래밍 언어들 중에서 가장 속도가 빠른 C의 장점을 살리는 것이 가능하다는 것을 의미한다. 마찬가지로 Swift 또한 C의 기본 골격을 가지고 있으며 Object-C의 높은 진입장벽을 낮춘 언어로서 평가되고 있다.

Ios 동작구조

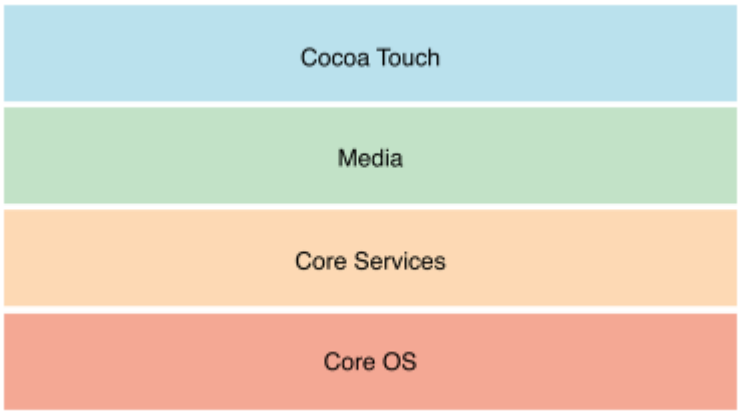


Figure 2) Ios 구조(출처: apple developer guide)

애플은 UNIX(유닉스) 기반의 독자적인 OS를 가지고 있는 회사이다. Ios 또한 애플이 가지고 있는 고유한 맥 OS를 기반으로 하는 다윈 커널을 기초로 동작하고 있다.

- Core OS에서는 안드로이와 마찬가지로 메모리 관리, 네트워크, 데이터 처리 등 운영체제가 관리하는 저수준의 API들로 이루어져있다.
- Core Services의 경우는 스마트폰에 보다 특화되어있는 예를 들어 기기의 위치를 감지하는 모션센서나 중력가속도센서 등 기본 PC에서는 필요로 하지 않았던 기능을 처리한다.
- Media는 2개의 아래 계층과는 달리 C와 Objective-C가 혼합되어 사용 되어진다. 아래 계층에서 하드웨어와 집적적인 연관성을 가진 저수준 API를 주로 처리했다면 비디오나 오디오 파일, 그래픽 처리 등 보다 소프트웨어적인 부분을 담당한다.
- Cocoa Touch에서는 사용자가 보는 화면의 그래픽 UI를 담당한다. Android에서의 Application계층과 그 성격이 유사하다고 할 수 있다.

c. Android와 Ios의 차이

프로그램은 어떠한 필요에 의해 만들어지며 그에 대한 목표를 뚜렷하게 보여준다. Android와 IOS 두 플랫폼에 대해 알아보면서 팀원들이 느낀 두 플랫폼의 목표는 각각 범용성, 효율성이었다.

Android의 경우 하나의 회사에서 만들어지는 스마트폰에 탑재되는 플랫폼이 아니다. 그로 인해 누구나 접근이 가능한 오픈 소스 기반의 리눅스를 OS로 채택하였고 또한 다수의 사용자들이 쉽게 접근하도록 어플리케이션의 구성 또한 패키지 형태를 선택하여 배포와 설치를 보다 용이하도록 설정되어있었다. 그에 반해 Ios는 아이폰이라는 단일 제품에서만 동작하도록 설계 되어졌고 때문에 모두가 사용할 수 있는 프로그램이 아닌 최고의 성능에 더 많은 노력을 기울였음을 발견할 수 있었다.

3. C# - 언어적 특성 파악하기

a. C#의 탄생

C#은 마이크로소프트사에서 .NET 프로그램을 개발할 수 있는 개발환경을 구축하기 위해 제작되었다. JAVA와 같은 세대에 개발된 개발 언어이며, 기본적으로 객체지향 언어이자 멀티 플랫폼 상에서 구동이 가능하도록 설계되었다. 이러한 프로그램 언어가 필요하게 된 배경에는 네트워크 프로그램의 수요가 폭발적으로 증가하였기 때문이다. 플랫폼 독립적인 프로그램을 만들 수 있다는 장점이 생산성 면에서 개발자들을 성공적으로 설득력을 얻어왔고, 개발자들은 어느 환경에서나 구동이 가능한 언어에 관심을 가지기 시작했다. 유니티 3D의 C# 프로그램은 .NET이 아닌 Mono-runtime이라는 가상머신 위에서 동작한다. JAVA와 같이 가상머신 위에 프로그램을 구동 시키는 방식을 사용하기에 여러 디바이스에서도 동일한 기능을 수행할 수 있게 되었다.

b. 객체지향 프로그래밍

1)정의

객체지향 프로그래밍(Object Oriented Programming, OOP)은 컴퓨터 프로그래밍을 다루는 관점 중 하나이다. 예전에는 프로그램을 명령어들의 집합이라고 생각했던 반면, 객체지향 프로그래밍에서는 프로그램을 객체들의 모임으로 정의하고 있는데 , 이를 통해 객체가 서로 메시지를 주고받으며, 데이터를 처리할 수 있게 한다.

2)객체지향 프로그래밍의 이유

프로그램이 한번 만들어지고 변경되지 않고 , 발전하지 않고 , 다른 곳에 응용되지 않는다면 객체지향 프로그래밍을 할 필요가 없을 것이다. 하지만 프로그램에서는 “변하지 않는 것은 ‘변화’ 뿐”이라고 할 정도로 프로그램은 어떤 이유에서 꼭 변경되고 확장되고 수정되며 재사용된다. 이러한 이유 때문에 확장성과 재사용성 , 유지보수성이 필요하게 되었다. 객체지향 프로그

래밍은 이러한 것들을 유용하게 해준다. 아래에서 객체지향 언어의 구성요소와 특징을 설명하겠다.

c. 기본 구성요소

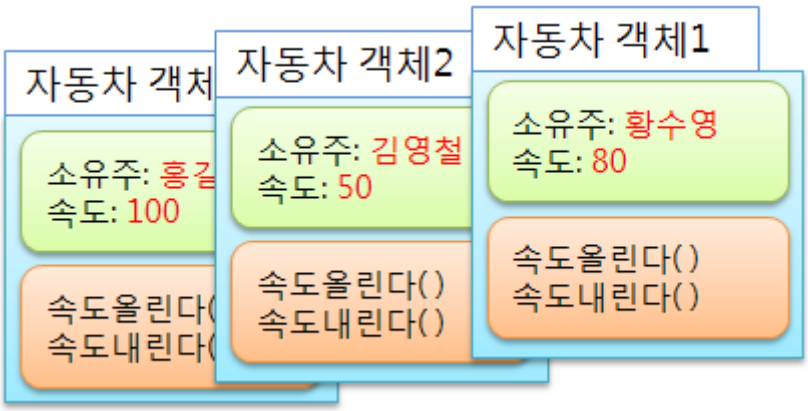
객체지향 언어는 기본적으로 클래스(Class), 객체(Object), 메서드(Method)로 구성된다.

(1)객체

'객체'라는 단어를 우리가 잘 사용하지 않을 뿐 우리는 객체들 속에서 객체들과 생활하고 있다. 즉 객체는 우리가 아는 거의 모든 것이다. 예들 들면 컴퓨터, 책상, 책, 의자, 나무, 숲, 시계, 전등, 사람 등등이다. 객체는 우리가 보고 만지고 느끼는 것 외에도 객체가 될 수 있다(비행경로, 비행시간, 날씨 정보, 인간관계, 생각, 관념 등등...)

('객체의 구조' 에 대한 그림)

예를 들어 자동차라는 '객체'에 대해서 보면 자동차는 상태로 소유주와 속도를 가질 수 있으며 행동으로 속도를 올리고 내릴 수 있다.



위 그림에서 객체의 특성으로 '독립적으로 존재하며 다른 객체와 구분된다'는 것을 알 수 있다. 다시 말해서 자동차 객체1과 자동차 객체2가 독립적으로 존재하고 , 자동차 객체1의 속도를 높이면 자동차 객체1의 속도만 올라가고 자동차 객체2의 속도나 다른 객체의 속도는 변하지 않습니다.

