



안드로이드 프로그래밍 기초  
(ver3.3.9)

# 목차 (Table of Contents)

---

1. 안드로이드 소개
2. 안드로이드 애플리케이션 구조
3. 독립 구동형 애플리케이션 개발
4. 클라우드 기반 애플리케이션 개발
5. 안드로이드 활용



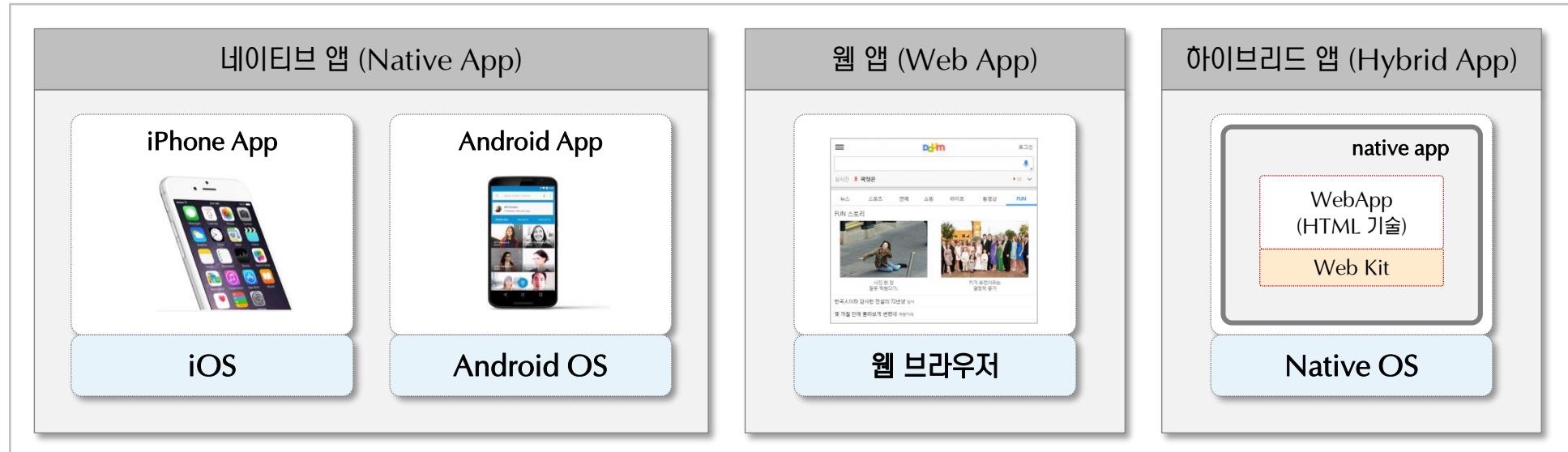
# 1. 안드로이드 소개

---

- 1.1 모바일 애플리케이션
- 1.2 안드로이드 개요
- 1.3 안드로이드 버전
- 1.4 안드로이드 아키텍처
- 1.5 안드로이드 개발 환경
- 1.6 안드로이드 빌드 과정
- 1.7 요약

# 1.1 모바일 애플리케이션

- ✓ 오늘날, 모바일 애플리케이션 시장은 iOS, Android 등 다양한 플랫폼 기반으로 빠르게 발전하고 있습니다.
- ✓ 모바일 애플리케이션은 개발 방식에 따라 Native App과 Web App으로 분류합니다.
- ✓ Native App은 iOS나 Android와 같은 특정 플랫폼 API를 사용하여 개발한 모바일 애플리케이션입니다.
- ✓ Web App은 HTML(또는 HTML5) 기반으로 개발하여 멀티 플랫폼을 지원하는 모바일 애플리케이션입니다.



## 1.2 안드로이드 개요

- ✓ 안드로이드는 구글에서 제공하는 리눅스 커널 기반의 모바일 기기 및 휴대용 장치를 위한 오픈소스 운영체제입니다.
- ✓ 안드로이드는 운영체제 뿐만 아니라 미들웨어, UI 컴포넌트, 표준 앱(달력, 메일, 지도 등)을 포함합니다.
- ✓ 안드로이드 앱은 Java 기반으로 작성하며, 모바일에 특화된 달vik 가상머신 위에서 동작합니다.
- ✓ 구글은 앱 개발을 위한 SDK 및 이클립스 ADT 플러그인, Android Studio와 같은 통합개발환경을 제공합니다.



# 1.3 안드로이드 버전 (1/3) – 개요

- ✓ 구글은 2005년 안드로이드를 인수하여, 2007년 처음으로 SDK를 출시하였습니다.
- ✓ 1.0 버전인 애플파이 부터 현재 최신버전인 6.0 마시멜로에 이르기 까지 매우 빠른 속도로 발전하고 있습니다.
- ✓ 재미있게도 구글은 안드로이드 버전을 나타내는 코드명으로 알파벳 순서의 디저트 이름을 사용합니다.
- ✓ 안드로이드의 발전과 더불어 많은 제조사들은 오픈 소스인 안드로이드를 사용하여 많은 단말기들을 출시하고 있습니다.



## 1.3 안드로이드 버전 [2/3] – 안드로이드 버전과 API 레벨

- ✓ API 레벨은 안드로이드 플랫폼 버전에 따라 부여된 번호를 뜻합니다.
- ✓ 안드로이드의 개발 플랫폼 버전이 결정되면 개발 애플리케이션에 API 레벨을 부여하여 버전을 결정합니다.
- ✓ 안드로이드는 SDK, NDK, PDK 세 종류의 개발 형태가 있습니다.
  - SDK(Software Development Kit)  
: Java API를 통한 기본 애플리케이션 개발.
  - NDK(Native Development Kit)  
: C/C++ 기반의 애플리케이션 개발.
  - PDK(Platform Development Kit)  
: 유닉스 계열의 운영체제에서 개발. SDK, NDK를 포함하는 개발 방법.

버전	API 레벨
Android 4.0/4.0.1/4.0.2	14
Android 4.0.3/4.0.4	15
Android 4.1	16
Android 4.2	17
Android 4.3	18
Android 4.4	19
Android 4.4W(Smart Watch)	20
Android 5.0/5.0.1/5.0.2	21
Android 5.1/5.1.1	22
Android 6.0/6.0.1	23

# 1.3 안드로이드 버전 (3/3) – 플랫폼 버전 별 사용 비율

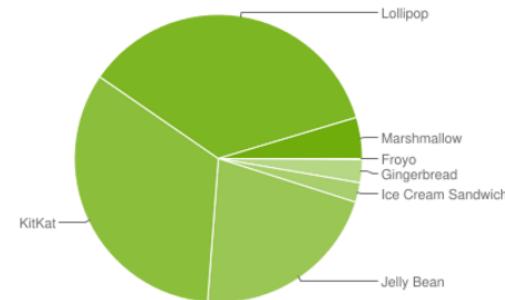
- ✓ 다음 표는 안드로이드 플랫폼 버전 별 사용 비율을 나타냅니다. (2016년 4월 기준)
- ✓ 안드로이드 애플리케이션을 개발할 때, 지원할 API 버전을 선정하는 것은 사용자 수를 결정하는 것을 의미합니다.
- ✓ 많은 사용자를 확보하려면 API 버전을 낮게 설정하면 되겠지만, 상위 버전의 최신 기능을 사용할 수 없는 단점이 있습니다.
- ✓ 애플리케이션을 개발할 때에는 사용할 플랫폼 기능과 지원해야 할 사용자 비율을 고려하여 API 버전을 선정합니다.

## Platform Versions

This section provides data about the relative number of devices running a given version of the Android platform.

For information about how to target your application to devices based on platform version, read [Supporting Different Platform Versions](#).

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.2%
4.1.x	Jelly Bean	16	7.8%
4.2.x		17	10.5%
4.3		18	3.0%
4.4	KitKat	19	33.4%
5.0	Lollipop	21	16.4%
5.1		22	19.4%
6.0	Marshmallow	23	4.6%



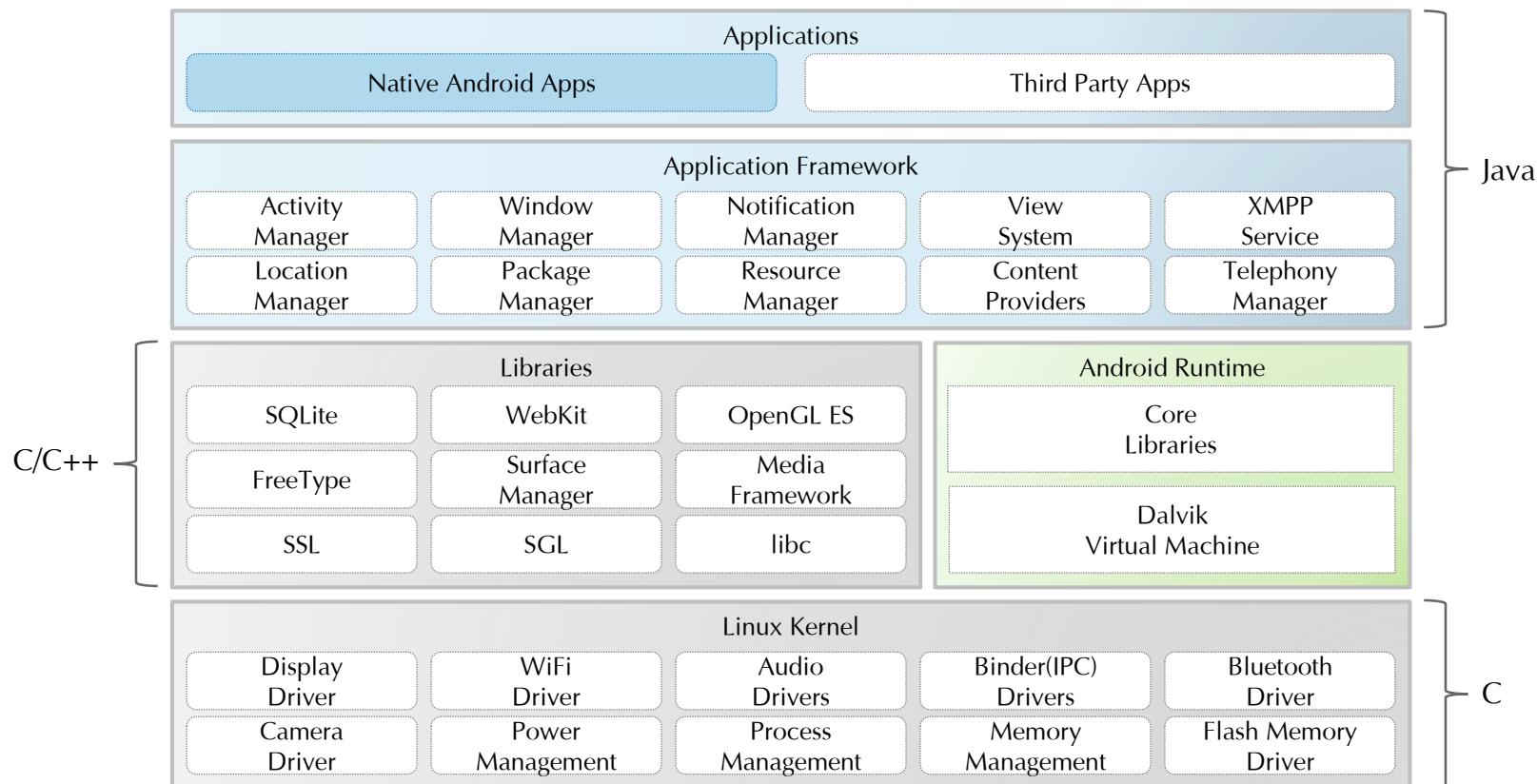
→ 97.3%

2016년 4월 기준

출처 : <http://developer.android.com/about/dashboards/index.html>

# 1.4 안드로이드 아키텍처 [1/6]-개요

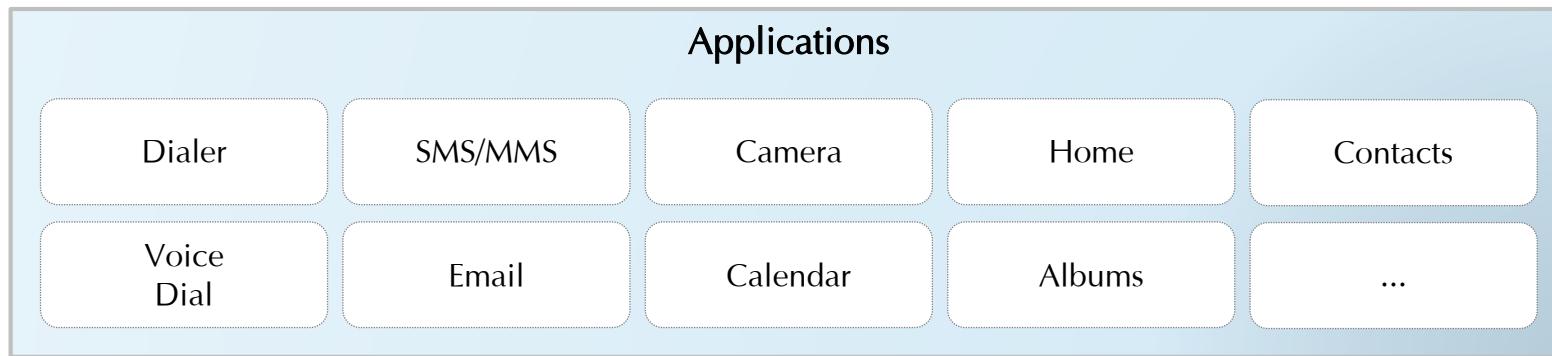
- ✓ 안드로이드의 아키텍처는 애플리케이션, 애플리케이션 프레임워크, 라이브러리, 안드로이드 런타임, 리눅스 커널 계층으로 나뉘어 있습니다.(하드웨어 추상 계층 제외)
- ✓ 구글에서는 안드로이드에 대해 다음과 같이 정의하고 있습니다.
  - “안드로이드는 운영체제와 미들웨어 그리고 핵심 애플리케이션을 포함하는 소프트웨어 스택이다.”



출처 : [https://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)](https://en.wikipedia.org/wiki/Android_(operating_system))

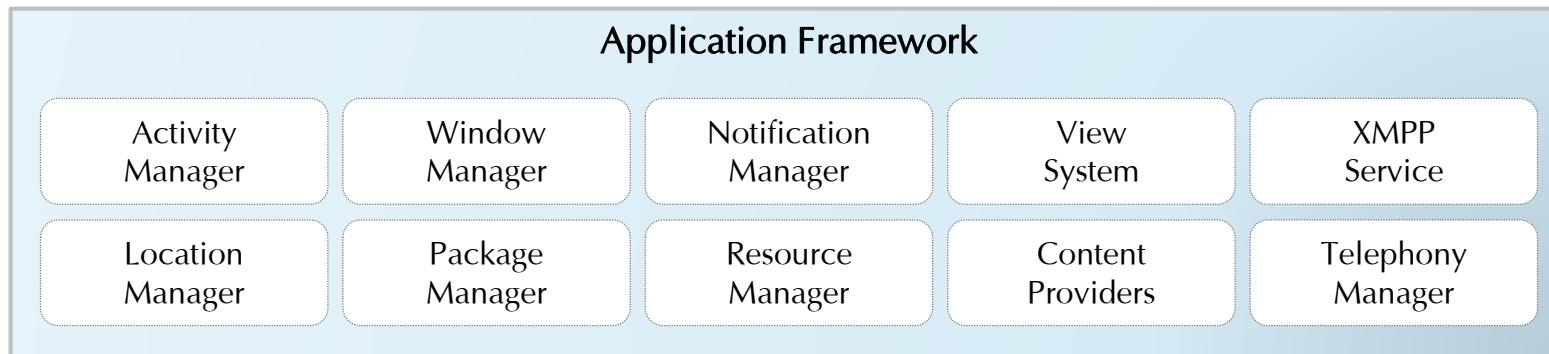
## 1.4 안드로이드 아키텍처 [2/6]–애플리케이션 계층

- ✓ 기본적인 애플리케이션들이 위치하는 계층이 Applications 계층입니다.
- ✓ Applications 계층의 애플리케이션들은 Java 언어로 개발합니다.
- ✓ 안드로이드는 기본적으로 내장되는 핵심 애플리케이션과 서드파티 애플리케이션이 동등한 권한을 갖습니다.
- ✓ 안드로이드의 API 버전에 따라 포함하고 있는 핵심 애플리케이션은 다를 수 있습니다.



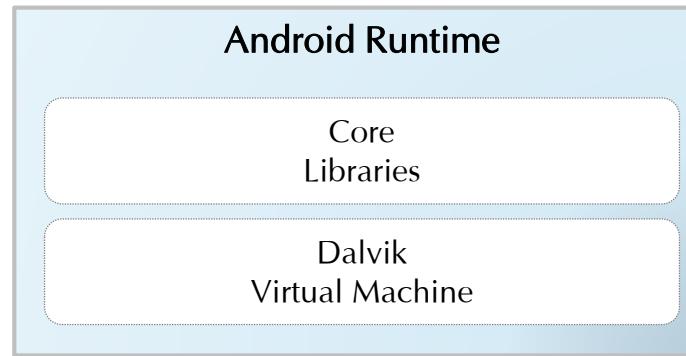
## 1.4 안드로이드 아키텍처 [3/6] – 애플리케이션 프레임워크 계층

- ✓ 애플리케이션 프레임워크 계층은 JNI(Java Native Interface)를 이용해 C/C++ 코드로 작성되어 있습니다.
- ✓ Core System service와 Hardware service로 구분되며 그 차이는 다음과 같습니다.
  - Core System services : Activity, View, Window, Package, ContentProvider의 관리
  - Hardware System services : Telepony, Location, WiFi, USB, 그리고 각종 센서들의 관리



## 1.4 안드로이드 아키텍처 [4/6] – 런타임 계층

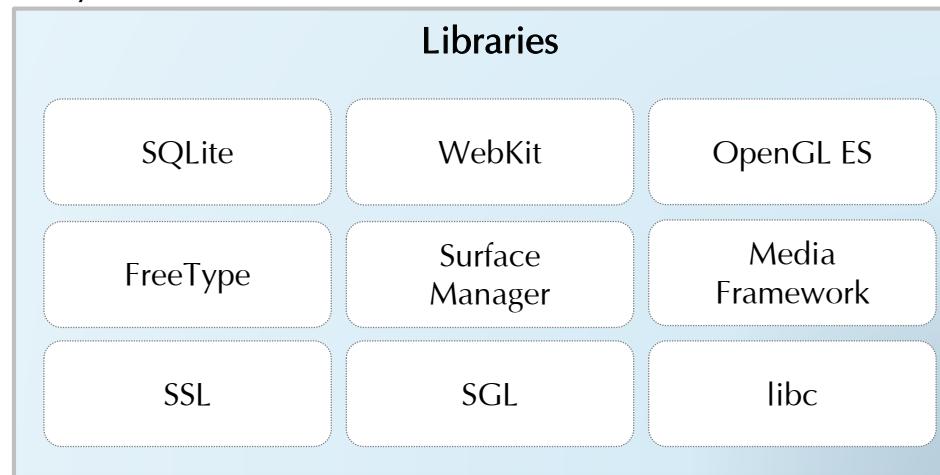
- ✓ Core Libraries는 안드로이드에서 Java의 핵심 라이브러리를 사용할 수 있도록 하는 Java 라이브러리 집합입니다.
- ✓ Dalvik Virtual Machine은 복수의 인스턴스 실행에 최적화된 레지스터 기반의 가상머신입니다.
  - Dalvik은 JVM과 달리 스레딩(Threading)과 메모리 관리와 같은 역할은 리눅스 커널에 의존합니다.
  - 애플리케이션이 동작하는 모든 프로세스에 Dalvik을 탑재합니다.



## 1.4 안드로이드 아키텍처 [5/6] – 라이브러리 계층

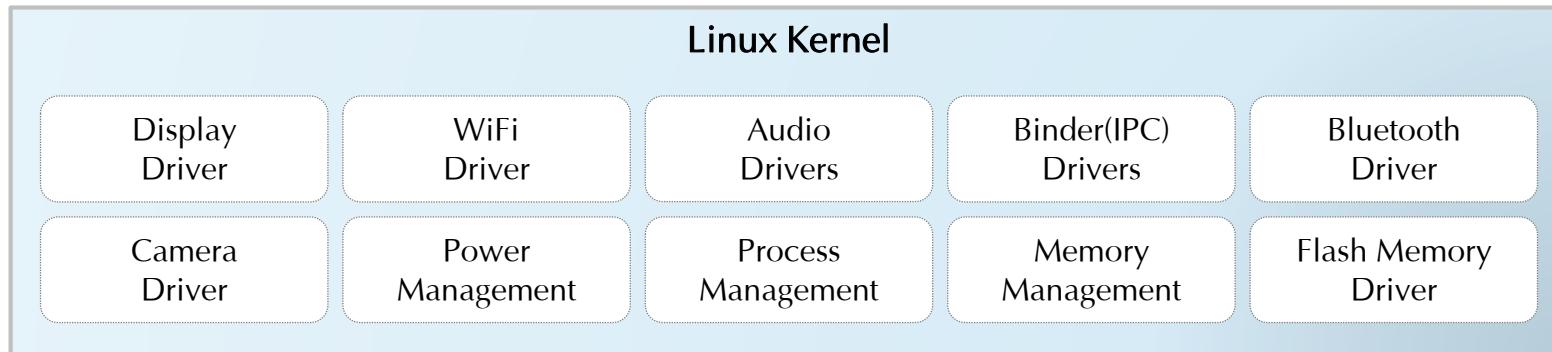
✓ Libraries 계층에는 다양한 코어 라이브러리들을 가지고 있으며 다음과 같이 포함하고 있습니다.

- 미디어 라이브러리 : 오디오 및 비디오 재생
- 서피스 매니저(Surface Manager) : 디스플레이 관리
- SQLite : 데이터베이스 지원
- WebKit과 SSL(Secure Socket Layer): 통합 웹 브라우저 엔진과 인터넷 보안



## 1.4 안드로이드 아키텍처 [6/6] – 리눅스 커널

- ✓ 리눅스 커널에서는 백그라운드 스레드와, 메모리, 보안, 드라이버 등을 담당합니다.
- ✓ 안드로이드에서 추가된 기능은 다음과 같습니다.
  - 바인더: 프로세스간의 통신
  - 전원 관리, 메모리 관리(Low Memory Killer)



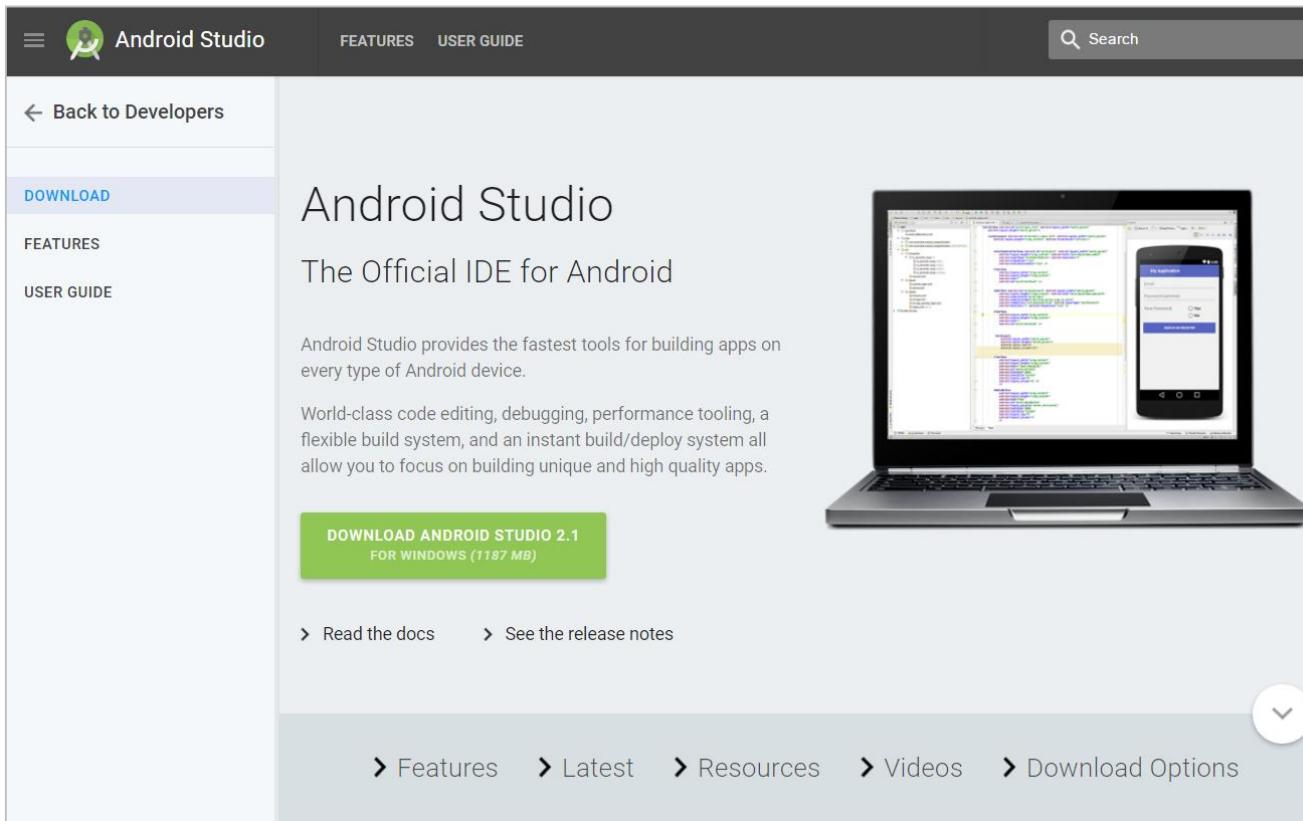
# 1.5 안드로이드 개발환경 – 개요

- ✓ 안드로이드 애플리케이션의 개발환경은 JDK, 안드로이드 SDK, 안드로이드 개발 IDE 를 설치하여 구성합니다.
- ✓ 안드로이드 애플리케이션은 Java 언어를 사용하여 개발합니다. 따라서, Java 개발을 위해 JDK를 설치합니다.
- ✓ 안드로이드 SDK는 애플리케이션 개발에 필요한 플랫폼과 안드로이드 프레임워크 API를 제공합니다.
- ✓ 안드로이드 개발 IDE는 이클립스 IDE에 ADT 플러그인 설치하거나, 공식 IDE인 Android Studio 를 사용합니다.



# 1.5 안드로이드 개발환경 – 안드로이드 스튜디오[1/2]

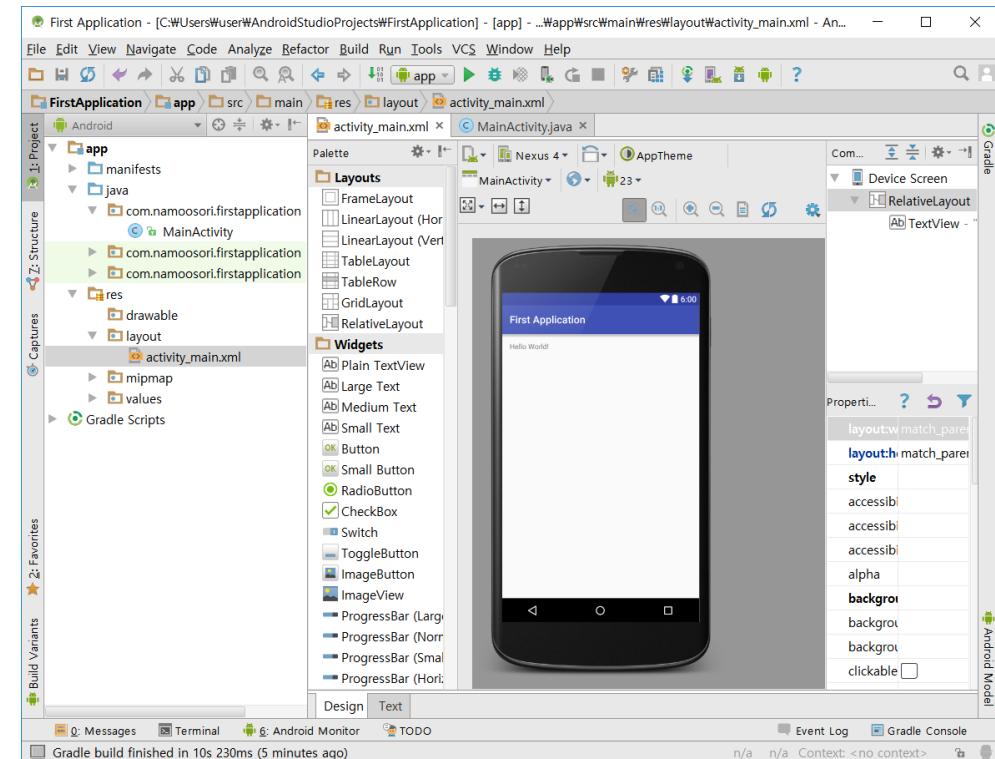
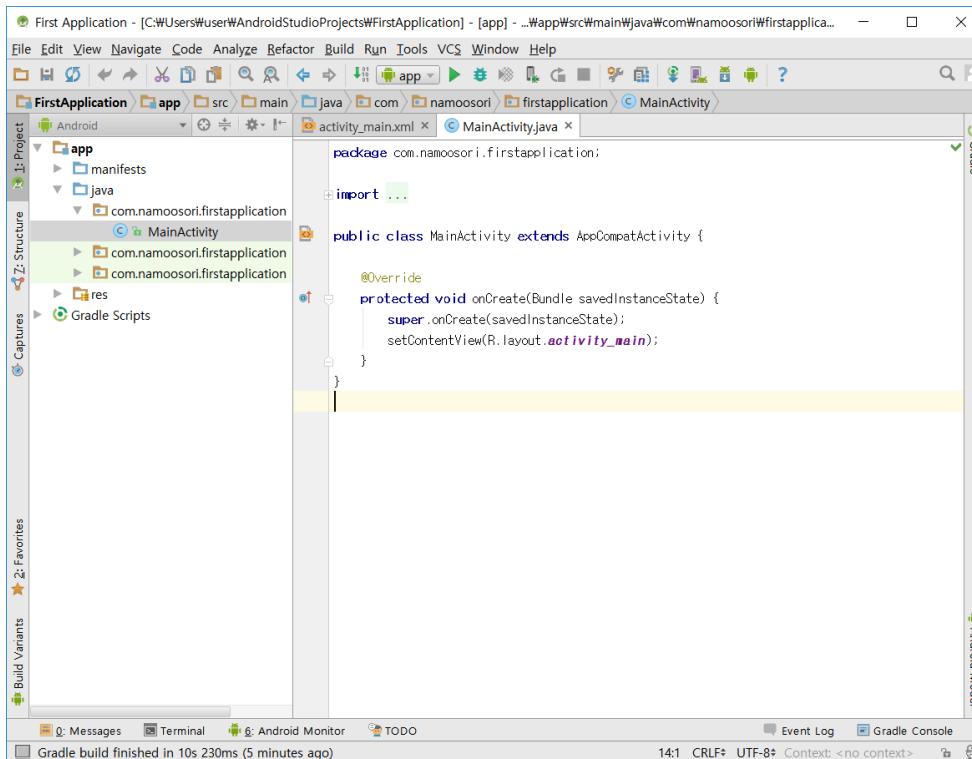
- ✓ 안드로이드 스튜디오는 2013년 5월 Google I/O를 통해 발표된 안드로이드 통합 개발 환경(IDE)입니다.
- ✓ IntelliJ 기반의 개발 환경으로 Apache License 2.0의 라이선스를 갖고 있습니다.
- ✓ 안드로이드 스튜디오의 빌드 시스템은 Gradle을 채택하고 있습니다.
- ✓ 안드로이드 스튜디오는 하나의 실행 파일에 모든 도구를 담고 있기 때문에 설치가 쉽습니다.



출처 : <https://developer.android.com/studio/index.html>

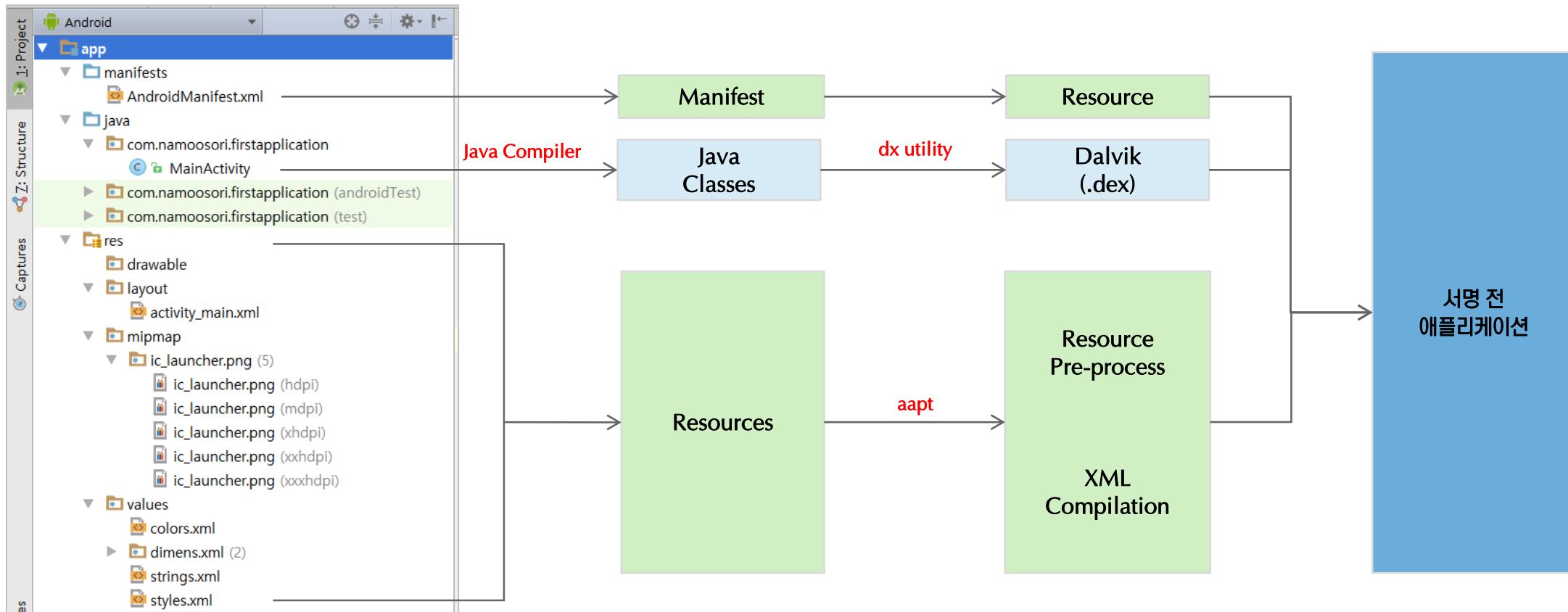
# 1.5 안드로이드 개발환경 – 안드로이드 스튜디오[2/2]

- ✓ 이클립스와 달리 안드로이드 스튜디오는 프로젝트마다 별도의 도구 창(window)을 갖습니다.
- ✓ 안드로이드 스튜디오는 해당 개발 시점에 사용 가능한 도구나 창들의 일부만을 화면에 나타냅니다.
- ✓ 안드로이드 스튜디오는 기본 뷰는 Android 뷰이며 주로 사용하는 딕레토리와 파일을 보여 줍니다.
- ✓ Android 뷰 외에는 Project 뷰를 자주 사용합니다.