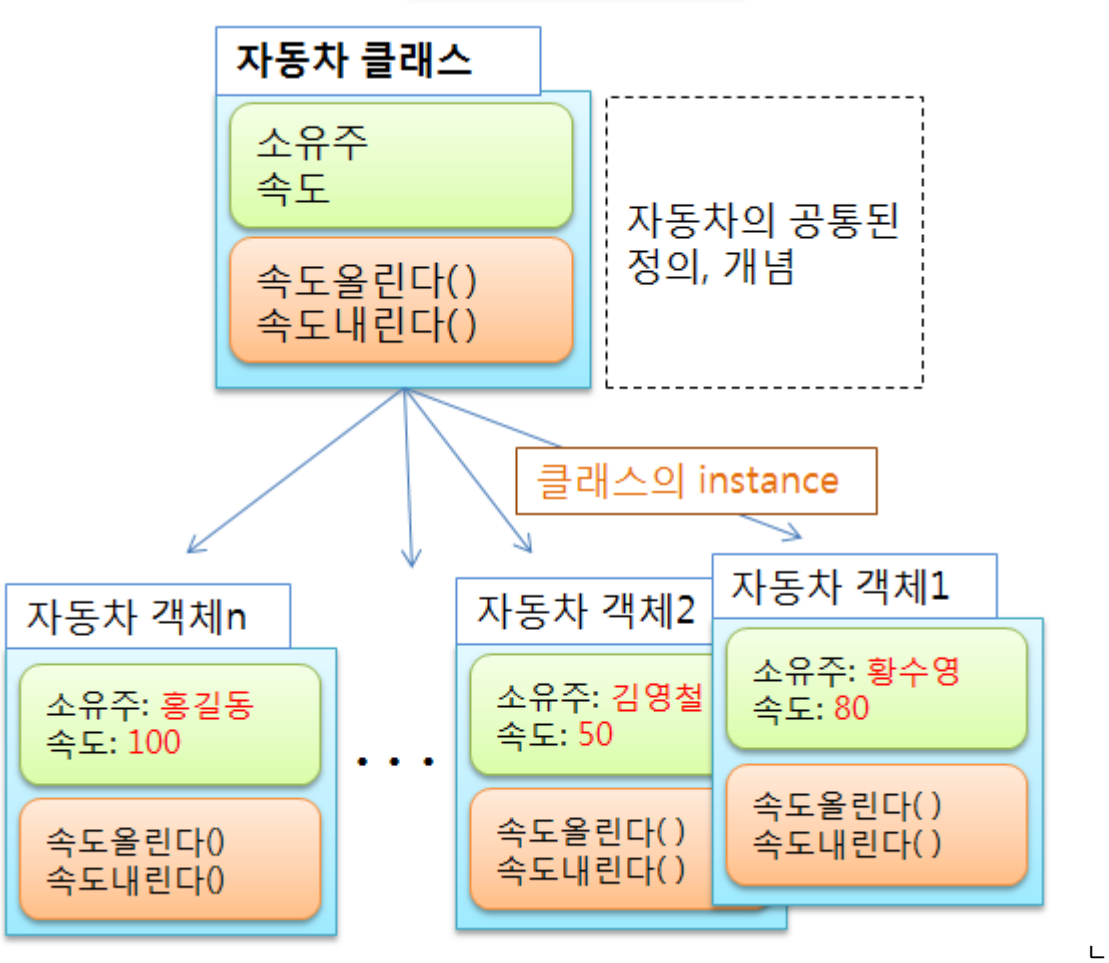
또 '객체'는 각각의 상태와 행동을 가진다. (위 그림에서 보면 속도라는 상태와 속도를 올리고 내리는 행동)

여기서는 객체의 예를 들기 위해 자동차에 대해 설명했지만 , 사실 자동차를 객체라고 하기는 어렵다. 뒤에서 자세히 설명 할 것이다. (위 그림의 자동차 객체 하나하나는 객체지만 자동차 객체는 자동차를 하나하나 구분하지 않으므로 객체라고 할 수는 없고 , 자동차는 자동차 객체의 공통적인 개념과 정의 , 틀을 표현하므로 '클래스'에 해당한다.)

(2)클래스

클래스란 상태와 그 상태를 다루는 행동들(메소드,연산)의 집합이라고 할 수 있다. 또 클래스는 객체들의 공통된 개념을 표현하고 정의하는데 따라서 클래스는 객체의 틀(template)이라고도 한다.

(클래스와 인스턴스의 차이: 클래스는 개념이고 정의이며 객체는 클래스의 인스턴스(실체)이다.)



(3)메서드

매서드란 '객체에서 속성을 변경하거나 연산을 할 때 사용하는 서브루틴'이라고 할 수 있다. 위의 그림에서의 예를 들면 , 자동차 클래스 안의 메서드로 '속도 올린다()' 와 '속도 내린다()'를 정의해서 자동차의 인스턴스 마다의 속성(속도)를 변경 할 수 있다.

4)특징

객체지향 언어의 특성에는 '추상화'(abstraction), '상속'(inheritance), '캡슐화'(encapsulation), '다형성'(polymorphism), '메시지'(message), '인터페이스'(interface)등이 있다. 본인은 상속, 캡슐화 , 다형성에 대해서 설명하겠다.

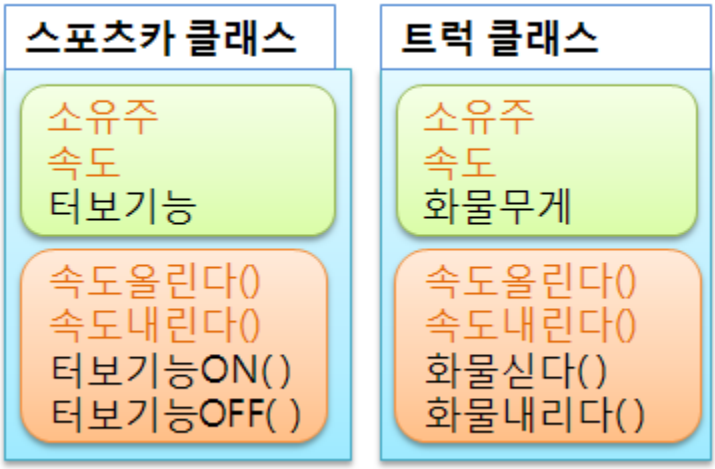
(1)상속

상속은 하위 클래스(자식클래스)가 상위 클래스(부모클래스)의 모든 특성(속성과 행동(메소드))을 이어받는 것을 말한다.(객체지향 프로그래밍에서 상속은 코드 재사용의 가장 기본적인 내용)

앞서 예를들어 설명한 자동차 클래스(부모클래스)에서 자식클래스(스포츠카 , 트럭 클래스)를 예로 설명하겠다. 하나의 클래스인 '자동차 클래스'가 있을 때 또 다른 클래스이 '스포츠카와 트럭 클래스'는 자동차 클래스의 모든 속성(소유주, 속도)과 행동(속도 올린다, 속도 내린다)를 이어받는다.

스포츠카 클래스나 트럭 클래스는 좀 더 일반적이고 추상적인 자동차클래스에 속하며 스포츠카 클래스는 스포츠카 만의 행동인 터보기능 ON,OFF를 수행 할 수 있고 , 트럭클래스는 화물을 싣고 내리는 행동을 수행 할 수 있다. 공통적인 특성으로는 스포츠카도 자동차이므로 속성(소유주와 속도)을 가지며 행동(속도 올린다,속도 내린다)을 갖는다. 트럭도 마찬가지이다.이렇게 스포츠카와 트럭은 자동차의 모든 특성인 속성과 행동을 상속받게 된다.

아래는 자동차 클래스 상속 후 스포츠카와 트럭 클래스가 가지게 되는 속성과 행동이다.



상속에서 추상적인고 일반적인 클래스를 상위 클래스(super class), 부모 클래스(parent

class)라하며 구체적인 클래스를 하위 클래스(sub class), 자식 클래스(child class)라고 한다.

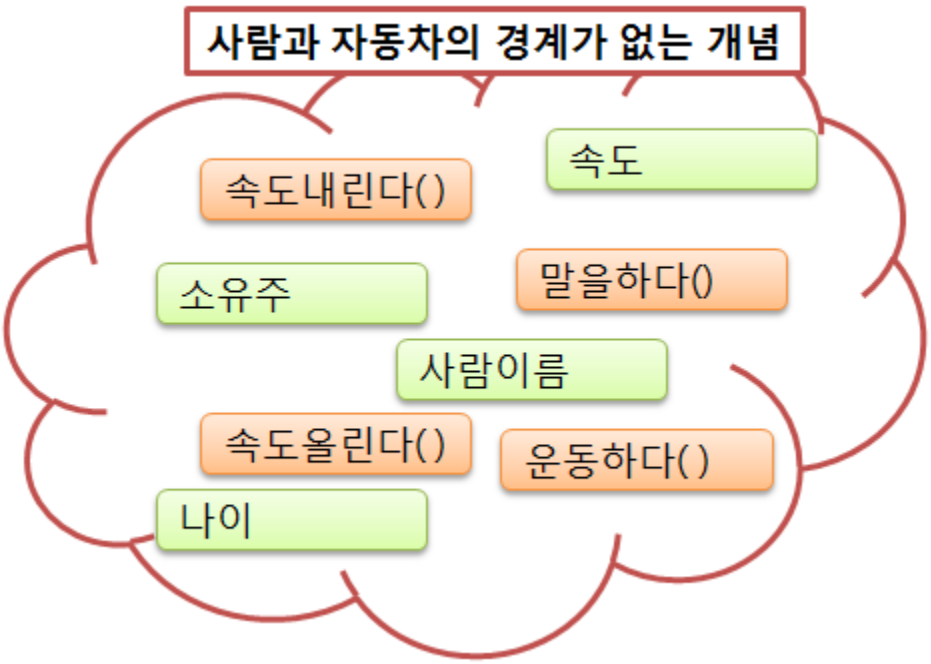
(2)캡슐화

캡슐화는 크게 두가지 개념으로 설명할 수 있다.