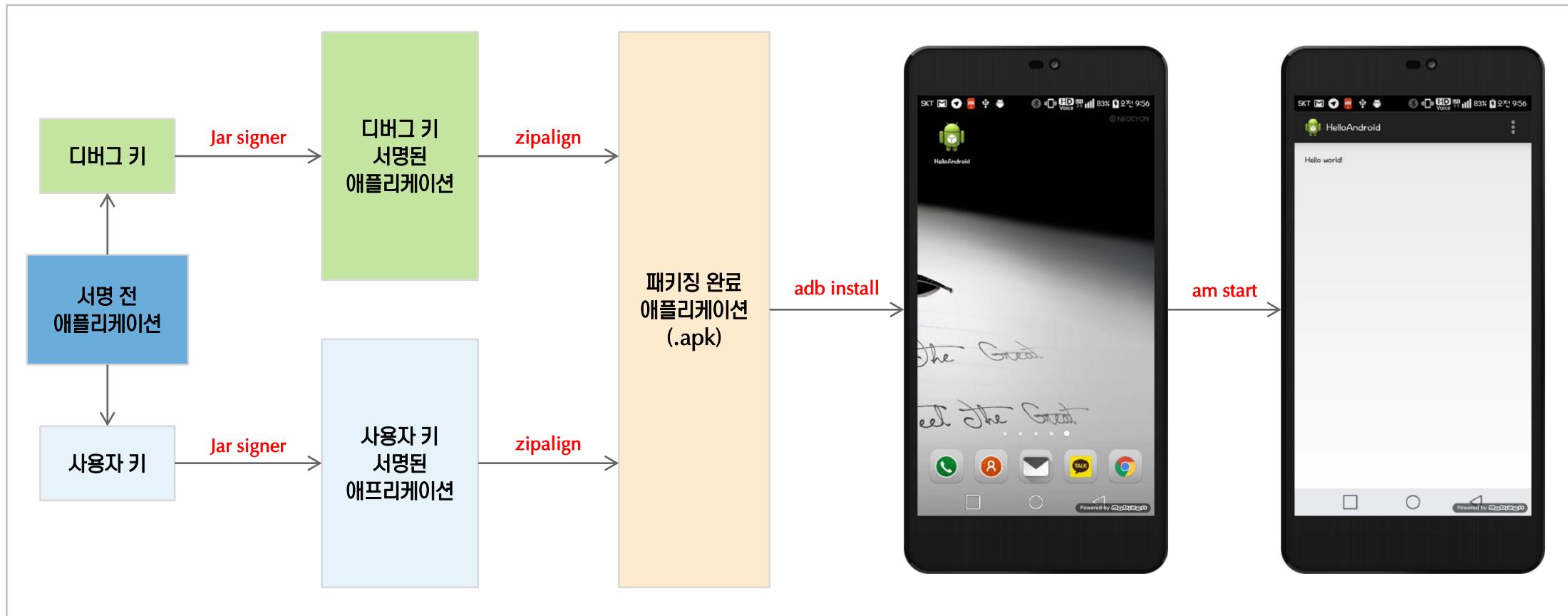
# 1.6 안드로이드 빌드 과정 (1/2)

- ✓ 안드로이드는 애플리케이션 작성부터 설치 및 실행까지 다양한 개발 도구를 사용하여 개발합니다.
- ✓ Java로 작성한 소스 코드는 javac.exe 컴파일러를 통해 class 파일로 변환되며, 다시 dex 도구를 통해 .dex (Dalvik EXecutable)파일로 변환합니다.
- ✓ 이미지와 같은 리소스 파일들은 aapt(android asset packing tool)에 의해 전처리 과정을 거칩니다.



## 1.6 안드로이드 빌드 과정 (2/2)

- ✓ Google Play에서 애플리케이션을 배포하려면 개발자 고유의 키(Custom key)를 이용하여 빌드하여야 합니다.
- ✓ Custom key를 사용하지 않은 애플리케이션은 기본적으로 디버그 키(Debug key)를 이용하여 빌드합니다.
- ✓ 완성된 애플리케이션(.apk)은 adb.exe(android debug bridge)를 이용하여 디바이스에 설치합니다.



## 1.7 요약

---

- ✓ 모바일 애플리케이션은 개발 방식에 따라 Native App, Web App, Hybrid App으로 분류합니다.
- ✓ 안드로이드는 모바일 운영체제를 포함한 미들웨어, UI 컴포넌트, 표준 앱 등을 포괄하는 모바일 플랫폼입니다.
- ✓ 안드로이드 앱은 Java 언어를 사용하여 개발하며, Dalvik 가상머신(VM)에서 동작합니다.
- ✓ 안드로이드의 구조는 앱, 앱 프레임워크, 라이브러리, 안드로이드 런타임, 리눅스 커널로 나눌 수 있습니다.
- ✓ 구글은 2005년에 안드로이드를 인수한 이래, 2016년 현재 안드로이드 버전 6.0(마시멜로)을 출시하였습니다.
- ✓ 안드로이드 개발환경은 이클립스 IDE를 사용하거나 공식 IDE인 Android Studio를 사용합니다.
- ✓ 안드로이드는 애플리케이션 작성부터 설치 및 실행까지 다양한 개발 도구들을 사용합니다.

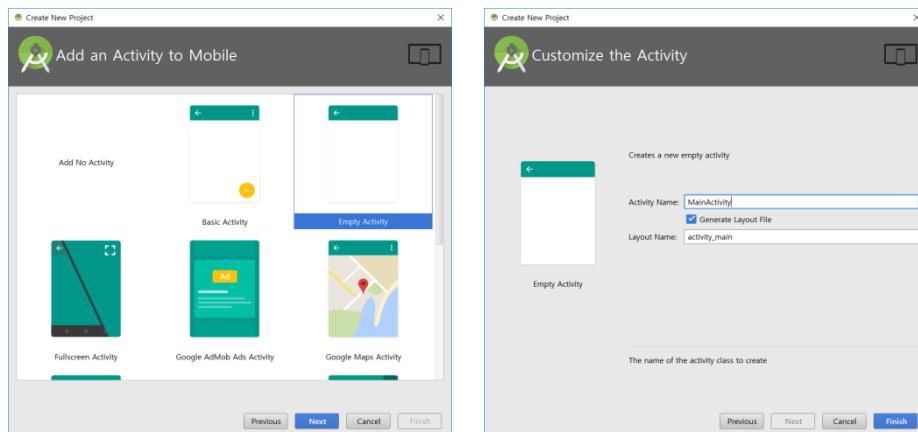
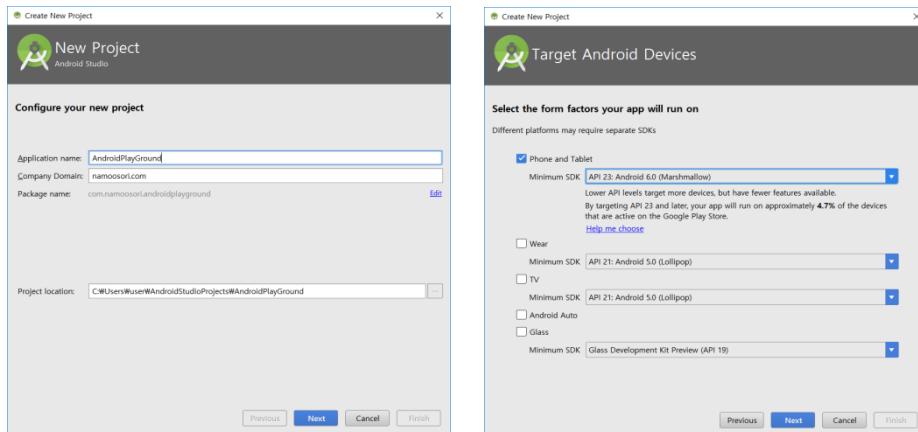


## 2. 안드로이드 애플리케이션 구조

- 
- 2.1 프로젝트 구조
  - 2.2 애플리케이션 컴포넌트와 주요 클래스
  - 2.3 AndroidManifest.xml
  - 2.4 안드로이드 애플리케이션 실행 절차
  - 2.5 안드로이드 UI
  - 2.6 리소스
  - 2.7 요약

## 2.1 프로젝트 구조 (1/2) – 기본정보

- ✓ 안드로이드 프로젝트를 생성할 때, 프로젝트에 대한 기본정보를 다음과 같이 입력합니다.
- ✓ Application Name과 Project Name은 AndroidPlayGround로 입력합니다.
- ✓ Company Domain은 namosori.com으로 입력합니다. (패키지 명으로 사용합니다.)
- ✓ 처음 생성될 화면(Activity)의 형태를 선택하고 이름과 레이아웃 명을 지정하면 첫 새로운 프로젝트를 생성합니다.



항목명	설명
Application Name	앱 사용자에게 보여 주는 이름
Company Domain	회사나 개인의 도메인을 입력하면 페이지명으로 지정됨
Package Name	안드로이드 시스템이 앱을 구분하기 위한 이름
Minimum Required SDK	하위 호환성을 위해 앱이 지원하는 최소 단말기 버전
Add an Activity to mobile	첫 화면의 기본 스타일을 선택
Activity Name	처음 생성되는 Activity의 이름 지정
Layout Name	처음 생성되는 레이아웃 파일의 이름 지정

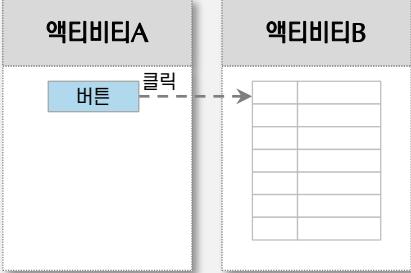
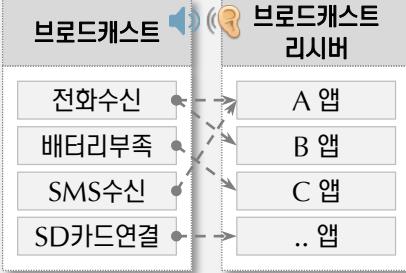
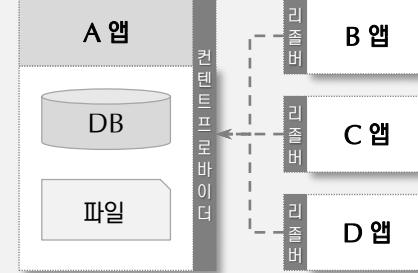
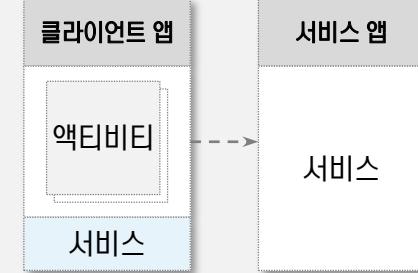
## 2.1 프로젝트 구조 (2/2) – 프로젝트 구성요소

- ✓ 안드로이드 스튜디오의 프로젝트 도구 창은 Project, Android, Packages, Tests 등 다양한 형태로 볼 수 있습니다.
- ✓ 기본적으로 Android 뷰로 설정되어 있으며 폴더의 구조는 다음과 같습니다.
- ✓ 크게 app과 Gradle Scripts로 구분되며 안드로이드 개발에 주로 사용되는 폴더들로 구성되어 있습니다.
- ✓ app 폴더에는 프로그램 개발에 꼭 필요한 파일들이 종류 별로 놓여 있습니다.



## 2.2 애플리케이션 컴포넌트와 주요 클래스 (1/2)

- ✓ 액티비티(Activity)는 가장 기본이 되는 컴포넌트로 사용자에게 보여주는 화면을 제공합니다.
- ✓ 브로드캐스트 리시버(Broadcast Receiver)는 장치나 앱에서 발생하는 이벤트를 수신하는 컴포넌트입니다.
- ✓ 컨텐트 프로바이더(Content Provider)는 앱 내 데이터를 외부 앱에 공개하려는 목적으로 사용하는 컴포넌트입니다.
- ✓ 서비스(Service)는 화면이 없으며 백그라운드에서 특정 작업을 수행하는 컴포넌트입니다.

액티비티 (Activity)	브로드캐스트 리시버 (Broadcast Receiver)	컨텐트 프로바이더 (Content Provider)	서비스 (Service)
			
<ul style="list-style-type: none"><li>✓ UI를 구성하는 기본 단위</li><li>✓ UI 제공 및 이벤트 처리</li><li>✓ 액티비티는 하나의 화면과 대응</li></ul>	<ul style="list-style-type: none"><li>✓ 다른 앱이나 안드로이드 OS로부터 보내오는 인텐트 형태의 데이터 수신</li><li>✓ 신호를 받으면 적절한 액티비티나 서비스를 띄우는 역할을 함</li></ul>	<ul style="list-style-type: none"><li>✓ 앱 내 데이터는 다른 앱에서 직접 접근할 수 없음</li><li>✓ 컨텐트 프로바이더는 앱 간 데이터를 공유하기 위한 방법</li></ul>	<ul style="list-style-type: none"><li>✓ 화면을 갖지 않는 백그라운드 처리로 네트워크 감시나 백그라운드 처리에 활용</li><li>✓ UI가 없으므로 사용자의 명령을 받아들일 수 있는 액티비티와 연결해서 사용함</li></ul>

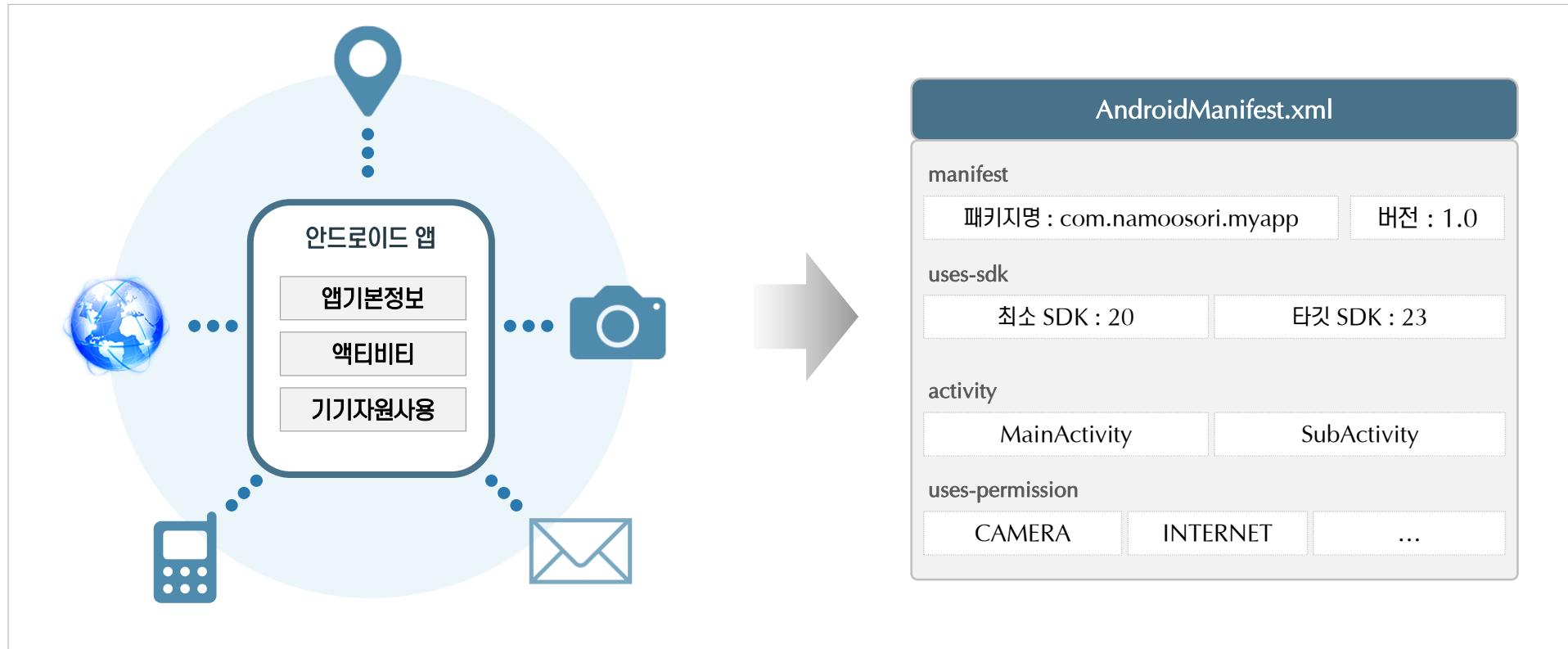
## 2.2 애플리케이션 컴포넌트와 주요 클래스 [2/2]

- ✓ 인텐트(Intent)는 안드로이드 애플리케이션을 구성하는 주요 컴포넌트는 아니지만 각 컴포넌트들을 호출하고 데이터를 전달하는 역할을 담당합니다.
- ✓ 프래그먼트(Fragment)는 Android 3.0(API Level-11) 버전부터 사용되는 클래스로 태블릿과 같이 큰 스크린에서 UI를 구성할 목적으로 만들었으며, 현재는 해상도에 상관없이 모든 장치에서 사용 합니다.

인텐트 (Intent)	프래그먼트 (Fragment)
<p>The diagram illustrates the interaction of Intent. On the left, Activity A contains a button labeled '클릭' (Click). An Intent is triggered from Activity A, represented by a dashed arrow pointing to Activity B. Activity B contains a list view. Another Intent is triggered from Activity B, represented by a dashed arrow pointing to a Service. The Service is shown as a separate box labeled '서비스 앱' (Service App) with a '서비스' (Service) label below it.</p> <ul style="list-style-type: none"><li>✓ 인텐트는 컴포넌트가 다른 컴포넌트를 호출할 때 사용합니다.</li><li>✓ 인텐트에는 데이터를 담아 다른 컴포넌트로 전달할 수 있습니다.</li><li>✓ 인텐트로 호출되는 컴포넌트는 액티비티, 서비스, 브로드캐스트리시버입니다.</li><li>✓ 인텐트 호출 기법은 명시적인 방법과 묵시적인 방법이 있습니다.</li></ul>	<p>The diagram illustrates the interaction of Fragment. On the left, Activity A contains a fragment with a blue header bar. An Intent is triggered from Activity A, represented by a dashed arrow pointing to Activity B. Activity B contains a fragment with a blue header bar. Both activities have a list view below their respective fragments.</p> <ul style="list-style-type: none"><li>✓ 프래그먼트는 액티비티 내에서 화면을 구성하는 일부라 할 수 있습니다.</li><li>✓ 프래그먼트는 반드시 액티비티 내에 포함되어야 하며 해당 액티비티와 생명주기를 같이 합니다.</li><li>✓ 프래그먼트는 액티비티 수행중에 추가나 제거가 가능하며 여러 액티비티에서 재사용도 가능합니다.</li></ul>

## 2.3 AndroidManifest.xml (1/5) – 개요

- ✓ `AndroidManifest.xml` 은 매니페스트라고 부르며, 애플리케이션 구성의 가장 기본이 되는 파일입니다.
- ✓ 애플리케이션 패키지명, 버전, SDK 최소버전과 타깃 버전을 설정합니다.
- ✓ 애플리케이션 화면을 구성하는 액티비티들을 매니페스트에 등록합니다.
- ✓ 기기 자원을 사용하기 위해 사용 권한을 매니페스트에 등록합니다. (예, 인터넷 허용, GPS 허용 등)



## 2.3 AndroidManifest.xml (2/5) – manifest, uses-sdk 노드

- ✓ manifest 노드는 애플리케이션 기본정보인 패키지명, 버전정보를 관리합니다.
- ✓ uses-sdk 노드는 애플리케이션이 장치 위에서 올바르게 동작할 수 있도록 만족해야 하는 SDK 버전을 정의합니다.
- ✓ minSdkVersion 속성은 앱에서 지원하는 최소 SDK 버전을 관리합니다.
  - minSdkVersion 보다 낮은 안드로이드 OS가 설치된 경우 애플리케이션이 설치되지 않습니다.
- ✓ targetSdkVersion 속성은 애플리케이션 개발 시 사용하는 단말의 버전을 설정합니다.
  - maxSdkVersion 을 작성하지 않으면 minSdkVersion 을 따릅니다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.namoosori.playground"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="23"
        android:targetSdkVersion="23" />
```



안드로이드 스튜디오에서는 버전 코드, 버전 이름, sdk 버전을 AndroidManifest에서 설정하지 않고 Gradle의 build.gradle 파일에서 설정합니다.

```
defaultConfig {
    applicationId "com.namoosori.androidplayground"
    minSdkVersion 23
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"
}
```

## 2.3 AndroidManifest.xml (3/5) – application 노드

- ✓ 하나의 매니페스트는 application 노드를 단 하나만 가질 수 있습니다.
- ✓ application 노드에는 애플리케이션 컴포넌트를 지정하는데 사용합니다. (액티비티, 리시버, 프로바이더, 서비스)
- ✓ application 노드는 다양한 속성을 지원합니다. 설정 가능한 속성은 개발자 가이드 문서를 참고합니다.
- ✓ <https://developer.android.com/guide/topics/manifest/application-element.html?hl=ko>

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.namoosori.playground"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="23"
        android:targetSdkVersion="23" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
```

## 2.3 AndroidManifest.xml (4/5) – activity 노드

- ✓ 애플리케이션에서 화면에 표시하는 모든 액티비티는 activity 요소를 매니페스트에 등록해야 합니다.
- ✓ 액티비티 클래스명은 android:name 속성에 지정합니다.
- ✓ 매니페스트에 정의하지 않은 액티비티를 시작하려고 하면, 런타임 예외가 발생하며 애플리케이션을 중단합니다.
- ✓ 컨텐트 프로바이더 및 브로드캐스트 리시버, 서비스도 매니페스트에 등록해야 사용할 수 있습니다.

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name=".activity.SubActivity"
        android:label="@string/sub_title" />
```



안드로이드 런처는 메인 액티비티의 android:label 속성에 지정한 이름을 애플리케이션의 이름으로 사용합니다.

## 2.3 AndroidManifest.xml (5/5) – uses-permission 노드

- ✓ uses-permission 노드에는 애플리케이션이 올바로 동작하기 위해 필요한 단말자원의 기능 권한을 선언합니다.
- ✓ 사용자는 애플리케이션을 설치하는 시점에 설정된 기능 권한들을 확인하고 동의하는 경우에만 설치합니다.
- ✓ android:name 속성에 사용하고자 하는 자원의 권한이름을 등록합니다.
- ✓ 예를 들어, 카메라를 사용하려면 android.permission.CAMERA 를 android:name 속성에 설정합니다.

```
...
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ...
</application>
```

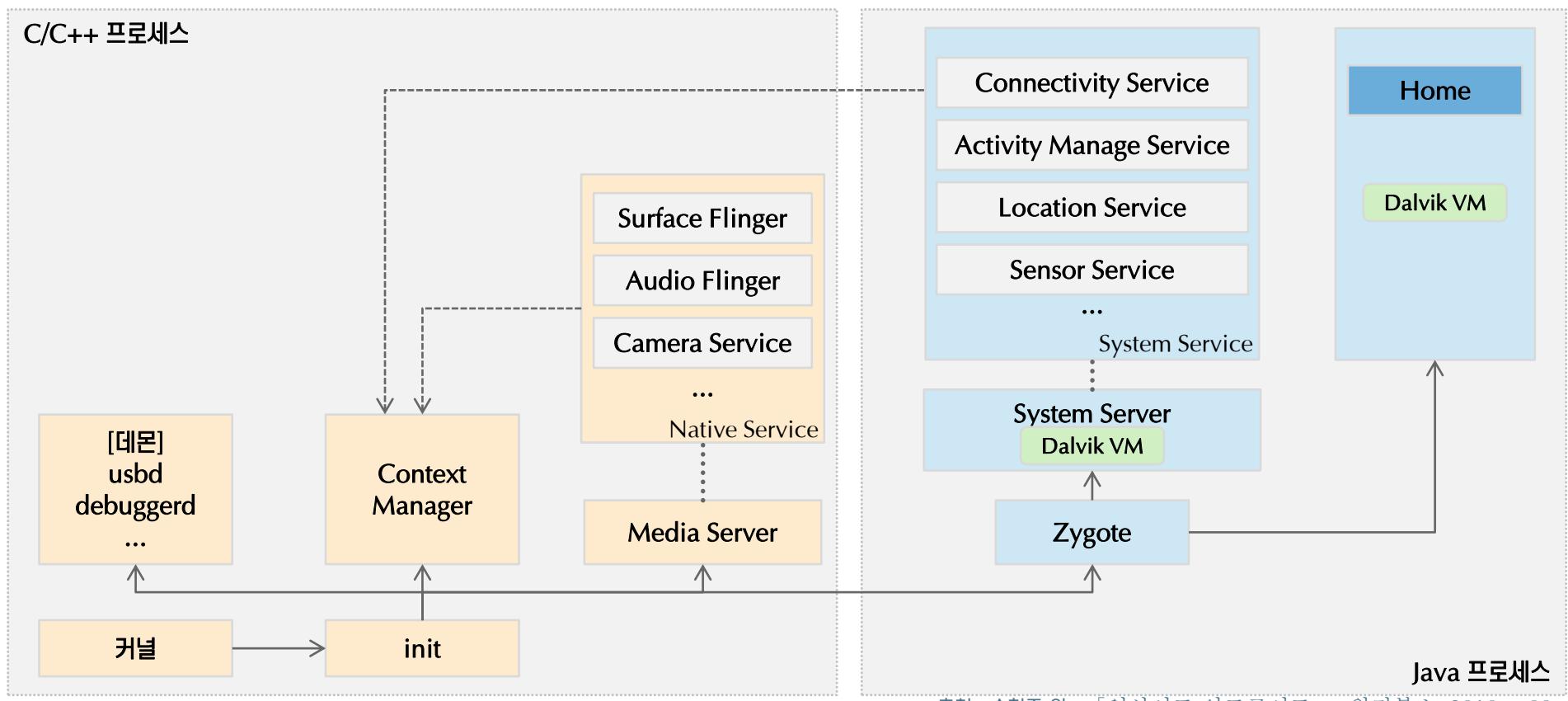


Android 6.0(마시멜로)는  
애플리케이션에 따라 별도로 권한 설  
정을 해야 합니다.

설명	권한
카메라	android.permission.CAMERA
진동	android.permission.VIBRATE
전화	android.permission.CALL_PHONE
인터넷	android.permission.INTERNET
외장저장장치	android.permission.WRITE_EXTERNAL_STORAGE
네트워크 상태	android.permission.ACCESS_NETWORK_STATE
휴대폰 번호정보	android.permission.READ_PHONE_STATE

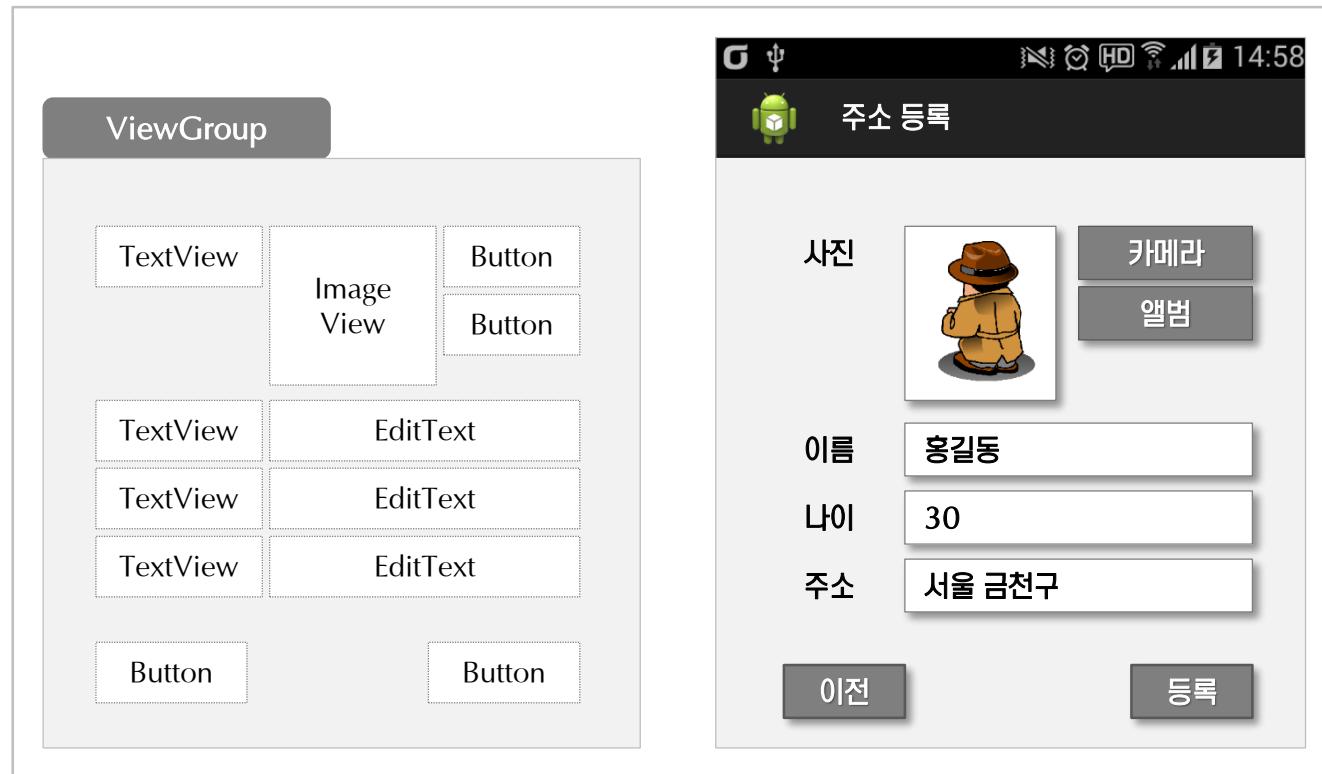
## 2.4 안드로이드 애플리케이션 실행 절차

- ✓ 안드로이드 디바이스가 실행되면 운영체제인 리눅스 커널이 수행되고 이후에 init 프로세스가 동작합니다.
- ✓ init 프로세스는 각종 디바이스들을 초기화하고 안드로이드의 동작을 위한 각종의 초기화 작업을 진행합니다.
- ✓ 컨텍스트 매니저(Context manager)는 시스템 서버가 제공하는 다양한 서비스들의 정보를 담고 있습니다.
- ✓ 시스템 서버(System server)는 다양한 애플리케이션들이 사용하는 중요 API를 제공합니다.



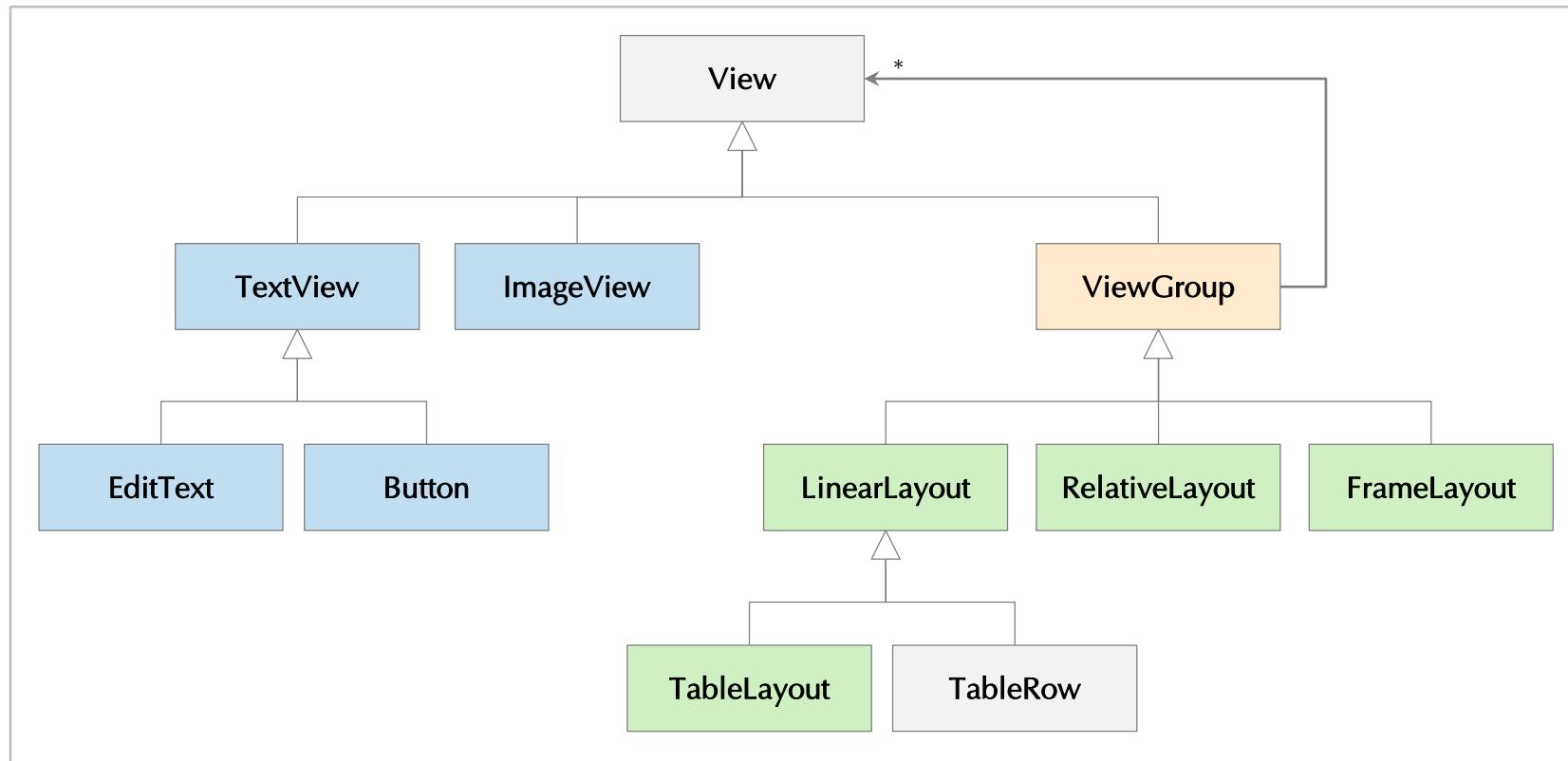
## 2.5 안드로이드 UI (1/9) – 개요

- ✓ 안드로이드 화면은 화면을 구성하는 최소 단위인 뷰(View)와 뷰들을 포함하는 뷰그룹(ViewGroup)으로 나뉩니다.
- ✓ 뷰는 각각의 기능을 갖고 있으며, 사용자의 이벤트에 반응하여 상호작용합니다. 뷰를 위젯(Widget)이라고도 합니다.
- ✓ 뷰그룹은 포함하고 있는 뷰들을 화면에 배치하기 위해 사용합니다. 뷰 그룹을 뷰 컨테이너(Container)라고도 합니다.
- ✓ 하나의 사용자 화면은 여러 모습과 기능을 가진 뷰들과 각기 다른 배치 방식을 갖고 있는 뷰 그룹을 통해 표현합니다.



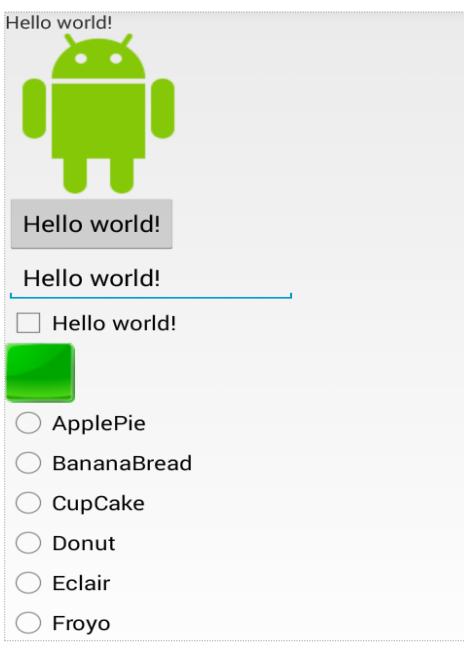
## 2.5 안드로이드 UI (2/9) – 뷰와 뷰그룹

- ✓ 안드로이드에서 화면에 보여지는 모든 구성요소들은 View 클래스를 상속합니다.
- ✓ 뷰에는 텍스트 출력 뷰(TextView), 텍스트 입력 뷰(EditText), 이미지 출력 뷰(ImageView) 등이 있습니다.
- ✓ ViewGroup에는 배치 방식에 따라 LinearLayout, RelativeLayout, FrameLayout, TableLayout 이 있습니다.
- ✓ 뷰그룹은 View 클래스를 상속하고 있으며, 뷰 그룹 안에 뷰 그룹을 배치할 수도 있습니다.



## 2.5 안드로이드 UI (3/9) – 대표적인 뷰(View)

- ✓ 안드로이드는 정보를 표현하기 위한 뷰 이외에 다양한 입력을 받을 수 있는 뷰들이 존재합니다.
- ✓ TextView 는 화면에 텍스트를 출력하고, EditText 는 사용자로부터 텍스트를 입력 받습니다.
- ✓ ImageView는 화면에 이미지를 출력하고, ImageButton은 텍스트가 아닌 이미지로 버튼을 표시합니다.
- ✓ 이 외에도 Button, RadioButton, CheckBox, Switch 뷰 등 다양한 UI 관련 뷰가 있습니다.



TextView	화면에 텍스트를 출력하는 뷰
ImageView	아이콘이나 비트맵을 출력하는 뷰
Button	사용자가 클릭하여 명령을 내릴 수 있는 뷰
EditText	문자열을 입력 받는 뷰
CheckBox	항목들을 선택하기 위한 뷰
ImageButton	텍스트가 아닌 이미지로 버튼을 표시하는 뷰
RadioButton	여러 항목 중 하나만 선택하는 뷰

## 2.5 안드로이드 UI [4/9] – View의 속성

- ✓ View 클래스는 뷰를 유일하게 식별하는 id를 지정하거나, 배경색상을 바꾸는 등 다양한 속성을 제공합니다.
- ✓ View의 하위 클래스들은 View 클래스가 가진 모든 속성을 물려받습니다.
- ✓ View의 속성은 레이아웃 XML에서 편집하거나, Java 소스로 직접 작성할 수 있습니다.
- ✓ View에 대한 속성을 이해하는 것이 화면을 구성하는 모든 클래스들을 이해하는 방법입니다.

속성	설명
id	뷰를 지칭하는 유일한 이름입니다. 사용형식 : "@+id/name", "@id/name"
layout_width layout_height	뷰의 폭과 높이를 지정합니다. 속성값 : match_parent, wrap_content, 정수 크기(단위: px, dp.. 등)
background	뷰의 배경을 지정합니다. 속성 값 : 사용자 지정 색상(예, #0000ff), 정의된 color id, 이미지 리소스 사용
padding	뷰와 내용 사이 간격을 지정합니다.
visibility	뷰의 표시여부를 지정합니다. 속성값 : visible(보이는 상태), invisible(안보이는 상태), gone(뷰에서 사라짐)
clickable longClickable	마우스 클릭 이벤트에 반응할지 여부를 지정합니다. 속성값 : true, false
focusable	키보드 포커스를 받을 수 있는지를 지정합니다. EditText나 Button은 디폴트가 true

## 2.5 안드로이드 UI (5/9) – 두 가지 구현방법

- ✓ 안드로이드에서 View를 이용해 화면을 구성하는 방법은 두 가지가 있습니다.
- ✓ 첫 번째는 XML을 사용하는 방법으로, 레이아웃 XML을 사용하여 뷰들을 배치합니다.
- ✓ 두 번째는 직접 Java 소스 코드를 통해 뷰 객체를 생성하여 화면을 구성할 수 있습니다.
- ✓ 동적으로 특정 View를 화면에 표시하는 경우를 제외하고, 대부분 레이아웃 XML을 통해 화면을 구성합니다.

### [방법1] Java 소스코드 사용하기

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
  
    super.onCreate(savedInstanceState);  
  
    // TextView 하나를 생성한다.  
    TextView mTextView = new TextView(this);  
  
    // 생성한 TextView 를 액티비티에 표시한다.  
    setContentView(mTextView);  
  
    // 보여줄 글자를 설정한다.  
    mTextView.setText("hello, android");  
}
```