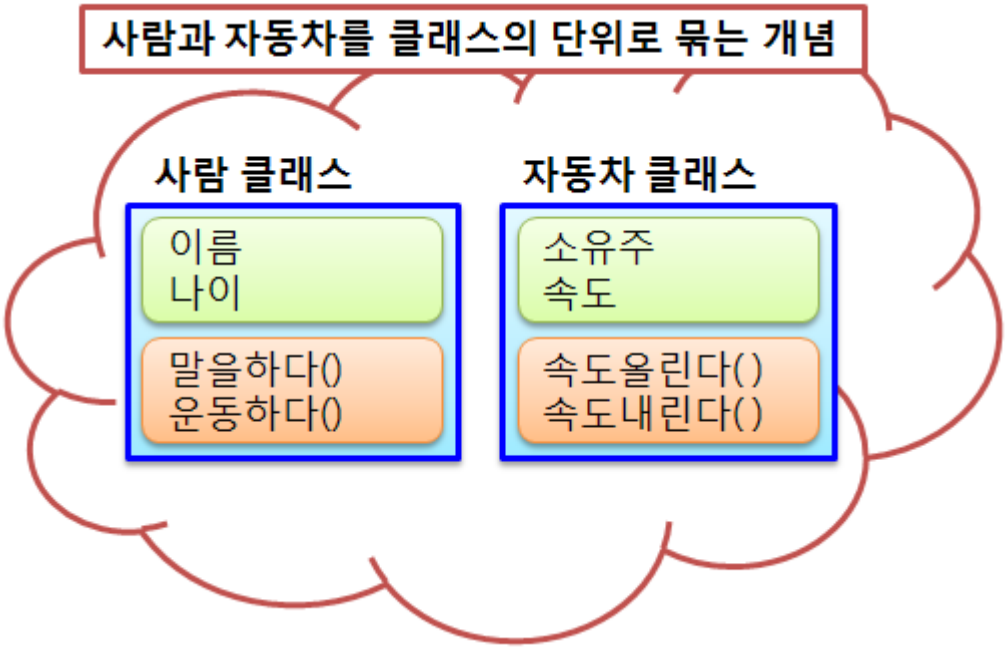
첫째, 하나의 단위로 묶는 개념.

객체는 속성과 행동을 가지고 다른 객체와 고유하게 구분되어 독립적으로 존재한다. 이때 속성과 행동을 묶어 다른 객체와 고유하게 구분하는 개념이 캡슐화이다.

간단한 예로 아래 자동차와 사람이라는 속성과 행동이 있을 때



속성과 행동을 공통되고 유사한 개념을 갖는 클래스로 묶습니다.

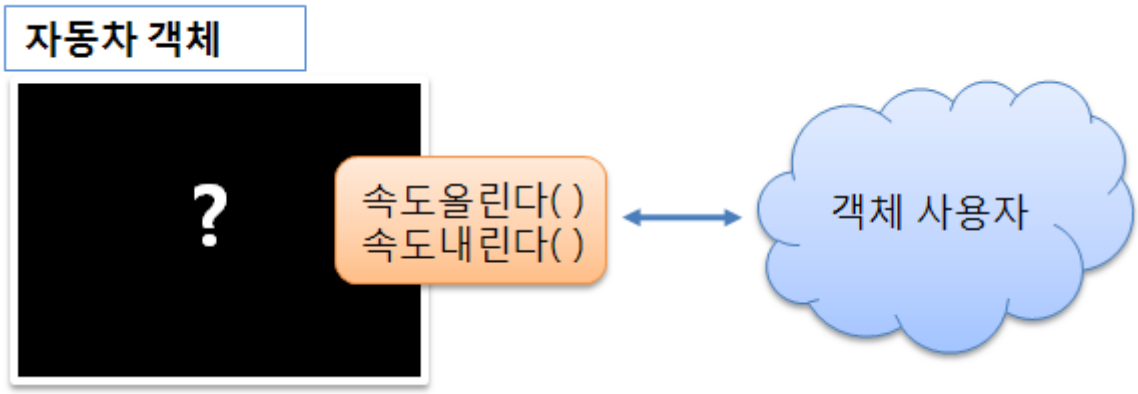


둘째, 정보은닉(information hiding)의 개념.

객체지향에서 정보은닉이란 '객체의 불필요한 세세한 내용은 객체 사용자에게 노출 시키지 않는 것'으로 본다. 즉 객체는 속성과 행동으로 이루어지므로 이 객체의 속성과 행동에서 객체 사용자에게 꼭 필요한 것만 노출(공개)하는 것을 말한다.

예로 자동차 객체를 사용하는 객체가 속도를 올리고 내리는 것 외에 알 필요가 없다면 자동차 객체는 두 행동(속도 올린다, 속도 내린다) 외엔 모두 감추게 되는것이다.

그래서 객체를 행동들만의 집합으로 보고 프로그래밍을 하게 되는데, 이러면 프로그래밍이 더 쉽고 명확해진다는 장점이 있다.

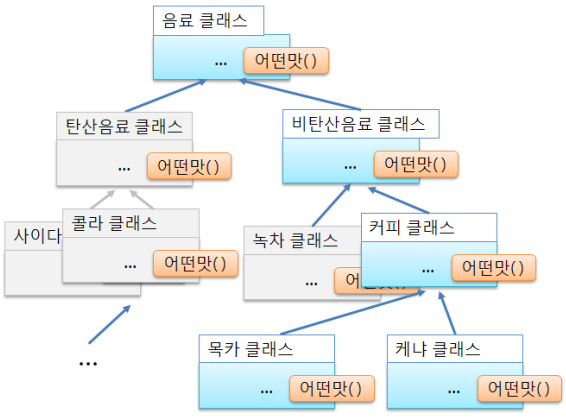


자동차 객체를 사용하는 클라이언트 입장에서는 자동차 객체의 세세한 내용을 알 필요 없이 자동차 객체와 소통(통신)하므로 자동차 객체를 사용하고 다루기가 쉬울뿐더러 자동차 객체도 객체 자신을 변경하고 확장쉽게 되는 것이다.

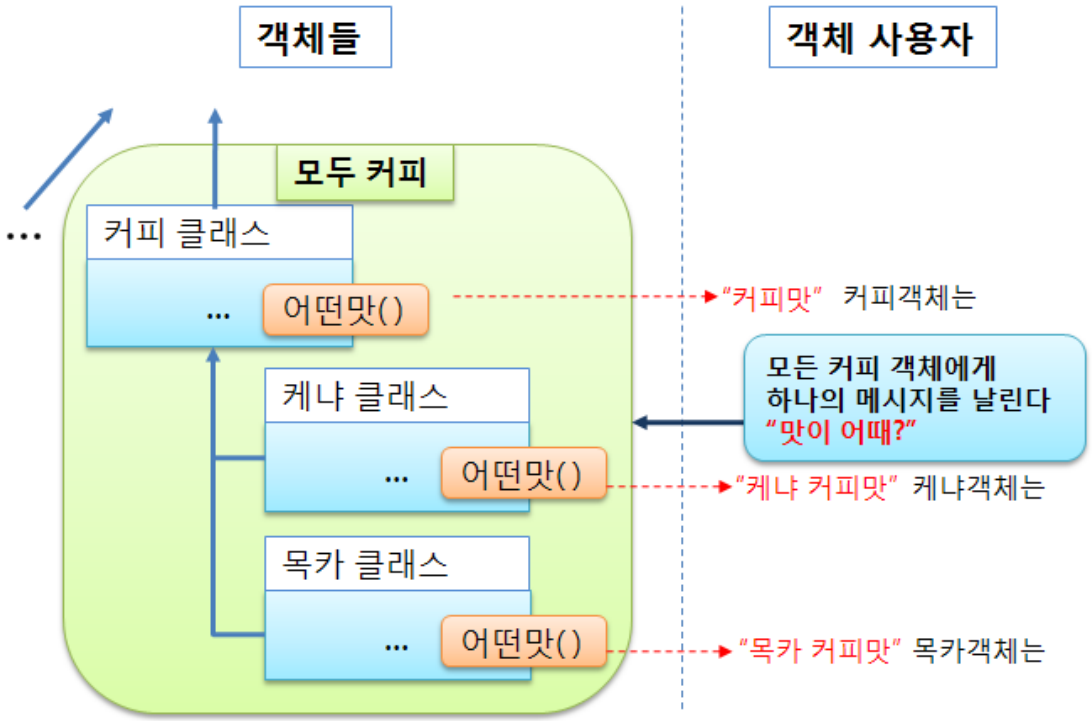
(3)다형성

다형성은 상호 관련된 객체들을 동일한 방식으로 다루는 개념을 말한다. 다시 말하면 메소드 호출 시 정확한 객체의 메소드(행동)가 호출되는 것이다.

아래 그림처럼 계층구조를 갖는 음료 클래스가 있다고 가정하면, 모든 음료 클래스는 맛을 응답할 수 있는 메소드를 가지고 있다.



이때, 커피에게 맛을 물어보면(맛에 대한 메서드를 호출하면) 각 클래스의 객체마다 자신의 객체에 맞는 응답(메소드 호출됨)을 한다. 커피 객체는 "커피맛'이다, 케냐 객체는 "케냐 커피맛"이다, 목카 객체는 "목카 커피맛"이다고 응답한다. 이렇게 관련된 객체들(커피 객체들)에게 하나의 메시지("맛이 어때?")를 보내면 자신의 객체에 맞게 응답(메소드가 호출됨)하는 것을 다형성이라고 한다.