*SampleActivity.java*

### [방법2] Java 소스코드 사용하기

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_sample);  
}
```

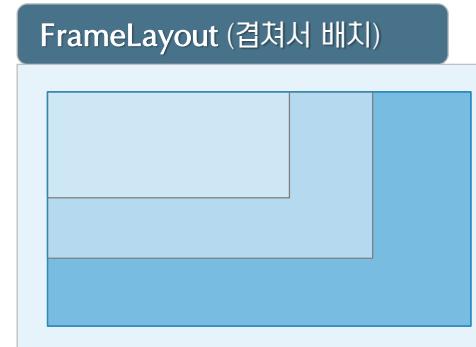
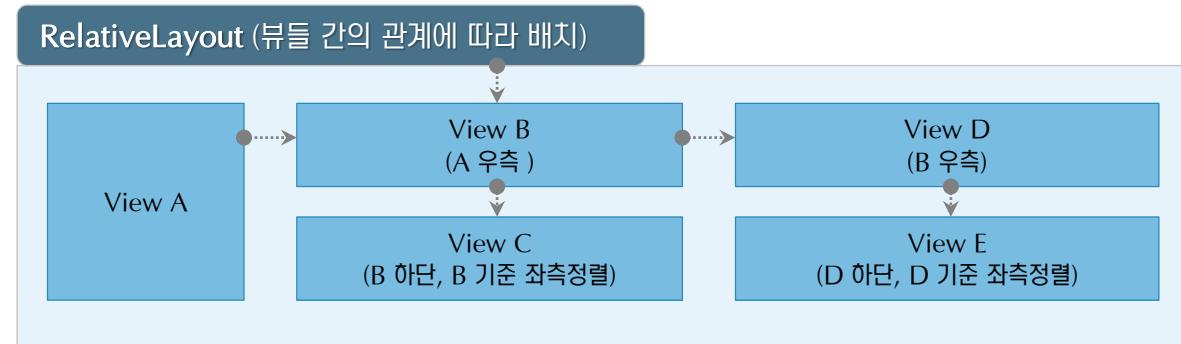
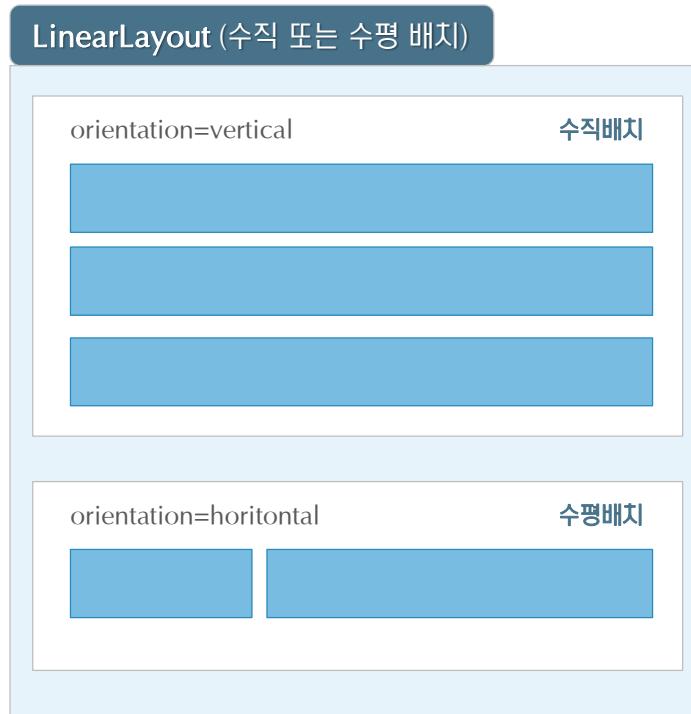
*SampleActivity.java*

```
<TextView xmlns:android=  
          "http://schemas.android.com/apk/res/android"  
          android:layout_width="match_parent"  
          android:layout_height="match_parent"  
          android:text="Hello, android~!! in XML">  
</TextView>
```

*activity\_sample.xml*

## 2.5 안드로이드 UI (6/9) – 뷰 그룹(ViewGroup)

- ✓ LinearLayout 은 자식 뷰들을 수직 또는 수평으로 배치합니다. 가중치를 통해 뷰의 상대적인 크기를 지정할 수 있습니다.
- ✓ RelativeLayout 은 가장 유연한 레이아웃으로 자식 뷰의 위치를 다른 자식 뷰들 사이의 관계로 설정하는 레이아웃입니다.
- ✓ FrameLayout 은 가장 간단한 레이아웃으로 각각의 뷰를 좌상단 모서리에 고정시키고 쌓는 형태로 배치합니다.
- ✓ TableLayout 은 테이블과 같은 형태인 행과 열로 구분된 격자를 이용하여 뷰를 배치합니다.



## 2.5 안드로이드 UI (7/9) – 치수 단위

- ✓ 뷰의 너비나 높이를 설정하기 위해 px, dip, sp 와 같은 다양한 치수 단위를 사용할 수 있습니다.
- ✓ px 는 화면을 표현하는 최소 단위(pixel)로 화면이나 이미지를 구성하는 하나의 점 단위를 의미합니다.
- ✓ dip 또는 dp는 밀도 독립 픽셀 단위라 하며, 화면 밀도에 따라 픽셀 수를 달리하는 독립된 픽셀 단위를 사용합니다.
- ✓ sp는 글꼴의 대각선 길이로 글자 크기를 설정하는 단위입니다. sp 역시 밀도독립 픽셀 단위입니다.

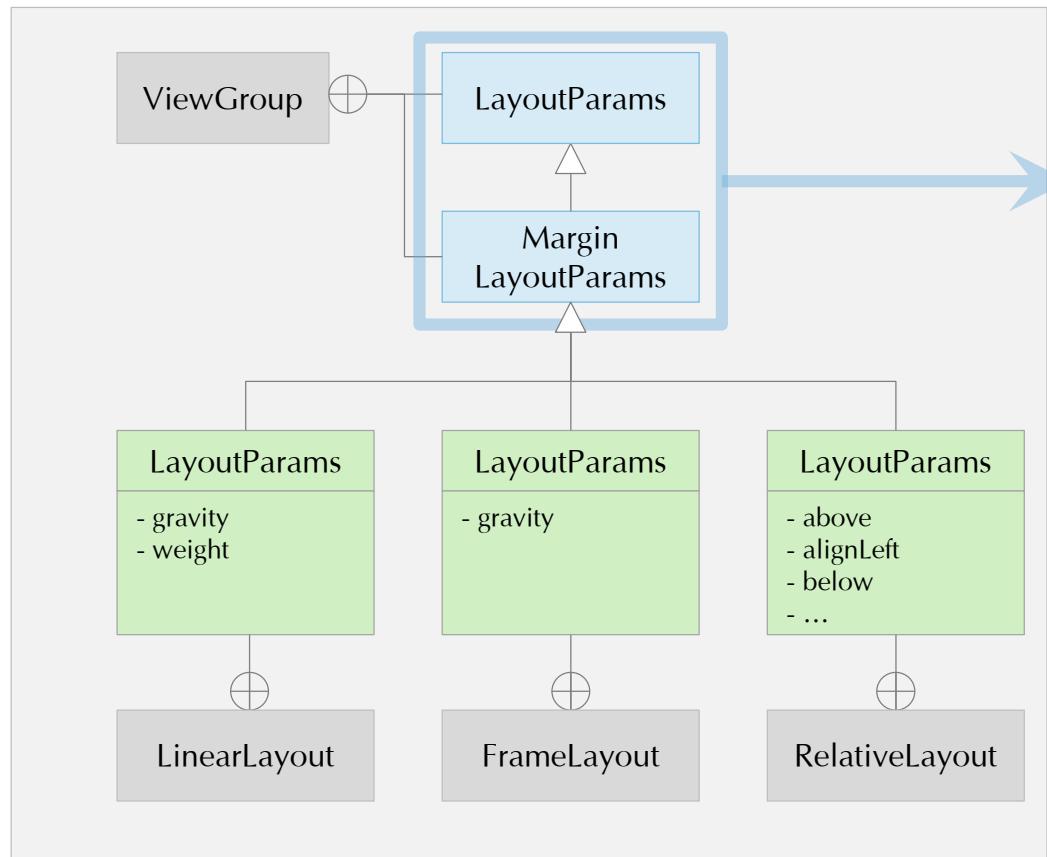
단위	설명
px (pixel)	화면이나 이미지를 구성하는 하나의 점 단위
dip 또는 dp (density-independent pixels)	해상도와 관련된 밀도독립 픽셀 단위
sp (scale-independent pixels)	글자 크기와 관련된 밀도독립 픽셀 단위
in (inch)	물리적 단위 : 2.54 cm == 1 inch
pt (point)	물리적 단위 : 1/72 inch == 1 point
mm (millimeter)	물리적 단위 : 1 cm == 10 mm



출처 : 박성근, 「이것이 안드로이드다.」 : 한빛미디어, 2014, p.374

## 2.5 안드로이드 UI (8/9) – 화면 배치 정보 (LayoutParams)

- ✓ 뷰그룹이 자식 뷰를 배치할 수 있도록 뷰가 뷰그룹에 제공하는 배치 정보를 LayoutParams 라고 합니다.
- ✓ 모든 뷰는 배치와 관련된 위치, 크기 등의 정보를 가지고 있으며, 뷰그룹은 해당 정보를 통해 자식 뷰를 배치합니다.
- ✓ 하위 뷰그룹은 최상위 ViewGroup의 LayoutParams 클래스를 상속하여 각각이 갖는 고유의 배치속성을 제공합니다.
- ✓ layout\_width 및 layout\_height 속성 값으로 wrap\_content나 match\_parent 를 주로 사용합니다.



클래스	속성	설명
ViewGroup.	layout_width	뷰의 너비 설정 (픽셀 또는 dp 값)
LayoutParams	layout_height	뷰의 높이 설정 (픽셀 또는 dp 값)
ViewGroup. Margin LayoutParams	layout_marginLeft	뷰 좌측 여백 설정
	layout_marginRight	뷰 우측 여백 설정
	layout_marginTop	뷰 상단 여백 설정
	layout_marginBottom	뷰 하단 여백 설정
	layout_margin	뷰 상/하/좌/우 여백을 동일하게 설정

### wrap\_content 와 match\_parent

<b>wrap_content</b>	<b>match_parent</b>
Android	Android

wrap\_content 는 뷰 내용에 맞추어 뷰의 크기를 조정합니다.

match\_parent 는 부모 뷰그룹에 맞추어 뷰의 크기를 조정합니다.

## 2.5 안드로이드 UI (9/9) – 레이아웃 최적화

- ✓ 레이아웃은 가능한 간단하게 유지해야 합니다.
- ✓ 불필요한 중첩을 피해야 합니다.
- ✓ 너무 많은 뷰를 사용하지 않도록 합니다.
- ✓ 깊은 중첩은 피합니다.

간단한 레이아웃 유지	레이아웃을 액티비티에 인플레이트 시키는 것은 오버헤드가 큰 작업입니다. 중첩된 레이아웃과 뷰는 애플리케이션의 성능에 많은 영향을 줄 수 있습니다.
불필요한 중첩 회피	중복된 레이아웃은 없는지 확인해야 합니다.
너무 많은 뷰 사용 지양	레이아웃에 추가된 뷰의 수가 많을 수록 인플레이트 과정에 많은 시간이 필요합니다.
깊은 중첩 회피	정해진 제한은 없지만, 중첩의 깊이는 10단계 이하로 유지합니다.

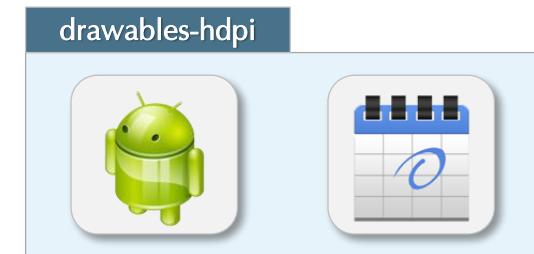
## 2.6 리소스

- ✓ 안드로이드에서 리소스는 레이아웃, 특정 형식의 값, 이미지, 색상, 스타일, 사운드 등을 의미합니다.
- ✓ 소스 코드와 리소스를 분리하면 프로그램의 확장 및 수정이 용이해서 유지 보수의 비용을 줄일 수 있습니다.
  - 예1) 문자열 리소스를 언어 별로 관리하면 소스코드 변경 없이 다국어를 지원할 수 있습니다.
  - 예2) 해상도에 따른 이미지를 별도로 관리할 수 있습니다.

리소스 폴더	설명
drawable	이미지 파일, 도형을 정의하는 XML 파일 등
layout	화면의 레이아웃을 정의하는 XML 파일
values	문자열, 색상, 배열, 크기, 스타일 등 다양한 값을 정의하는 XML 파일
menu	메뉴 구성 파일
anim	애니메이션 리소스

values

strings.xml	
title	Android PlayGround
moveBtn	Move
startBtn	Start



values-ko

strings.xml	
title	안드로이드 놀이터
moveBtn	이동
startBtn	시작



## 2.7 요약

---

- ✓ 안드로이드 프로젝트를 생성하기 위하여 애플리케이션의 기본정보를 입력하고, 사용할 SDK 버전을 입력합니다.
- ✓ 안드로이드 애플리케이션을 구성하는 컴포넌트는 액티비티, 브로드캐스트 리시버, 컨텐트 프로바이더, 서비스가 있습니다.
- ✓ `AndroidManifest.xml` 는 매니페스트라고 하며, 애플리케이션에 대한 전체 정보를 관리합니다.
- ✓ 안드로이드 화면은 화면을 구성하는 최소 단위인 뷰(View)와 뷰 들의 묶음인 뷰그룹(View Group)으로 구성합니다.
- ✓ 해상도가 서로 다른 단말기에서 동일한 모양으로 출력하기 위해서, 밀도독립 픽셀단위를 사용해야 합니다.
- ✓ 안드로이드는 리소스와 소스 코드를 분리하여 유연하고 변경이 쉬운 애플리케이션을 개발이 가능합니다.

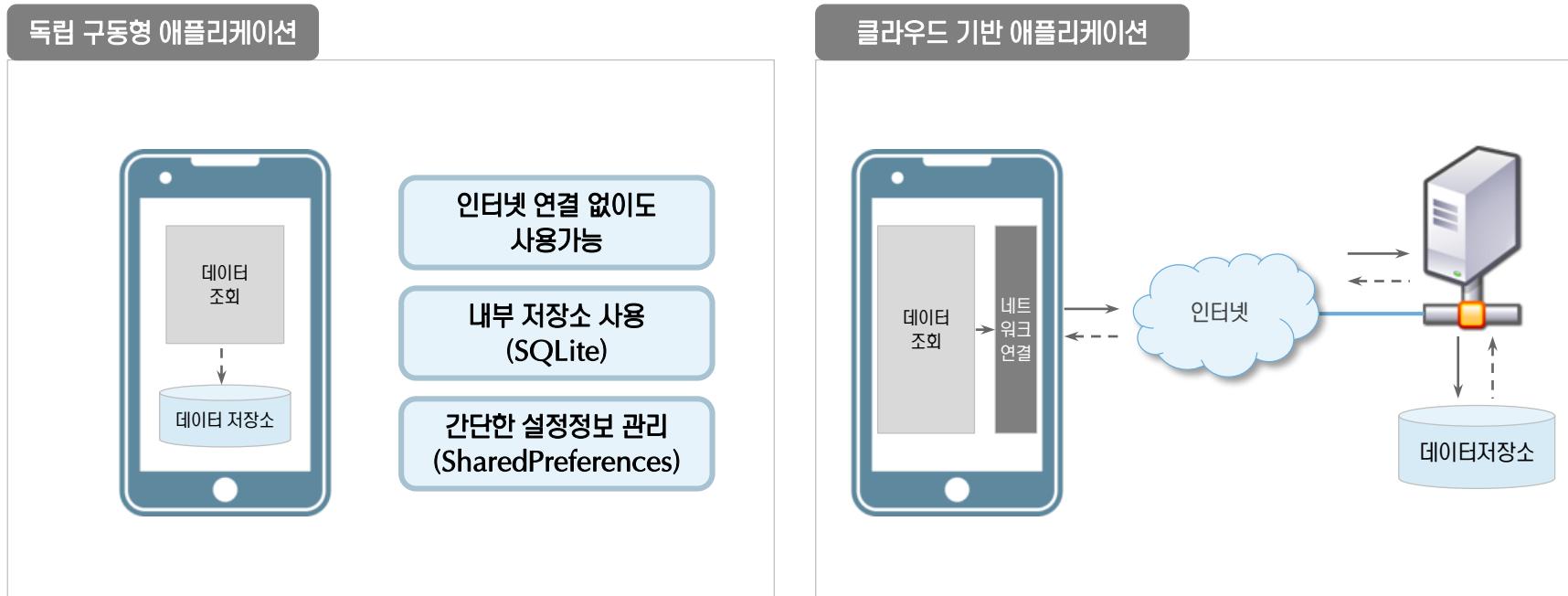


### 3. 독립 구동형 애플리케이션 개발

- 
- 3.1 독립 구동형 애플리케이션
  - 3.2 액티비티와 인텐트
  - 3.3 UI 이벤트 처리
  - 3.4 리스트뷰와 어댑터
  - 3.5 프래그먼트
  - 3.6 내부 데이터 관리
  - 3.7 다이얼로그
  - 3.8 옵션 메뉴와 컨텍스트 메뉴
  - 3.9 서비스

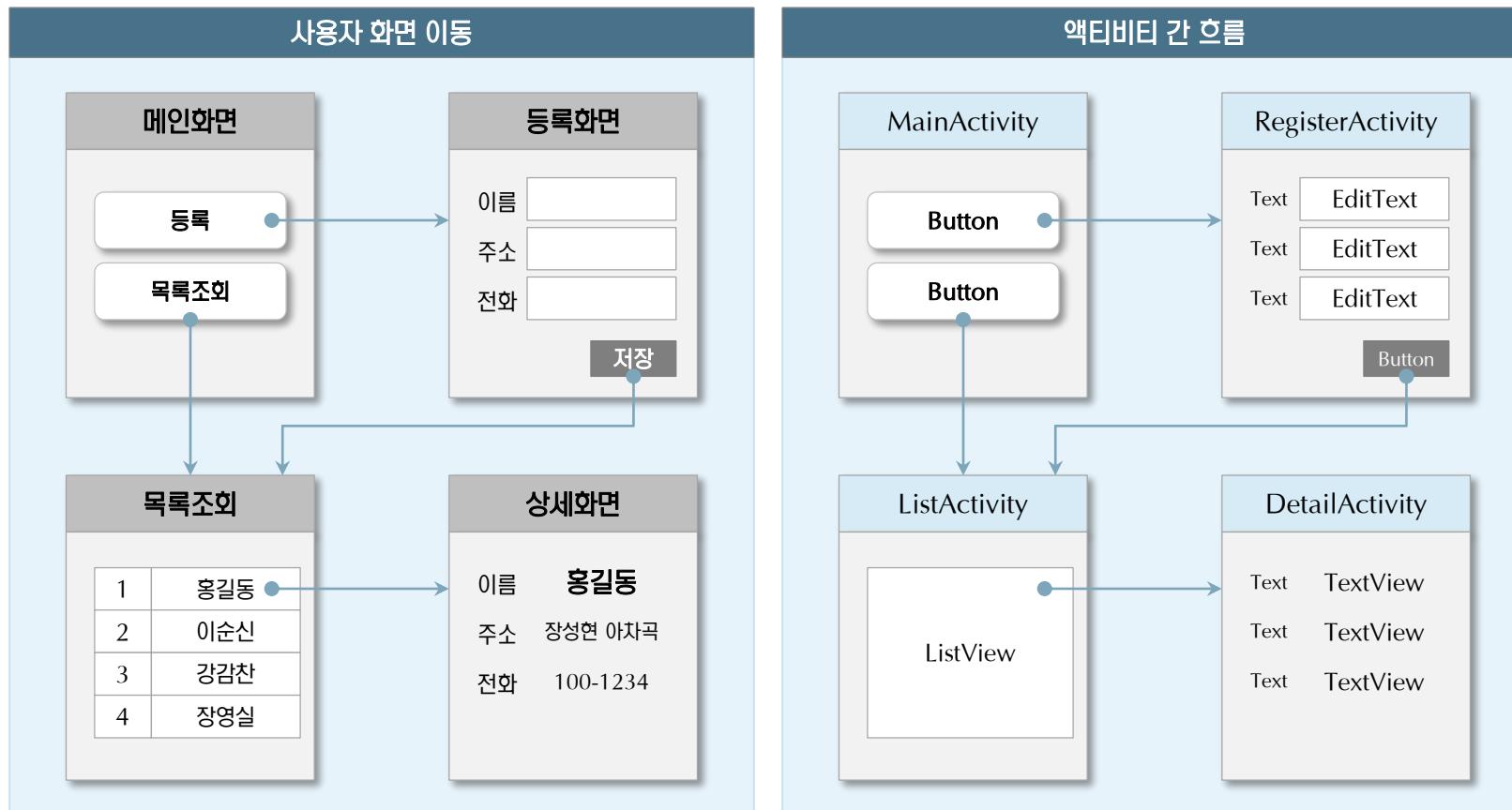
### 3.1 독립 구동형 애플리케이션

- ✓ 네트워크에 연결하지 않아도 독립적으로 구동할 수 있는 형태를 독립 구동형 애플리케이션이라고 합니다.
- ✓ 독립 구동형 애플리케이션은 애플리케이션의 상태나 사용자 데이터를 저장하기 위해 내부 저장소를 이용합니다.
- ✓ 안드로이드는 애플리케이션 내부에서 다량의 데이터를 관리할 수 있도록 관계형 데이터베이스인 SQLite를 제공합니다.
- ✓ 액티비티나 애플리케이션에서 사용하는 간단한 설정정보는 SharedPreferences를 사용하여 저장 할 수 있습니다.

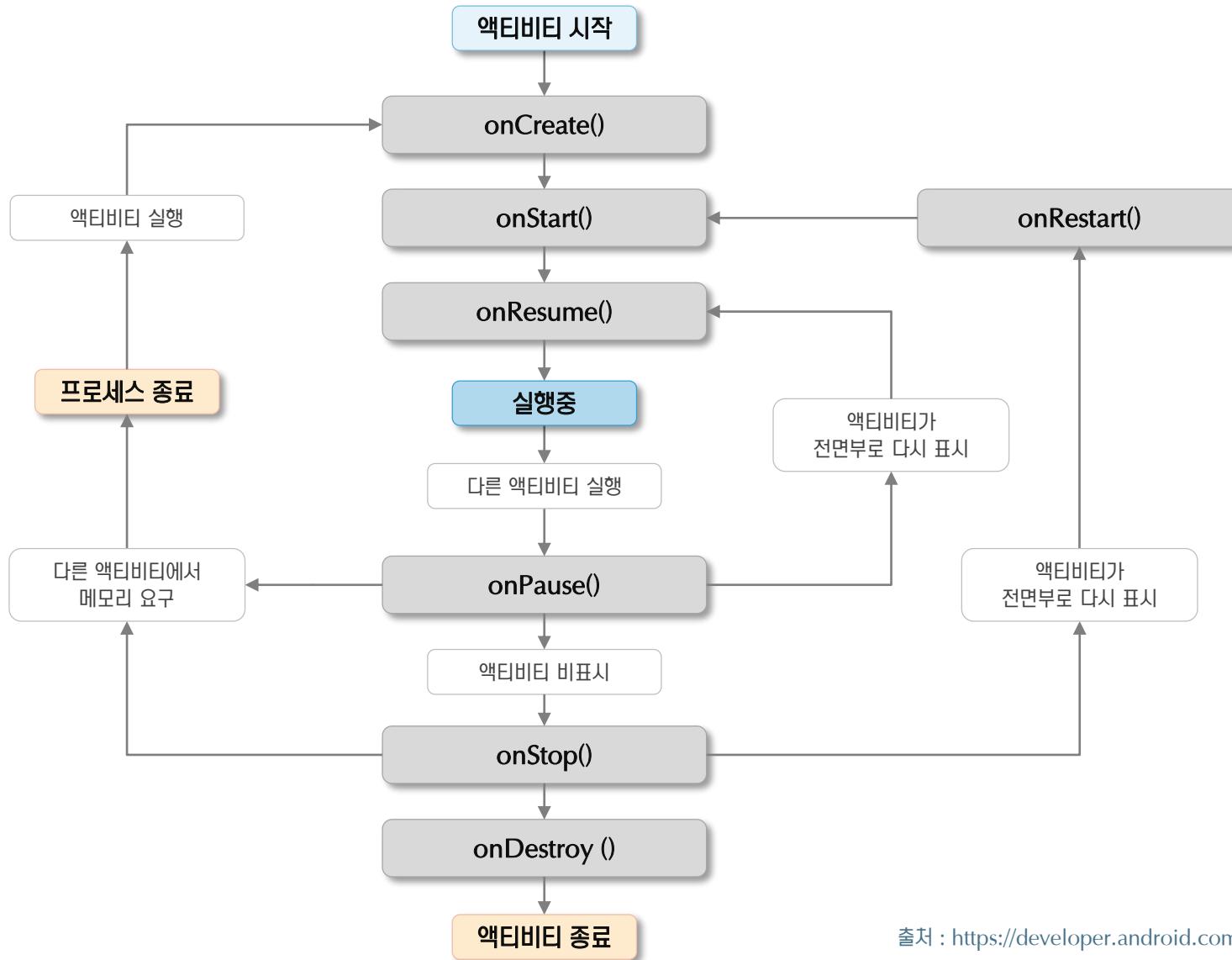


## 3.2 액티비티와 인텐트 (1/16) – 액티비티 개요

- ✓ 액티비티는 안드로이드 애플리케이션을 구성하는 4대 컴포넌트 중 하나로 사용자에게 보여지는 화면을 담당합니다..
- ✓ 액티비티는 Activity 클래스를 확장하여 구현하며, 사용자와 상호작용하는 다양한 뷰들로 구성합니다.
- ✓ 안드로이드 애플리케이션 개발에서 사용자 화면 이동은 액티비티 간의 흐름을 의미합니다.
- ✓ 안드로이드는 액티비티가 실행되고 종료될 때까지 다양한 생명주기 메소드를 호출합니다.



## 3.2 액티비티와 인텐트 (2/16) – 액티비티 생명주기 (1/4)

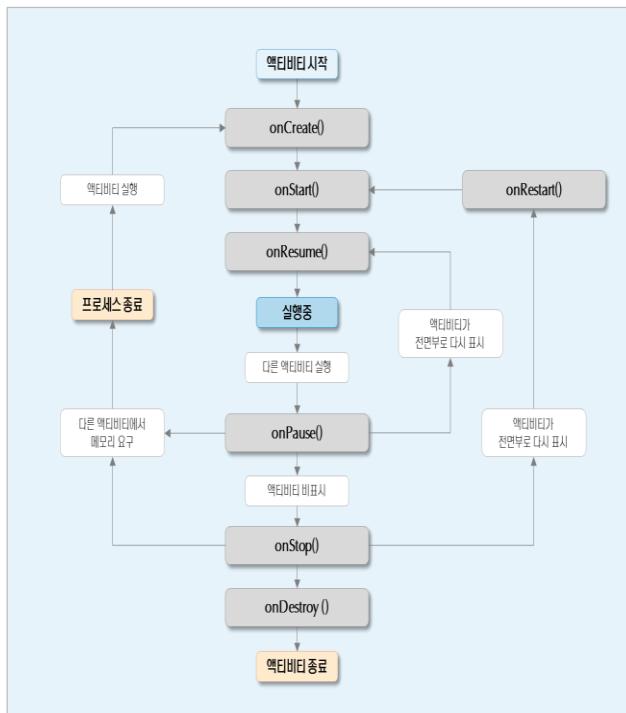


출처 : <https://developer.android.com/guide/components/activities.html>

## 3.2 액티비티와 인텐트 (3/16) – 액티비티 생명주기 (2/4)

✓ 안드로이드 프레임워크에서 ActivityManager 는 액티비티의 생명주기를 관리합니다.

- 액티비티가 실행되면 onCreate(), onStart(), onResume() 메소드를 순서대로 호출합니다.
- 이전 키를 눌러 액티비티를 종료하면 onPause(), onStop(), onDestroy() 메소드가 순서대로 호출합니다.
- 다른 액티비티가 호출되었다가 이전으로 돌아오는 경우 onRestart(), onStart(), onResume() 메소드가 차례로 호출합니다.



메소드	설명
onCreate()	Activity가 생성될 때마다 호출 layout 설정, view 생성, view와 data간의 바인딩 등과 같은 초기화 작업
onRestart()	Activity가 stop상태에서 Running 상태로 전환될 때 호출
onStart()	Activity를 화면에 표시할 수 있는 상태로 만들기 위해 호출 완료 후 Activity가 foreground에 표시되면 onResume(), 그렇지 않을 경우 onStop() 메소드가 호출됨
onResume()	Activity가 foreground에 표시되고, 사용자와 상호작용할 수 있는 상태 바로 전에 호출
onPause()	A Activity가 B Activity에 위해 포커스를 빼앗겼을 때 호출됨
onStop()	Activity가 사용자에게 전혀 보이지 않게 되면 호출 시스템이 자원이 필요하면 이 메소드 실행 중에라도 Activity가 강제 종료 될 수 있음
onDestroy()	Activity가 종료될 때 호출됨, 시스템 자원이 모자라면 강제 종료 될 수 있음

출처 : <https://developer.android.com/guide/components/activities.html>

## 3.2 액티비티와 인텐트 (4/16) – 액티비티 생명주기 (3/4)

- ✓ 액티비티 생명주기 메소드 호출 시점을 확인하기 위하여 로그를 남기는 방법을 살펴봅니다.
- ✓ 로그란 프로그램 실행 중에 발생한 각종 정보를 모니터링하거나 디버깅하기 위한 용도로 사용합니다.
- ✓ 안드로이드는 로그를 출력하는 Log 클래스를 제공합니다. 로그는 5가지 형태로 기록 가능합니다.
- ✓ 실행 중에 출력되는 로그는 안드로이드 개발환경이 제공하는 LogCat을 통해 실시간으로 확인할 수 있습니다.

로그레벨	메소드	설명
Verbose	Log.v	다양한 정보를 출력하기 위한 용도
Debug	Log.d	문제 발생 시 원인 분석의 기반이 되는 정보를 출력하기 위한 용도
Information	Log.i	각종 클래스 변수의 내용을 확인하기 위한 용도
Warning	Log.w	실행 중인 앱이 문제가 발생할 소지가 있음을 알리는 용도
Error	Log.e	실행 중인 앱에 치명적인 에러가 발생했음을 알리는 용도



## 3.2 액티비티와 인텐트 (5/16) – 액티비티 생명주기 (4/4)

- ✓ 액티비티 생명주기 로그를 확인하기 위한 새로운 액티비티 클래스를 추가하여 아래와 같이 구현합니다.
  - 클래스 명은 LifecycleLogActivity 이며, 로그 태그명은 L\_TAG 상수에 클래스 명으로 초기화하도록 선언합니다.
  - Log.d를 통해 생명주기 메소드마다 디버그 레벨로 로그를 남기도록 코드를 추가합니다.
- ✓ 구현이 완료되면, 액티비티를 실행하여 액티비티 상태가 변경될 때마다 로그가 출력되는지 확인합니다.