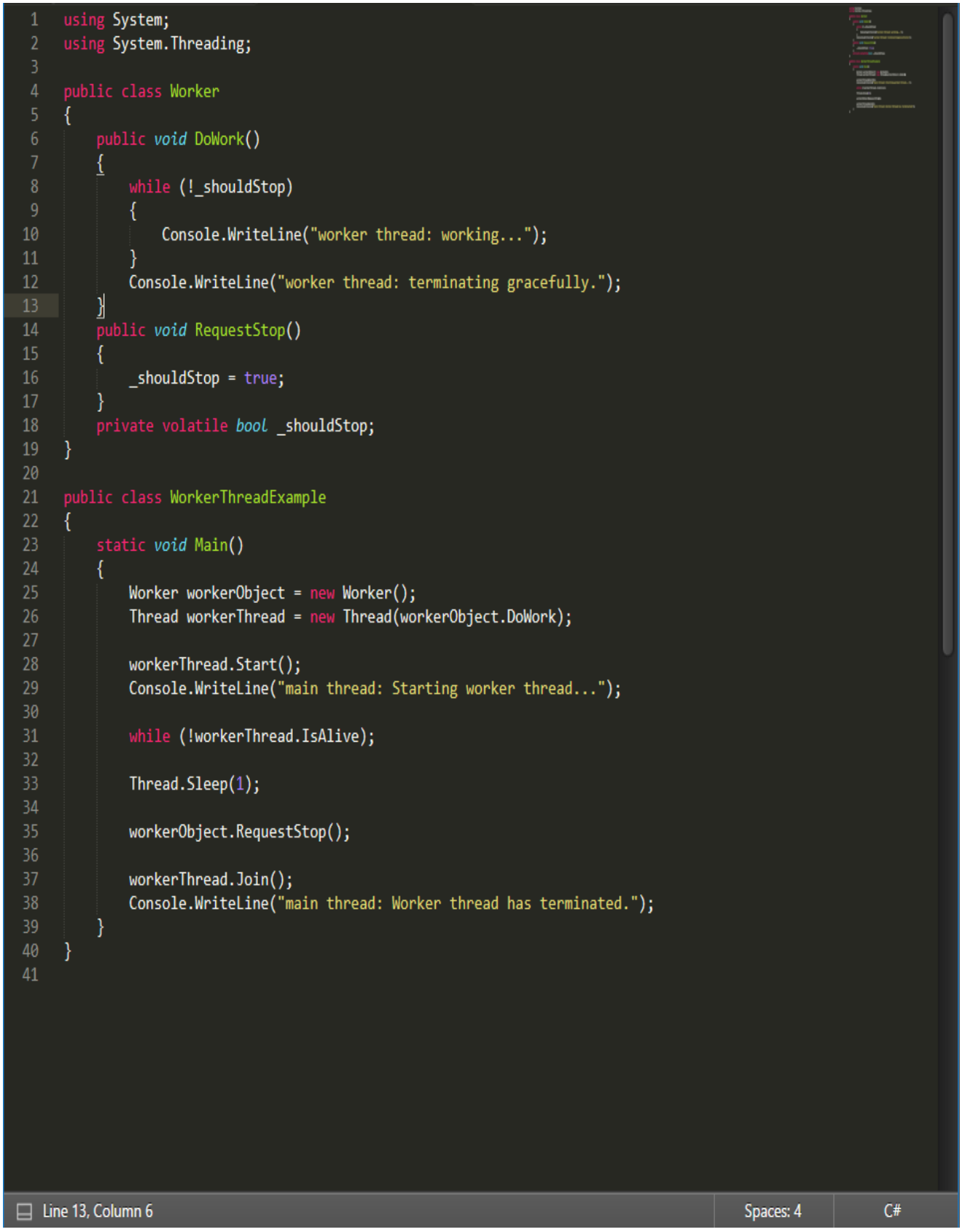


d. C#을 사용한 예제

'경주 시뮬레이션'을 통해 실습을 진행하겠다.

이 예제에서는 작업자 스레드를 만들고 기본 스레드와 함께 이 스레드를 병렬로 사용하여 작업 처리를 수행하는 방법을 보여 준다. 또한 다른 스레드의 작업이 끝날 때까지 한 스레드가 대기하도록 만들고 스레드를 올바르게 종료하는 방법도 사용한다.

(예제코드첨부)



```
using System;
using System.Threading;

public class Worker
{
    public void DoWork()
    {
        while (!_shouldStop)
        {
            Console.WriteLine("worker thread: working...");
        }
        Console.WriteLine("worker thread: terminating gracefully.");
    }
    public void RequestStop()
    {
        _shouldStop = true;
    }
    private volatile bool _shouldStop;
}

public class WorkerThreadExample
{
    static void Main()
    {
        Worker workerObject = new Worker();
        Thread workerThread = new Thread(workerObject.DoWork);

        workerThread.Start();
        Console.WriteLine("main thread: Starting worker thread...");

        while (!workerThread.IsAlive);

        Thread.Sleep(1);

        workerObject.RequestStop();

        workerThread.Join();
        Console.WriteLine("main thread: Worker thread has terminated.");
    }
}

```

Line 13, Column 6 | Spaces: 4 | C#

(코드설명)

이 예제에서는 작업자 스레드에서 실행할 DoWork 메서드가 포함된 Worker라는 클래스를 만듭니다. 이는 작업자 스레드의 Main 함수이다. 작업자 스레드는 이 메서드를 호출하여 실행을 시작하고 이 메서드가 반환될 때 자동으로 종료됩니다. DoWork 메서드는 다음과 같다.

```
6       public void DoWork()
7       {
8           while (!_shouldStop)
9           {
10              Console.WriteLine("worker thread: working...");
11          }
12          Console.WriteLine("worker thread: terminating gracefully.");
13      }
```

Worker 클래스에는 DoWork에 반환할 시기를 알리는 데 사용되는 추가 메서드가 포함됩니다. RequestStop이라는 이 메서드는 다음과 같다.

```
14      public void RequestStop()
15      {
16          _shouldStop = true;
17      }
```

RequestStop 메서드는 _shouldStop 데이터 멤버를 true로 할당한다. 이 데이터 멤버는 DoWork 메서드에서 검사하므로 이는 간접적으로 DoWork가 반환되도록 하여 결과적으로 작업자 스레드가 종료된다. DoWork는 작업자 스레드에서 실행되고 RequestStop은 기본 스레드에서 실행되므로 _shouldStop 데이터 멤버는 다음과 같이 volatile로 선언한다..

```
18      private volatile bool _shouldStop;
```

작업자 스레드를 실제로 만들기 전에 Main 함수에서 Worker 개체와 Thread의 인스턴스를 만든다. 스레드 개체는 Worker.DoWork 메서드에 대한 참조를 다음과 같이 Thread 생성자에 전달하여 이 메서드를 진입점으로 사용하도록 구성하였다.

```
25          Worker workerObject = new Worker();
26          Thread workerThread = new Thread(workerObject.DoWork);
```

이 시점에서는 작업자 스레드 개체가 존재하고 구성되어 있지만 실제 작업자 스레드는 아직 작성되어 있지 않다. 실제 작업자 스레드는 Main에서 Start 메서드를 호출해야 작성된다.

```
28          workerThread.Start();
```

이제 시스템에서 작업자 스레드의 실행을 초기화하지만 이 과정은 기본 스레드에 대해 비동기적으로 수행된다. 즉, Main 함수는 작업자 스레드를 동시에 초기화하는 동안 코드를 계속하여 즉시 실행하는데, 작업자 스레드가 실행되기도 전에 Main 함수가 이 스레드를 종료하지 않도록 하기 위해 Main 함수는 작업자 스레드 개체의 IsAlive 속성이 true로 설정될 때까지 반복된다

```
31          while (!workerThread.IsAlive);
```

그런 다음 Sleep을 호출하여 기본 스레드를 잠시 중단한다. 이렇게 하면 Main 함수가 다른 명령을 실행하기 전에 작업자 스레드의 DoWork 함수에서 DoWork 메서드 안의 루프를 몇 차례 반복하여 실행할 수 있다.

```
33          Thread.Sleep(1);
34
```

1밀리초가 경과하면 Main에서는 앞서 설명한 Worker.RequestStop 메서드를 사용하여 작업자 스레드 개체를 종료하도록 신호를 보낸다.

```
35          workerObject.RequestStop();
36
```

마지막으로, Main 함수가 작업자 스레드 개체에 대한 Join 메서드를 호출한다. 이 메서드는 개체가 가리키는 스레드가 종료될 때까지 현재 스레드를 차단하거나 대기 상태로 만드는데, 따

라서 Join은 작업자 스레드가 반환되고 자체 종료될 때까지 반환되지 않는다.

```
37          workerThread.Join();
38          Console.WriteLine("main thread: Worker thread has terminated.");
```

이 단계에서는 Main을 실행하는 기본 스레드만 남게 되는데, 이 스레드는 최종 메시지 하나를 표시한 다음 반환되고 종료된다.

(출력결과)

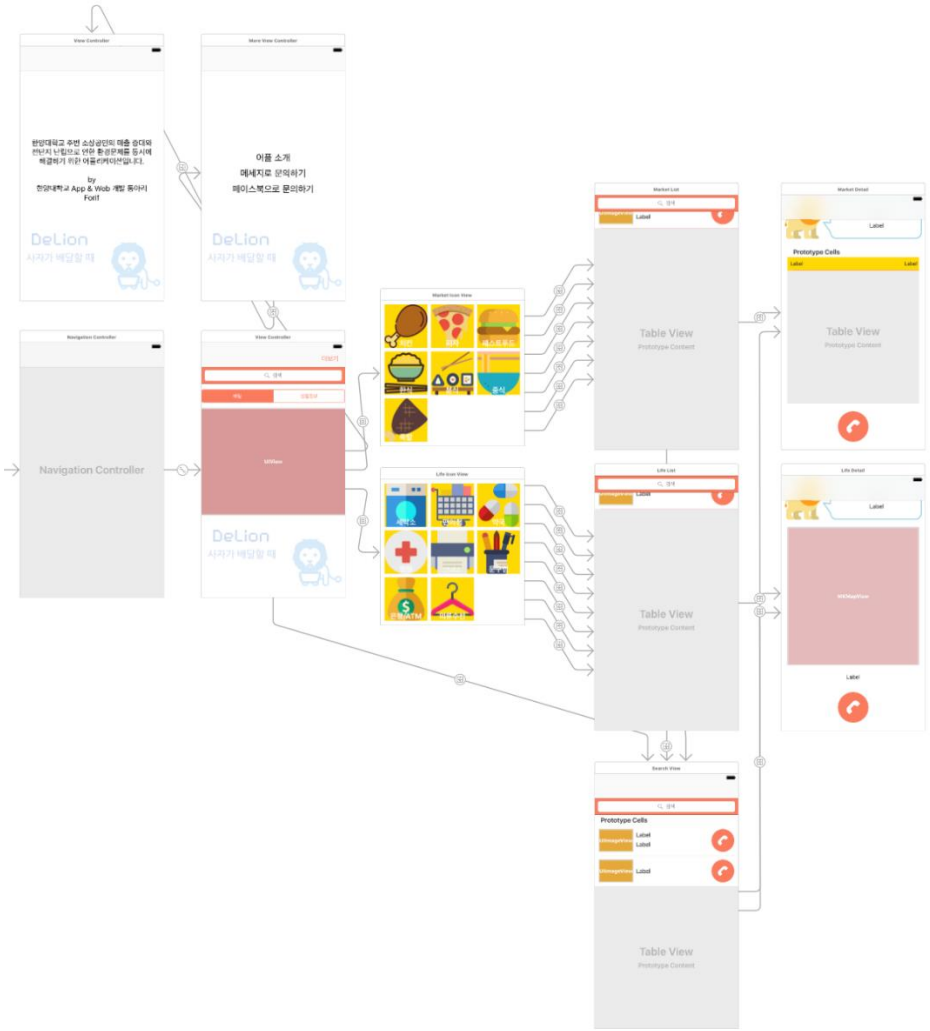
```
main thread: starting worker thread...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: working...
worker thread: terminating gracefully...
main thread: worker thread has terminated
```