```
public class LifecycleLogActivity extends Activity {  
  
    private static final String L_TAG =  
        LifecycleLogActivity.class.getSimpleName();  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        Log.d(L_TAG, "onCreate() invoked");  
    }  
  
    @Override  
    protected void onStart() {  
        super.onStart();  
        Log.d(L_TAG, "onStart() invoked");  
    }  
  
    @Override  
    protected void onRestart() {  
        super.onRestart();  
        Log.d(L_TAG, "onRestart() invoked");  
    }  
  
    @Override  
    protected void onResume() {  
        super.onResume();  
        Log.d(L_TAG, "onResume() invoked");  
    }  
  
    @Override  
    protected void onPause() {  
        super.onPause();  
        Log.d(L_TAG, "onPause() invoked");  
    }  
  
    @Override  
    protected void onStop() {  
        super.onStop();  
        Log.d(L_TAG, "onStop() invoked");  
    }  
  
    @Override  
    protected void onDestroy() {  
        super.onDestroy();  
        Log.d(L_TAG, "onDestroy() invoked");  
    }  
}
```

## 3.2 액티비티와 인텐트 [6/16] – 다른 액티비티에 값 전달

- ✓ 다른 액티비티 호출할 때에는 `startActivity()` 메소드에 선언된 인텐트 객체를 넘겨 호출을 시작합니다
- ✓ 다른 액티비티에 전달되는 값은 거의 모든 타입에 대해 오버로딩되어 있으며 배열이나 객체까지도 저장할 수 있습니다.
  - 객체를 전달할 때에는 해당 객체가 `Serializable` 인터페이스를 구현하고 있어야 합니다.
  - 또한, 액티비티가 전달된 객체를 받을 때에는 인텐트의 `getSerializableExtra()` 메소드를 통해서 받습니다.

다른 액티비티를 호출

```
Intent intent = new Intent(MyActivity.this, OtherActivity.class);
startActivity(intent);
```

다른 액티비티에 값 전달

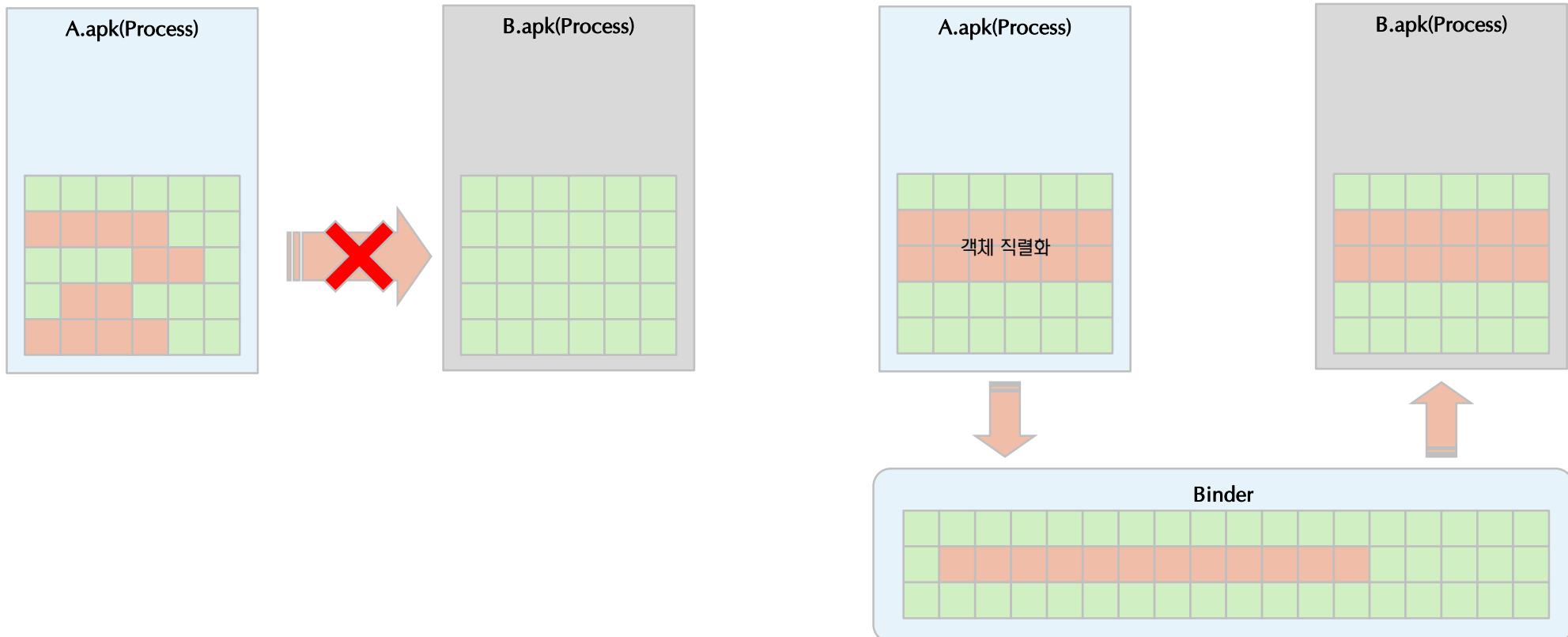
```
Intent intent = new Intent(MyActivity.this, OtherActivity.class);
intent.putExtra("name", "홍길동");
startActivity(intent);
```

다른 액티비티의 전달된 값 받기

```
// Other Activity
Intent intent = getIntent();
String value = intent.getStringExtra("name");
```

## 3.2 액티비티와 인텐트 [6/16] – 다른 액티비티에 값 전달

- ✓ 컴포넌트간에 데이터를 전달하기 위해서는 객체 직렬화가 필요합니다.
- ✓ 데이터의 이동은 리눅스 커널의 바인더가 진행합니다.
- ✓ 안드로이드에서 객체 직렬화는 자바의 Serializable 인터페이스를 구현합니다.
- ✓ 또는 안드로이드의 Parcel, Bundle 객체를 이용할 수도 있습니다.(권장: Bundle의 이용)



## 3.2 액티비티와 인텐트 (7/16) – 결과 값을 받는 액티비티

✓ 결과값을 받기 위한 액티비티 호출을 위해서 `startActivityForResult()` 메소드를 사용합니다.

- 메소드 인자의 두 번째 값은 콜백 메소드의 `requestCode` 값을 의미하며 0 이상의 중복되지 않는 정수를 넘깁니다.
- 만약, 음수 값을 넘기는 경우 리턴을 받지 않겠다는 의미이며, 예제의 1은 호출 대상의 식별자로 사용합니다.

✓ 전달된 값을 받기 위한 콜백 함수 정의는 호출된 액티비티가 끝나면 `onActivityResult()` 메소드를 호출합니다.

결과값을 받기 위한 액티비티 호출

```
Intent intent = new Intent(MyActivity.this, OtherActivity.class);
startActivityForResult(intent, 1);
```

전달된 값을 받기 위한 콜백 함수

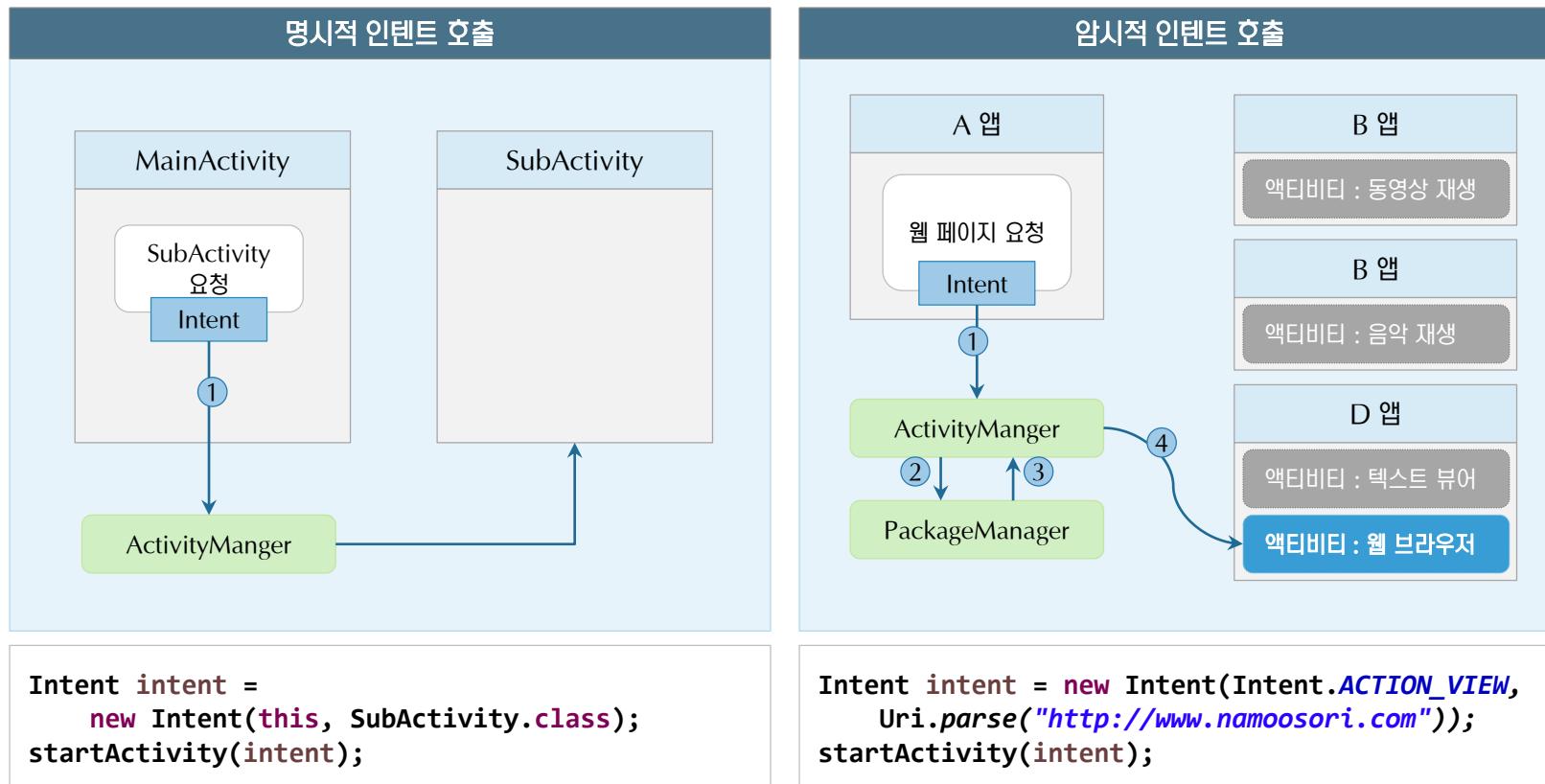
```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case 1:
            if (resultCode == RESULT_OK) {
                String value = data.getStringExtra("key");
            }
            break;
    }
}
```

호출된 액티비티에서 값 전달

```
// Other Activity
Intent intent = new Intent();
intent.putExtra("key", "value");
setResult(RESULT_OK, intent);
finish();
```

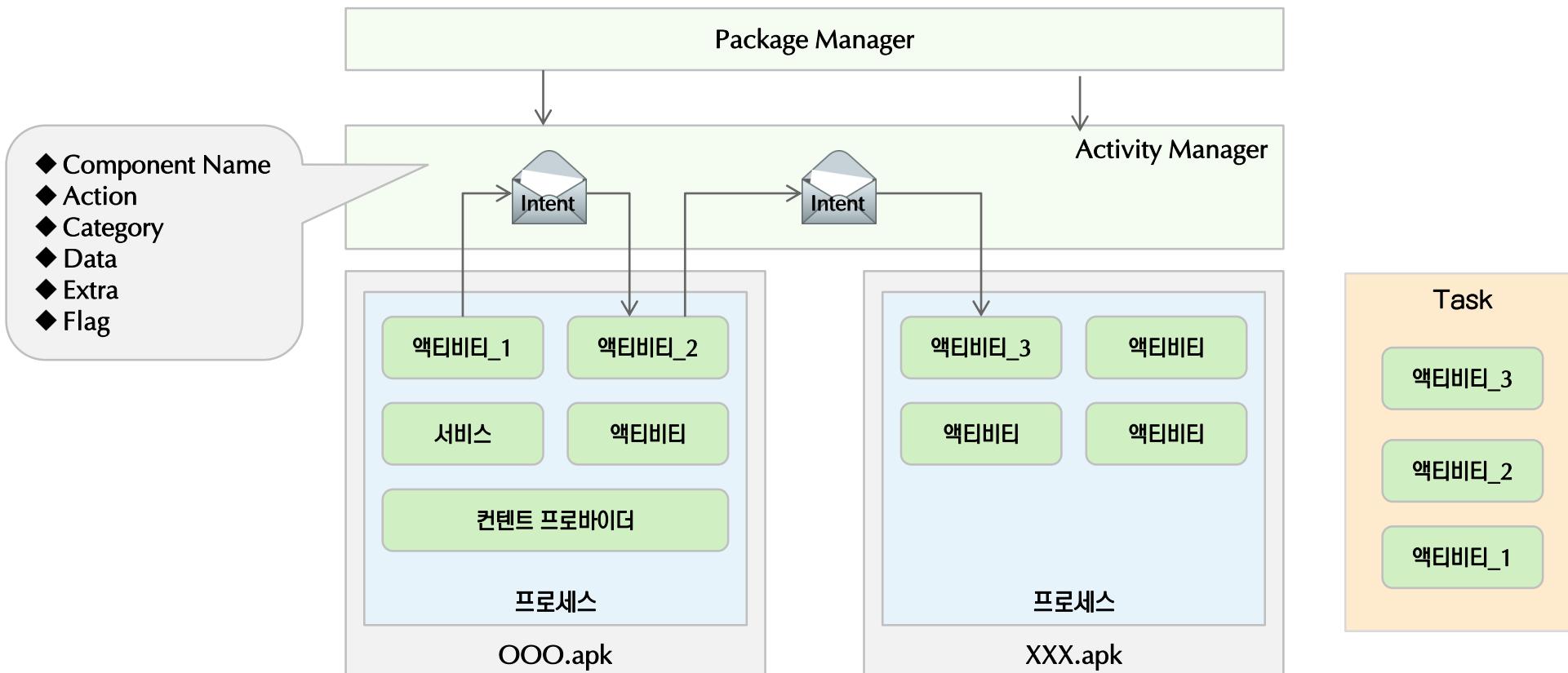
## 3.2 액티비티와 인텐트 [8/16] – 인텐트 개요

- ✓ 인텐트(Intent)는 애플리케이션 컴포넌트 간에 상호작용을 위하여 주고 받는 메시지입니다.
- ✓ 대상 컴포넌트를 지정하는 방식에 따라 명시적 인텐트와 암시적 인텐트로 구분합니다.
- ✓ 명시적(Explicit) 인텐트는 실행할 대상 컴포넌트를 정확히 명시하는 방식입니다.
- ✓ 암시적(Implicit) 인텐트는 액션, 카테고리 등을 포함하여 인텐트 필터에 의해 대상 컴포넌트를 식별하는 방식입니다.



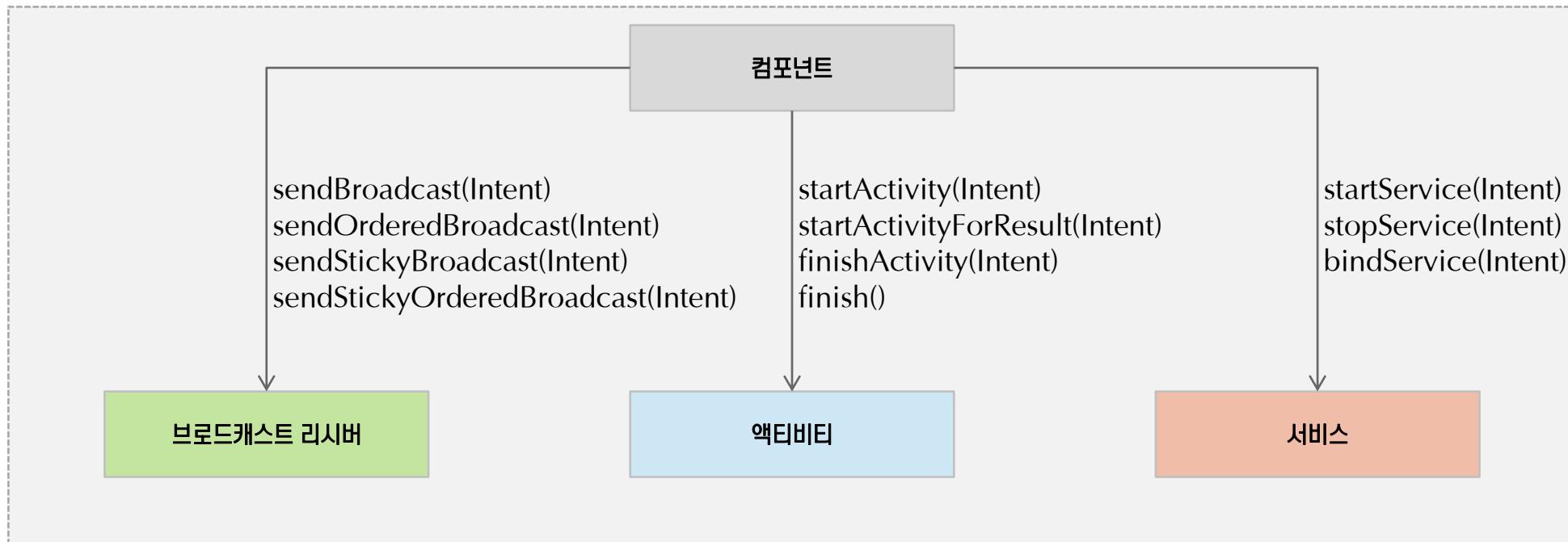
## 3.2 액티비티와 인텐트 (9/16) – 인텐트의 구성요소

- ✓ 인텐트 클래스의 핵심 속성들은 Component Name, Action, Category, Data, Extra, Flag 입니다.
- ✓ Component Name 속성은 명시적으로 컴포넌트를 호출할 때 사용합니다.
- ✓ Action, Category, Data 속성은 암시적으로 컴포넌트를 호출할 때 사용합니다.
- ✓ Flag 는 해당 인텐트에 대한 메타데이터로 Task 관리에 사용되며, Extra 속성은 데이터를 담아 전달할 때 사용합니다.



## 3.2 액티비티와 인텐트 (10/16) –명시적 인텐트

- ✓ 인텐트를 통해 활성화 가능한 컴포넌트는 액티비티, 서비스, 브로드캐스트 리시버입니다.
- ✓ 명시적 인텐트는 활성화 대상 컴포넌트의 이름이 지정되어 있는 인텐트를 뜻합니다.
- ✓ 명시적으로 컴포넌트의 이름을 지정할 때는 인텐트 클래스의 생성자, setComponent(), setClass(), setClassName() 메소드를 이용합니다.



## 3.2 액티비티와 인텐트 (11/16) – 암시적 인텐트

- ✓ 명시적 인텐트는 개발자가 직접 작성한 컴포넌트 이외에는 호출할 수 없습니다.
- ✓ 안드로이드가 제공하고 있는 다양한 컴포넌트를 활용하기 위해서는 암시적 인텐트를 활용해야 합니다.
- ✓ 암시적 인텐트에 특정 작업을 지정하면 해당 작업을 수행할 수 있는 모든 컴포넌트를 호출할 수 있습니다.
- ✓ 암시적 인텐트에 사용되는 인텐트 클래스의 속성은 Action, Category, Data 입니다.

The diagram illustrates the connection between an implicit intent and its declaration in the AndroidManifest.xml file. On the left, a blue dashed box encloses a white envelope icon and a light green box titled "암시적 인텐트" (Implicit Intent) containing a list of attributes: Component Name, Action, Category, Data, Extra, and Flag. A blue dashed line connects this box to the right side of the diagram. On the right, a dark grey box contains the XML code for the AndroidManifest.xml file. The code defines a manifest with a package of "kr.co.nextree.example.helloandroid". It includes an application tag with an allowBackup attribute set to true and a label string resource. Inside the application tag is an activity tag for ".MainActivity" with its own label string resource. This activity tag contains an intent-filter section with an action of "android.intent.action.MAIN" and a category of "android.intent.category.LAUNCHER". The entire XML structure is enclosed in a manifest tag.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="kr.co.nextree.example.helloandroid"
    android:versionCode="1"
    android:versionName="1.0" >
    <application android:allowBackup="true" android:label="@string/app_name" >
        <activity android:name=".MainActivity" android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

## 3.2 액티비티와 인텐트 (12/16) – 암시적 인텐트의 활용

- ✓ 암시적 인텐트 활용에 대해서 아래와 같이 살펴보도록 합니다.
  - 전화걸기, 메일보내기, 웹 브라우저 호출
- ✓ Action을 이용한 컴포넌트 활성화는 안드로이드가 제공하는 다양한 컴포넌트를 사용하는데 이용합니다.
- ✓ 사용자는 자신만의 Action을 정의하여 사용할 수 있습니다.

활용1. 전화걸기