4. Xamarin을 사용한 실제 어플리케이션 개발

앞서 Android와 IOS의 차이로 인해 발생하는 문제점, 그리고 그로 인해 나타나게 된 Xamarin 크로스 플랫폼 및 이에 사용되는 C# 프로그래밍 언어에 대해서 연구해보았다. 본격적으로 이러한 플랫폼을 이용하여 교내 배달 어플리케이션인 '딜라이온'을 개발한 것에 대해 설명할 것이다.

a. 전체 UI

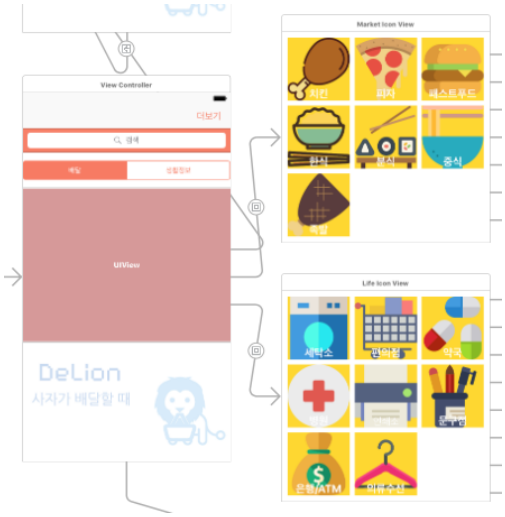
기본적으로 모든 어플리케이션에는 UI가 존재한다. 우리는 Xamarin을 이용해 Android와 IOS의 UI를 한 번에 작업할 수 있었다. 다음은 '딜라이온'의 전체적 UI 틀이다.



또한 총 15개의 코드 파일이 존재하지만, 그 중 핵심인 Main 코드, List 코드(여기서 Market-배달업체 와 Life-편의시설의 List 코드는 유사해 Market List 코드만 다룸.), Detail 코드(여기서 Market-배달업체 와 Life-편의시설의 Detail 코드는 각각 다른 기능을 수행하기 때문에 둘 모두 다룸.)만을 소개할 예정이다. (더 보기 메뉴나 검색 창에 대한 구현은 크게 중요하지 않기 때문에 생략한다.)

b. Main 구현 코드

Main화면이란 처음 어플리케이션에 들어갔을 때 나타나는 UI에 대한 작동을 구현하는 부분으로, 앞서 보여준 UI에서 다음 부분이 Main에 해당된다.



다음은 Main 코드 사진과 그에 대한 간단한 설명이다.

```swift
class ViewController: UIViewController, UISearchBarDelegate {

    // 메인 아울렛

    @IBOutlet weak var mainSegment: UISegmentedControl! // 세그먼트(배달,생활정보)
    @IBOutlet weak var mainContainer1: UIView! // 배달 Icon View
    @IBOutlet weak var mainContainer2: UIView! // 생활정보 Icon View
    @IBOutlet weak var searchBar: UISearchBar! // 검색창


    override func viewDidLoad() {
        super.viewDidLoad()
        self.title = "딜라이온"
        self.searchBar.delegate = self

        // 메인 컨테이너 세팅
        mainContainer1.isHidden = false
        mainContainer2.isHidden = true
        mainContainer1.layer.borderWidth = 5
        mainContainer2.layer.borderWidth = 5
        mainContainer1.layer.borderColor = UIColor(red: 245/255, green: 122/255, blue: 98/255, alpha: 1.0).cgColor
        mainContainer2.layer.borderColor = UIColor(red: 245/255, green: 122/255, blue: 98/255, alpha: 1.0).cgColor
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    // 세그먼트(배달,생활정보) 클릭 시
    @IBAction func mainSegmentAction(sender: AnyObject) {
        if(mainSegment.selectedSegmentIndex == 0) // 배달
        {
            mainContainer1.isHidden = false
            mainContainer2.isHidden = true
        }
        else if(mainSegment.selectedSegmentIndex == 1) // 생활정보
        {
            mainContainer1.isHidden = true
            mainContainer2.isHidden = false
        }
    }

    // 검색바 클릭 시 searchSegue 활성화
    func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
        performSegue(withIdentifier: "searchSegue", sender: nil)
        self.searchBar.text = ""
    }

    // searchSegue 활성화 시, 검색 스트링 넘김
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        let searchresult = segue.destination as? SearchView
        searchresult?.searchstring = self.searchBar.text!
    }
}
```

- viewDidLoad 메소드 : 프론트(container layer)를 Android, IOS 동시에 세팅하는 역할을 한다.

- mainSegmentAction 메소드 : 배달/생활정보 버튼을 클릭 시 알맞은 아이콘 세트가 나타나게 해주는 역할을 한다.

- searchBarSearchButtonClicked 메소드 : 검색 바를 클릭할 경우, 검색이 원활히 되도록 구현해 놓은 searchSegue가 수행되도록 하는 역할을 한다.

- prepare 메소드 : 검색 바에 검색 스트링을 입력했을 경우, 이를 SearchView를 통해 결과를 출력하도록 하는 역할을 한다.

c. Market List 구현 코드

List 화면이란 Main화면에서 해당 아이콘 클릭 시, 그에 맞는 카테고리의 업체들 정보를 서버에서 불러와 List 형식으로 출력해주는 부분으로, 앞서 보여준 UI에서 다음 부분이 List에 해당된다.

각각 Market List와 Life List 코드를 자신의 작동 코드로 보유하고 있으나, 둘 모두 유사한 기능을 하기 때문에 Market List 코드 만을 설명할 것이다. 또한, 코드가 매우 길기 때문에 중요한 메소드만 부분 첨부했다. 다음이 Market List 코드 사진이며, 그에 대한 간단한 설명을 첨부한다.

```
// 테이블 셀 클릭 시 그 마켓정보를 MarketDetail으로 전달
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "detailSegue"{
        let destination = segue.destination as? MarketDetail
        let dic = values[(tableView.indexPathForSelectedRow?.row)!] as! NSDictionary
        destination?.markettitle = dic["name"] as! String
        destination?.marketid = dic["id"] as! String
    }

    // searchSegue 활성화 시, 검색 스트링 넘김
    else if segue.identifier == "searchSegue"{
        let searchresult = segue.destination as? SearchView
        searchresult?.searchstring = self.searchBar.text!
    }
}

// db연결
func getFromJSON(){
    let url = NSURL(string: "http://222.239.250.218/delion/index.php/shop/shop_list/\(shoptype)")
    let request = NSMutableURLRequest(url: url! as URL)

    let task = URLSession.shared.dataTask(with: request as URLRequest){data,response,error in
    guard error == nil && data != nil else
    {
        print("Error:", error as Any)
        return
        }

        let httpStatus = response as? HTTPURLResponse

        if httpStatus?.statusCode == 200
        {
            if data?.count != 0
            {
                self.values = try! JSONSerialization.jsonObject(with: data!, options: .allowFragments) as! NSArray
                DispatchQueue.main.sync {
                    self.tableView.reloadData()
                }
            }
        }
        else
        {
            print("error httpstatus code is :", httpStatus!.statusCode)
        }
    }
    task.resume()
}
```

- prepare 메소드 : List 화면에서 해당 리스트를 클릭했을 경우, 그에 대한 정보를 MarketDetail 화면에 넘겨주는 역할을 한다.

- getFromJSON 메소드 : Xamarin 플랫폼 역시 JSON을 이용하여 데이터베이스의 정보들을 불러온다. 데이터베이스에 적절한 데이터들(이 경우, 하나의 업체 당 여러 개의 정보를 필요로 하기 때문에 NSArray를 이용했음.)을 뽑아 리스트를 reload하는 역할을 한다.

```
    // 그 셀마다 들어가야 하는 것 배정
    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = self.tableView.dequeueReusableCell(withIdentifier: "marketcell", for: indexPath) as! MarketCell
        let maindata = values[indexPath.row] as! NSDictionary

        cell.marketName.text = maindata["name"] as? String
        cell.marketPlace.text = maindata["branch"] as? String
        if maindata["img"] as? String == "null"
        {
            cell.marketPhoto.image = defaultimg
        }
        else
        {
            let url = NSURL(string: "\(maindata["img"] as! String)")
            let imageData:NSData = NSData(contentsOf: url as! URL)!

            DispatchQueue.main.async {
                let image = UIImage(data: imageData as Data)
                cell.marketPhoto.image = image
            }
        }

        cell.callButton.setTitle("tel://\(maindata["phone"] as! String)", for: .normal)

        return cell
    }

    // 마켓 리스트 갯수만큼 출력
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.values.count
    }

    // 마켓 리스트 높이 지정
    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return 80
    }

    // 전화버튼 클릭 시
    @IBAction func callMarket(sender: UIButton) {
        if sender.currentTitle == "tel://"{ // 전화번호가 없다면
            let alertController = UIAlertController(title: "알림", message: "위 매장은 전화번호가 없습니다.", preferredStyle: UIAlertControllerStyle.
                alert)
            let okAction = UIAlertAction(title: "확인", style: .default, handler: nil)
            alertController.addAction(okAction)
            self.present(alertController, animated: true, completion: nil)
        }
        else{ // 전화번호가 있다면 연결
            let phoneURL = NSURL(string: sender.currentTitle!)
            if #available(iOS 10.0, *) {
                UIApplication.shared.open(phoneURL as! URL, options: [:], completionHandler: nil)
            } else {
                UIApplication.shared.openURL(phoneURL as! URL)
            }
        }
    }
}
```

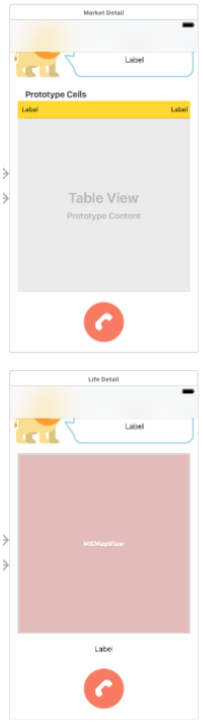
- tableView 메소드 : 세 개의 tableView 메소드가 존재한다. 뒤의 두 메소드의 경우 List의 프론트 세팅의 역할을 하고, 앞의 메소드의 경우 getFromJSON 메소드가 실행되면 데이터베이스에서 가져온 데이터들(name, branch, img, phone이 이에 해당함.)을 List의 각각의 cell에 세팅해주는 역할을 한다.

- callMarket 메소드 : List에서 전화버튼을 클릭할 경우, 해당 버튼에 저장되어있는 전화로 연결해주는 역할을 한다. 번호가 없는 업체의 경우, 알람창을 띄워준다.

d. Detail 구현 코드

Detail화면이란 List화면에서 해당 리스트 클릭 시, 그에 대한 업체 정보를 서버에서 불러와 ListView를 이용해 출력해주는 부분으로, 앞서 보여준 UI에서 다음 부분이 Detail에 해당

된다.



위에서부터 각각 Market Detail과 Life Detail코드를 자신의 작동 코드로 보유하고 있으며, UI적으로만 보아도 둘의 기능을 확연히 다르다. 따라서 각각의 중요 메소드만을 사진으로 보이고 간단한 설명을 덧붙인다.

d -1. Market Detail 구현 코드

Market Detail 화면의 경우, 해당 업체의 영업시간 및 전화번호, 메뉴를 묶어 큰 메뉴 섹션과 그에 해당하는 메뉴들과 가격을 보여준다. 또한 맨 밑에는 List화면과 마찬가지로 전화버튼이 존재한다. 다음은 Market Detail 부분의 코드와 간단한 설명이다.

```
class MarketDetail: UIViewController, UITableViewDataSource, UITableViewDelegate{

    // 배달 디테일 화면 (db연동)

    @IBOutlet weak var tableView: UITableView! // 메뉴 표
    @IBOutlet weak var marketTime: UILabel! // 마켓영업시간
    @IBOutlet weak var marketPhone: UILabel! // 마켓전화번호
    @IBOutlet weak var callButton: UIButton! // 전화버튼

    var selectedIndexPath: IndexPath? = nil
    var values:NSArray = []

    var titles:[String] = []
    var items:[[String]] = [[]]
    var prices:[[String]] = [[]]
    var titlenum = 0

    var markettitle:String = ""
    var marketid:String = ""
    override func viewDidLoad() {
        super.viewDidLoad()
        self.title = markettitle
        tableView.layer.borderWidth = 3
        tableView.layer.borderColor = UIColor(red: 245/255, green: 122/255, blue: 98/255, alpha: 1.0).cgColor
        self.marketTime.adjustsFontSizeToFitWidth = true

        self.getFromJSON()
    }
```

- viewDidLoad 메소드 : 역시나 프론트를 세팅한 후, 바로 getFromJSON 메소드를 부르는 역할을 한다.

```
// db연결
func getFromJSON(){
    let url = NSURL(string: "http://222.239.250.218/delion/index.php/shop/shop_menu/\(marketid)")
    let request = NSMutableURLRequest(url: url! as URL)

    let task = URLSession.shared.dataTask(with: request as URLRequest){data,response,error in
        guard error == nil && data != nil else
        {
            print("Error:", error as Any)
            return
        }

        let httpStatus = response as? HTTPURLResponse

        if httpStatus?.statusCode == 200
        {
            if data?.count != 0
            {
                self.values = try! JSONSerialization.jsonObject(with: data!, options: .allowFragments) as! NSArray
                let setting = self.values[0] as! NSDictionary

                DispatchQueue.main.sync {
                    self.marketTime.text = "영업시간 : \(setting["opentime"] as! String)"
                    self.marketPhone.text = "tel : \(setting["phone"] as! String)"
                    self.callButton.setTitle("tel://\(setting["phone"] as! String)", for: .normal)

                    self.titles.append(setting["extender_menu"] as! String)
                    self.items[self.titlenum].append(setting["menu"] as! String)
                    self.prices[self.titlenum].append("\(setting["price"] as! String)원")

                    for i in 1 ..< self.values.count {
                        let dic = self.values[i] as! NSDictionary
                        if dic["extender_menu"] as! String == self.titles[self.titlenum] //같은 section일 경우
                        {
                            self.items[self.titlenum].append(dic["menu"] as! String)
                            self.prices[self.titlenum].append("\(dic["price"] as! String)원")
                        }
                        else // 다른 section일 경우
                        {
                            self.titlenum += 1
                            self.titles.append(dic["extender_menu"] as! String)
                            self.items.append([dic["menu"] as! String])
                            self.prices.append(["\(dic["price"] as! String)원"])
                        }
                    }
                    self.tableView.reloadData()
                }
            }
        }
        else
        {
            print("error httpstatus code is :", httpStatus!.statusCode)
        }
    }
    task.resume()
}
```

- getFromJSON 메소드 : 해당 Detail화면에 필요한 정보들을 데이터베이스에 알맞게 접근해 정보들(opentime, phone, extender_menu, menu, price가 이에 해당함.)을 뽑아와 세팅해주는 역할을 한다. extender_menu의 경우 메뉴들을 크게 묶은 메뉴들로써 이는 list의 section으로 구현하였다.

- 이외에도 앞서 List 화면에서 설명한 것과 유사한 callMarket 메소드와 tableView 메소드가 여럿 존재하지만 생략한다.

d-2 Life Detail 구현 코드

Life Detail 화면의 경우, 해당 생활편의 업체의 영업시간 및 전화번호, 업체 주소 및 위치를 지도를 이용해 보여준다. 또한 맨 밑에는 List화면과 마찬가지로 전화버튼이 존재한다. 다음은 Life Detail 부분의 코드와 간단한 설명이다.

```
class LifeDetail : UIViewController{

    // 생활정보 디테일 화면 (db연동)

    @IBOutlet weak var lifeMap: MKMapView! // 구글지도
    @IBOutlet weak var lifeTime: UILabel! // 마켓영업시간
    @IBOutlet weak var lifePhone: UILabel! // 마켓전화번호
    @IBOutlet weak var lifeAddress: UILabel! // 마켓주소
    @IBOutlet weak var callButton: UIButton! // 전화버튼

    var values:NSArray = []

    var lifetitle:String = ""
    var lifeid:String = ""
    var xpoint:Double = 0.0
    var ypoint:Double = 0.0
    override func viewDidLoad() {
        super.viewDidLoad()
        self.title = lifetitle
        lifeMap.layer.borderWidth = 3
        lifeMap.layer.borderColor = UIColor(red: 149/255, green: 205/255, blue: 219/255, alpha: 1.0).cgColor
        lifeAddress.layer.borderWidth = 3
        lifeAddress.layer.borderColor = UIColor(red: 149/255, green: 205/255, blue: 219/255, alpha: 1.0).cgColor
        self.lifeTime.adjustsFontSizeToFitWidth = true
        self.lifeAddress.adjustsFontSizeToFitWidth = true

        // 지도구현
        let span:MKCoordinateSpan = MKCoordinateSpanMake(0.01, 0.01)
        let location:CLLocationCoordinate2D = CLLocationCoordinate2DMake(ypoint as CLLocationDegrees, xpoint as CLLocationDegrees)
        let region:MKCoordinateRegion = MKCoordinateRegionMake(location, span)
        lifeMap.setRegion(region, animated: true)

        let annotation = MKPointAnnotation()

        annotation.coordinate = location
        annotation.title = lifetitle
        lifeMap.addAnnotation(annotation)


        self.getFromJSON()
    }
```

- viewDidLoad 메소드 : 기존의 viewDidLoad 메소드와의 차이점은 이 메소드 내에서 지도를 구현한다는 점이다. Xamarin 플랫폼에서 제공해주는 지도구현을 위한 다양한 함수를 이용함으로써 Android, IOS의 지도를 동시에 편리하게 구현하였다.

```
// db연결
func getFromJSON(){
    let url = NSURL(string: "http://222.239.250.218/delion/index.php/lifeinfo/life_detail/\(lifeid)")
    let request = NSMutableURLRequest(url: url! as URL)

    let task = URLSession.shared.dataTask(with: request as URLRequest){data,response,error in
        guard error == nil && data != nil else
        {
            print("Error:", error as Any)
            return
        }

        let httpStatus = response as? HTTPURLResponse

        if httpStatus?.statusCode == 200
        {
            if data?.count != 0
            {
                self.values = try! JSONSerialization.jsonObject(with: data!, options: .allowFragments) as! NSArray
                let setting = self.values[0] as! NSDictionary

                DispatchQueue.main.sync {
                    self.lifeTime.text = "영업시간 : \(setting["opentime"] as! String)"
                    self.lifePhone.text = "tel : \(setting["phone"] as! String)"
                    self.lifeAddress.text = setting["life_add"] as? String
                    self.callButton.setTitle("tel://\(setting["phone"] as! String)", for: .normal)
                }
            }
        }
        else
        {
            print("error httpstatus code is :", httpStatus!.statusCode)
        }
    }
    task.resume()
}
```

- getFromJSON 메소드 : 해당 Detail화면에 필요한 정보들을 데이터베이스에 알맞게 접근해 정보들(opentime, phone, life_add가 이에 해당함.)을 뽑아와 세팅해주는 역할을 한다.

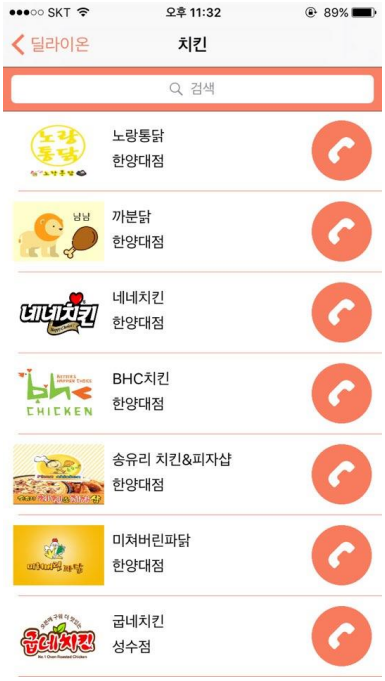
- 이외에도 앞서 List 화면에서 설명한 것과 유사한 callMarket 메소드가 존재하지만 생략한다.

e. 딜라이온 with Xamarin

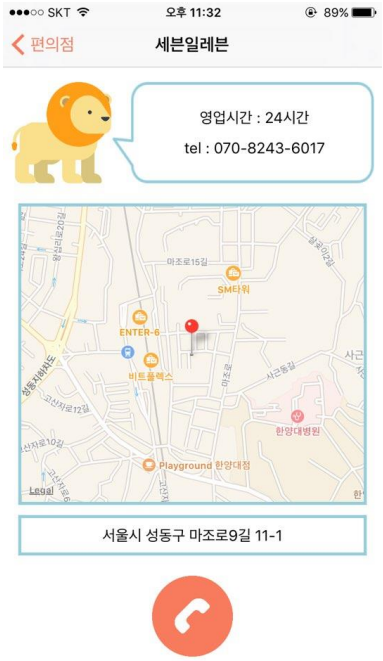
이전의 방식대로였다면 Android는 Android Studio에서 JAVA 언어를 사용해서, 그리고 IOS는 Xcode에서 swift 혹은 object-c 언어를 사용해서 각각 개발 했어야만 했다. 하지만 우리는 새로운 크로스 플랫폼인 Xamarin을 이용해서 간단한 어플리케이션을 완성할 수 있었다. 약간의 불안정함에 있어서 각각의 OS에 대해 수정을 해야하는 부분도 있었지만, 간단한 뷰는 모두 한 번에 개발을 진행할 수 있었다.

현재 ‘딜라이온’은 Android, IOS 양쪽 모두 정상 작동되며 앞으로 발생하는 오류에 대한 유지/보수 또한 Xamarin 플랫폼을 이용한다면 쉽게 해결할 수 있을 것이라 예상된다.

마지막으로 현재 ‘딜라이온’의 작동 사진을 첨부한다. 순서대로 Main, List, Market Detail, Life Detail 화면이다.




●●●○○ SKT
오후 11:32
89%
치킨
노랑통닭
영업시간 : 12:00~05:00
tel : 02-2291-0800
세트메뉴
3종세트(후라이드+양념+깐풍) 18000원
웰빙파닭(후라이드+양념) 18000원
웰빙파닭(후라이드+깐풍기) 18000원
웰빙파닭(양념+깐풍기) 19000원
치킨
후라이드치킨(뼈or순살) 16000원
양념치킨(뼈or순살) 17000원
깐풍치킨(뼈or순살) 17000원
매콤후라이드 17000원
사이드메뉴


●●●○○ SKT
오후 11:32
89%
편의점
세븐일레븐
영업시간 : 24시간
tel : 070-8243-6017
SM타워
ENTER-6
비트플렉스
한양대병원
Playground 한양대점
서울시 성동구 마조로9길 11-1

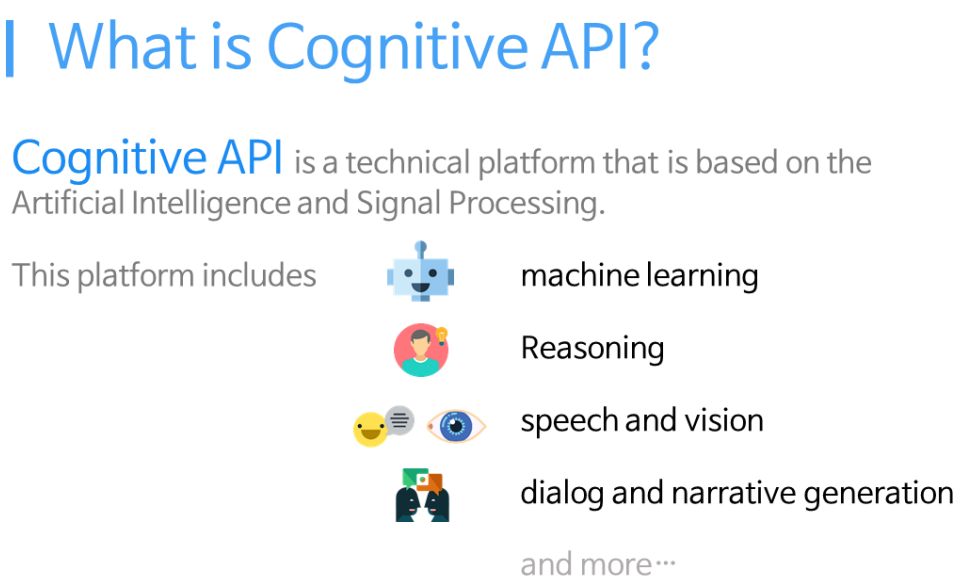
5. 다양한 API 적용 - Cognitive API with Xamarin

a. Xamarin의 API 호환성 검증

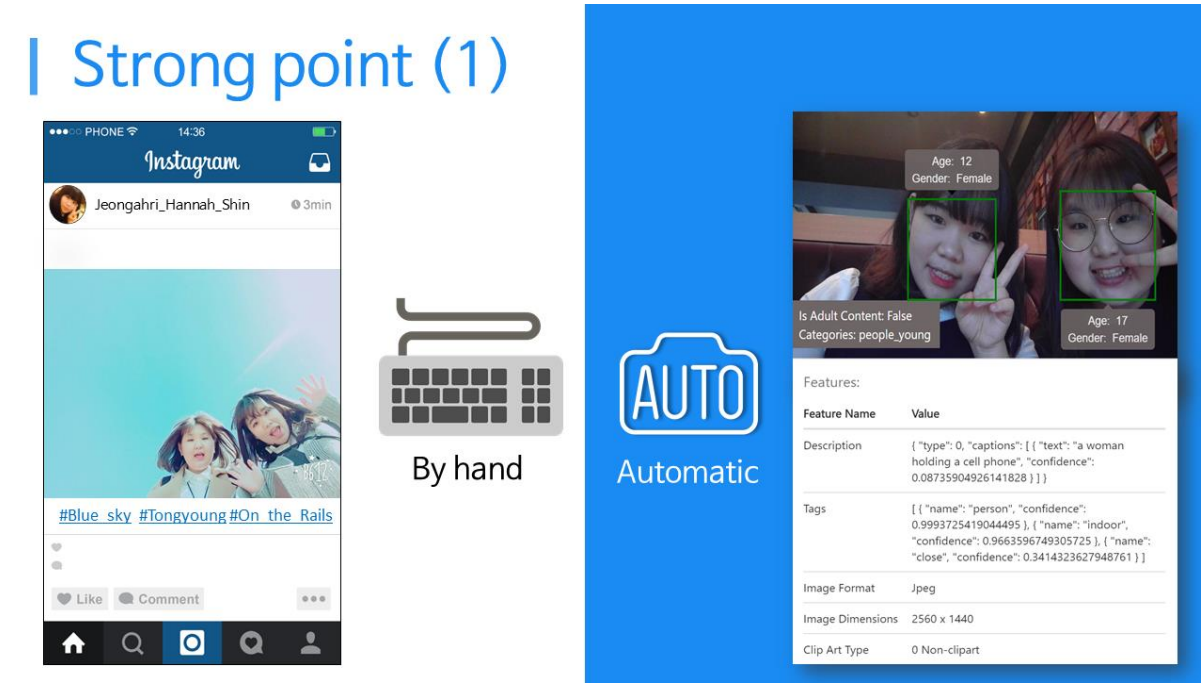
Xamarin Framework는 기존에 우리가 Android와 Ios 플랫폼에서 활용하던 API가 100% 호환 가능하도록 설계되었다. 과연 높은 호환성 보유 여부가 진실인지, 그 적용에 있어서 한계점은 없는지 알아보기 위하여 Microsoft사에서 최근 출시한 Cognitive API (인지 API)를 사용하여 실험해 보기로 하였다.

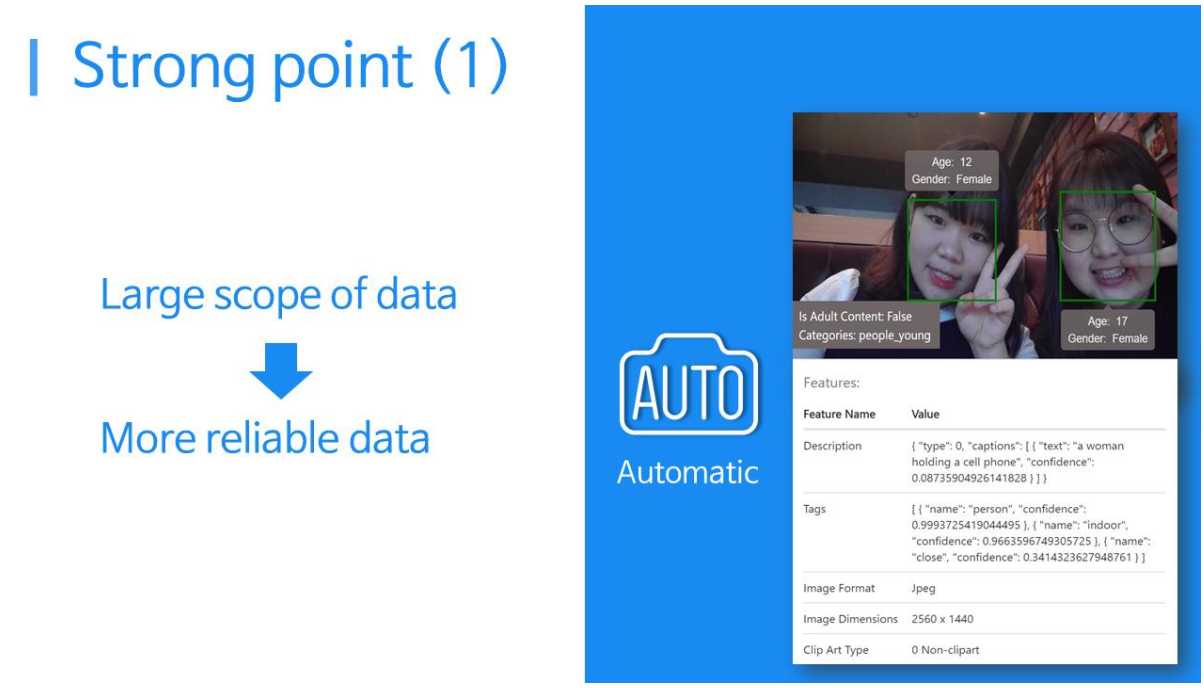
1) Cognitive API란?

(자료 출처; Hanyang FMS 연구 자료)



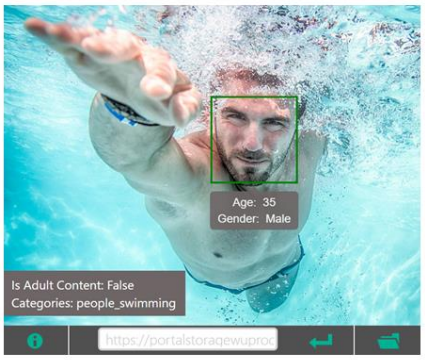
Cognitive API는 인공지능과 인간의 신호 처리를 기반으로 한 기술 플랫폼이다. 컴퓨터가 스스로의 학습을 통해 시스템 스스로가 어떤 사항에 대해 생각하고 직접 답변을 유도해내는 시스템을 말한다. 예를 들어, 시각적(visual) 인지는 디바이스의 카메라로 사진을 찍음으로써 사진 안에 있는 사물이나 사람 등을 인식할 수 있다.





다음은 Cognitive API의 종류 중, Computer Vision API를 사용하여 본인의 사진을 분석한 예제이다. 사진 속 요소의 묘사, 이미지 포맷 부터 사진 내 사람의 특성 까지 인지하여 JSON 방식의 태그 형태로 보여주고 있다.

Strong point (2)



Various data into categorized forms

Features:	
Feature Name	Value
Description	{ "type": 0, "captions": [{ "text": "a man swimming in a pool of water", "confidence": 0.7850108124440484 }] }
Tags	[{ "name": "water", "confidence": 0.9996442794799805 }, { "name": "sport", "confidence": 0.9504992365837097 }, { "name": "swimming", "confidence": 0.9062818288803101, "hint": "sport" }, { "name": "pool", "confidence": 0.8787588477134705 }, { "name": "water sport", "confidence": 0.631849467754364, "hint": "sport" }]
Image Format	Jpeg
Image Dimensions	1500 x 1155
Clip Art Type	0 Non-clipart
Line Drawing Type	0 Non-LineDrawing
Black & White Image	False
Is Adult Content	False
Adult Score	0.14750830829143524
Is Racy Content	False
Racy Score	0.12601403892040253
Categories	[{ "name": "people_swimming", "score": 0.98046875 }]
Faces	[{ "age": 35, "gender": "Male", "faceRectangle": { "width": 312, "height": 312, "left": 745, "top": 338 } }]
Accent Color	#19A4B2

다음은 Cognitive Computer Vision API를 통해 분석할 수 있는 Feature들을 나열한 것이다.

Strong point (3)

Global language

Vision	Speech	Language	Knowledge	Search
Computer Vision	Bing Speech	Bing Spell Check	Academic	Bing Autosuggest
Content Moderator	Custom Recognition	Language Understanding	Entity Linking	Bing Image Search
Emotion	Speaker Recognition	Linguistic Analysis	Knowledge Exploration	Bing News Search
Face		Text Analytics	Recommendations	Bing Video Search
Video		Translator		Bing Web Search
		WebLM		

Low input cost → getting information from all over the world

다음과 같이 Cognitive API는 크게 Vision, Speech, Language, Knowledge, Search 등 사람만이 가지고 있는 인지적 특성과 감각을 소프트웨어화 하였다. 이 중에서도 이번 회차에서는 Vision API를 Xamarin Framework에 적용해 볼 것이다.

b. Cognitive API - Xamarin에의 적용

Xamarin Framework에 Vision API를 적용하기 위한 과정은 다음과 같다. 먼저 시각적으로 분석하기 위한 사진을 가져오기 위해 Media Plugin for Xamarin and Windows를 Visual Studio 2015 개발환경 상에서 설치해준다. 주변 사물의 이름을 입력한 후에, 그 사물의 사진을 찍어 Vision API가 돌려주는 JSON 태그를 통해 사물이 제대로 인식되고 있는지를 확인한다.

다음은 Xamarin 위에서 카메라를 작동시켜서 사진을 업로드 하기 위한 C# 코드 예제이다.

```
var imagePicker = new UIImagePickerController();
imagePicker.SourceType = UIImagePickerControllerSourceType.Camera;
PresentViewController(imagePicker, true, null);
imagePicker.Canceled += async delegate {
    await imagePicker.DismissViewControllerAsync(true);
};

imagePicker.FinishedPickingMedia += async (object s, UIImagePickerMediaPickedEventArgs e) {
    //Insert code here for upload to Cognitive Services
};
```

업로드한 사진을 API를 사용하여 분석하기 위해서 분석 명령을 시동하는 코드를 작성한다.

```
public async Task GetImageDescription(Stream imageStream)
{
    VisualFeature[] features = { VisualFeature.Tags, VisualFeature.Categories, Visu
alFeature.Description};
    return await visionClient.AnalyzeImageAsync(imageStream, features.ToList(), nul
l);
}
```

사진의 분석 결과가 먼저 입력한 사물의 이름과 일치하는지 확인하기 위해 다음과 같은 코드를 작성한다. 변수 selectWord는 우리가 처음에 입력했던 사물의 이름이므로 default처리한다. 또한 반복/제어문을 통해 분석 결과로 받아온 JSON 태그 요소가 selectWord와 일치하는지 검증한다. 한국어 지원이 안되므로 분석 언어는 영어로 설정하고, 소문자로만 인식하도록 처리해준다.

```
var selectedWord = Words.FirstOrDefault();
foreach (var tag in result.Description.Tags)
{
    if (tag == selectedWord.English.ToLower())
    {
        Acr.UserDialogs.UserDialogs.Instance.ShowSuccess("Correct!");
        Words.Remove(selectedWord);
        if (Words.Count > 0)
           lblWord.Text = Words.FirstOrDefault().Translation;
        else
        {
            lblWord.Text = "Game Over";
            btnSkip.Alpha = 0;
            btnSnapPhoto.Alpha = 0;
        }
        return;
    }
 }
```

c. API 적용 결과

신 Framework임에도 불구하고 API 적용과 호환성 면에서는 훌륭한 결과를 보였다. 하지만, 플랫폼 특성 별로 UI의 구조가 조금씩 상이하므로 Image Formatting에 대한 면에서는 별도로 설정해줄 필요성이 있었다.