```
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:010-1234-5678"));  
startActivity(intent);
```

활용2. 메일보내기

```
Intent intent = new Intent(Intent.ACTION_SENDTO, Uri.parse("mailto:androidkim@company.com"));  
startActivity(intent);
```

활용3. 웹 브라우저 호출

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"));  
startActivity(intent);
```

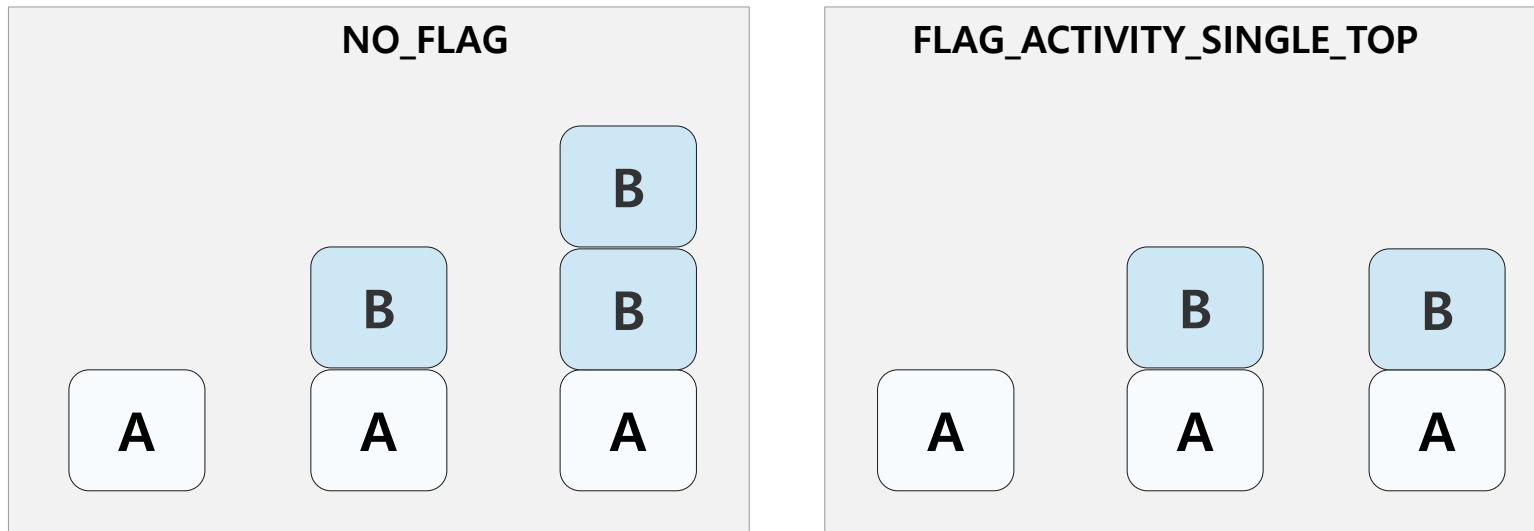
활용4. Action 이용하기

```
Intent intent = new Intent(Intent.ACTION_MAIN);  
intent.setComponent(new ComponentName("com.myexample.helloandroid",  
"com.myexample.helloandroid.MainActivity"));  
startActivity(intent);
```

## 3.2 액티비티와 인텐트 (13/16) – 인텐트 Flag로 Activity 스택 관리 (1/4)

- ✓ 액티비티 스택은 안드로이드가 Task를 관리하는 방법입니다.
- ✓ Intent Flag를 이용하면 액티비티 스택을 관리 할 수 있습니다.
- ✓ 호출하는 액티비티가 자신을 다시 호출하는 경우, 기존의 액티비티를 재활용하도록 합니다.
- ✓ 이는 인텐트 플래그를 FLAG\_ACTIVITY\_SINGLE\_TOP 으로 선언하여 액티비티를 재활용합니다.

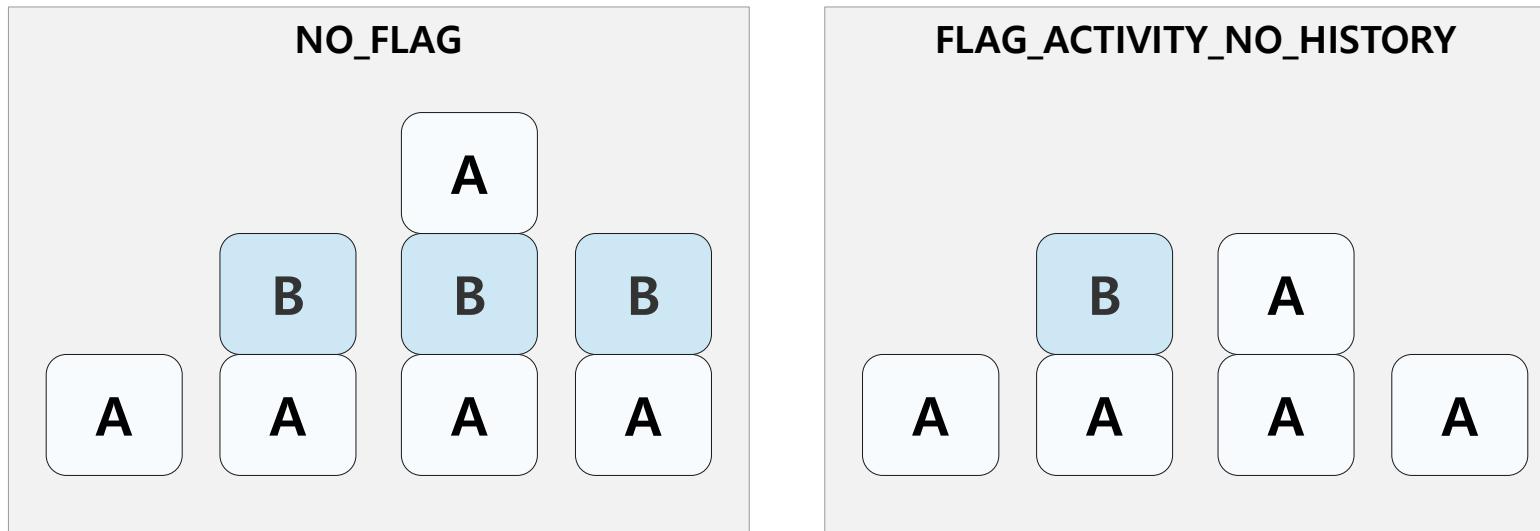
```
Intent intent = new Intent(Bactivity.this, Bactivity.class);
// 인텐트 플래그 FLAG_ACTIVITY_SINGLE_TOP 선언을 통해 기존 액티비티를 재활용 합니다.
intent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
startActivity(intent);
```



## 3.2 액티비티와 인텐트 (14/16) – 인텐트 Flag로 Activity 스택 관리 (2/4)

- ✓ 액티비티가 스택에 그 흔적을 남기지 않도록 설정합니다.
- ✓ 이는 인텐트 플래그를 FLAG\_ACTIVITY\_NO\_HISTORY 로 선언하여 액티비티의 이력을 남기지 않습니다.
- ✓ 액티비티의 이력을 남기지 않을 경우 사용자가 back 버튼을 눌렀을 때 이전 액티비티가 나타나지 않습니다.
- ✓ 다이얼로그 형태의 화면을 구성하거나 일회성으로 사용되는 액티비티를 만들 때 주로 사용 합니다.

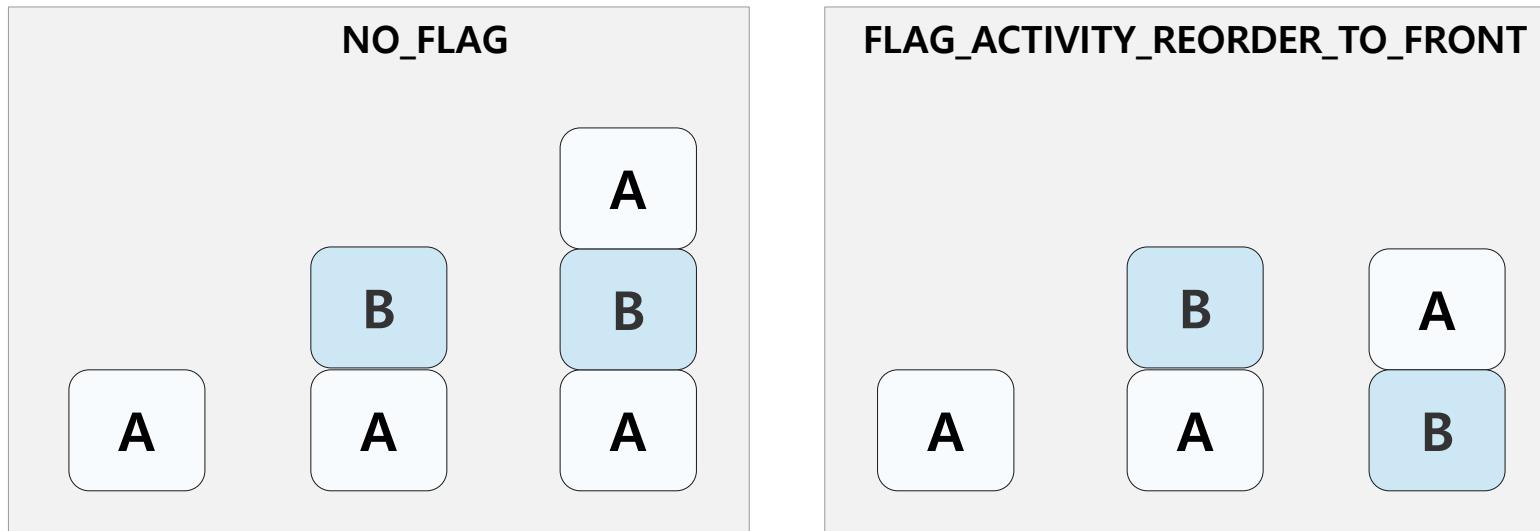
```
Intent intent = new Intent(MyActivity.this, TargetActivity.class);
// 인텐트 플래그 FLAG_ACTIVITY_NO_HISTORY 선언을 통해 액티비티 이력을 남기지 않습니다.
intent.setFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
startActivity(intent);
```



## 3.2 액티비티와 인텐트 (15/16) – 인텐트 Flag로 Activity 스택 관리 (3/4)

- ✓ FLAG\_ACTIVITY\_REORDER\_TO\_FRONT는 동일한 액티비티가 스택에 쌓이는 것을 방지합니다.
- ✓ 실행하고자 하는 액티비티가 존재한다면 생성 대신 순서를 가장 앞으로 재정렬하도록 설정합니다.
- ✓ 이는 인텐트 플래그를 FLAG\_ACTIVITY\_REORDER\_TO\_FRONT로 선언하여 순서를 재정렬합니다.

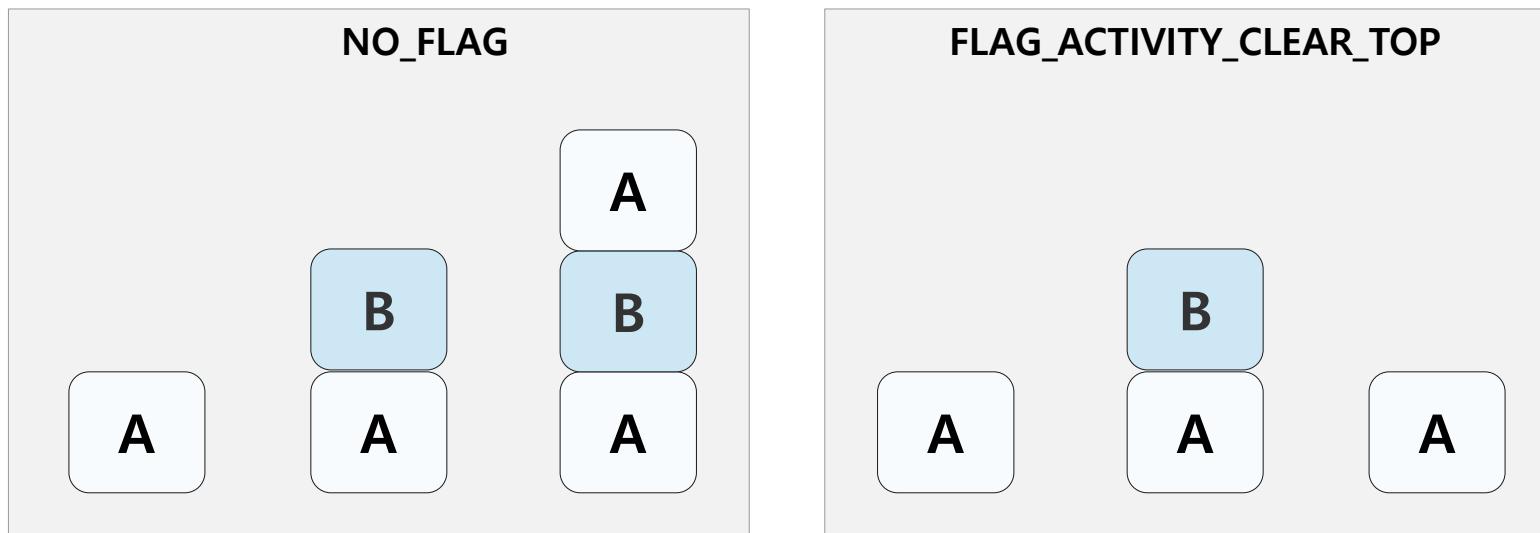
```
Intent intent = new Intent(MyActivity.this, TargetActivity.class);
// 인텐트 플래그 FLAG_ACTIVITY_REORDER_TO_FRONT 선언을 통해 순서를 앞으로 재정렬합니다.
intent.setFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
startActivity(intent);
```



## 3.2 액티비티와 인텐트 (16/16) – 인텐트 Flag로 Activity 스택 관리 (4/4)

- ✓ 실행 할 액티비티가 이미 스택에 존재한다면, 해당 액티비티 위에 존재하는 다른 모든 액티비티를 종료시키도록 설정합니다.
- ✓ 이는 인텐트 플래그를 FLAG\_ACTIVITY\_CLEAR\_TOP 으로 선언하여 스택에 존재하는 액티비트를 종료 시킵니다.
- ✓ 이 플래그를 사용하면 루트가 되는 액티비티와 호출된 액티비티만 스택에 남습니다.

```
Intent intent = new Intent(MyActivity.this, TargetActivity.class);
// 인텐트 플래그 FLAG_ACTIVITY_REORDER_TO_FRONT 선언을 통해 순서를 앞으로 재정렬합니다.
intent.setFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
startActivity(intent);
```



## 3.3 UI 이벤트 처리 (1/7) – 리스너 인터페이스

- ✓ 리스너 인터페이스는 이벤트 발생 시에 해당 이벤트를 리스너가 받아 처리할 수 있도록 정의한 인터페이스입니다.
- ✓ 이벤트 처리는 이벤트에 해당하는 리스너 인터페이스의 메소드에서 처리하도록 구현해야 합니다.
- ✓ 아래는 대표적인 리스너 인터페이스와 각 리스너에 선언된 이벤트 처리 메소드입니다.

리스너	이벤트 처리 메소드
OnTouchListener	boolean onTouch(View v, MotionEvent event)
OnKeyListener	boolean onKey(View v, int keyCode, KeyEvent event)
OnClickListener	void onClick(View v)
OnLongClickListener	boolean onLongClick(View v)
OnFocusChangeListener	void onFocusChange(View v, boolean hasFocus)

### 3.3 UI 이벤트 처리 (2/7) – 리스너 인터페이스 구현 (1/5)

✓ 리스너 인터페이스를 구현하는 방법은 클래스 선언에 따라 다양한 방법으로 구현할 수 있습니다.

- 일반적인 클래스(내부 클래스 포함)를 통해 리스너 인터페이스를 구현하는 경우
- 액티비티 클래스가 리스너 인터페이스를 구현하는 경우
- 익명 내부 클래스를 통해 리스너 인터페이스를 구현하는 경우
- 익명 내부 클래스의 임시 객체를 사용하여 리스너 인터페이스를 구현하는 경우

```
/** 일반적인 클래스를 사용한 리스너 인터페이스 구현 */
public class TouchListener implements View.OnTouchListener {
    @Override
    public boolean onTouch(View view, MotionEvent event) {
        ...
    }
}
```

```
/** 액티비티 클래스가 리스너인터페이스 구현 */
public class MyMainActivity extends Activity
    implements View.OnTouchListener {
    @Override
    public boolean onTouch(View view, MotionEvent event) {
        ...
    }
}
```

```
/** 익명 내부 클래스를 통한 리스너 인터페이스 구현 */
public class MyMainActivity extends Activity {
    View.OnTouchListener touchListener =
        new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent event) {
                ...
            }
        };
}
```

```
/** 익명 내부 클래스의 임시 객체를 통한 리스너 인터페이스 구현 */
public class MyMainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        mBtn.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent event) {
                ...
            }
        });
    }
}
```

### 3.3 UI 이벤트 처리 (3/7) – 리스너 인터페이스 구현 (2/5)

- ✓ 일반적인 클래스(내부 클래스 포함)를 통해 리스너 인터페이스를 구현하는 방식입니다.
- ✓ 동일한 이벤트 처리는 다양한 클래스에 사용할 경우 구현하는 방식입니다.
- ✓ 별도 클래스로 선언하므로 다른 여러 액티비티에서 해당 리스너 인터페이스 구현 클래스를 선언하여 사용할 수 있습니다.

```
// 1. 리스너 구현 클래스 선언
public class TouchListener implements View.OnTouchListener {
    @Override
    public boolean onTouch(View view, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            Toast.makeText(MyActivity.this, "Touch Event Call~~", Toast.LENGTH_SHORT).show();
            return true;
        }
        return false;
    }
}

public class MyActivity extends Activity {
    // 2. 리스너 객체 생성
    TouchListener touchListener = new TouchListener();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        // 3.리스너 등록
        mBtn.setOnTouchListener(touchListener);
    }
}
```

### 3.3 UI 이벤트 처리 (4/7) – 리스너 인터페이스 구현 (3/5)

- ✓ 액티비티 클래스가 리스너 인터페이스를 구현하는 방식입니다.
- ✓ 즉, 액티비티 클래스가 이벤트 핸들러 역할을 함께 갖게 되는 방식입니다.
- ✓ 액티비티 클래스가 리스너 이므로 리스너 등록은 별도 리스너 객체를 생성하지 않고 액티비티 객체 자체를 등록합니다.

```
// 1. 액티비티 클래스를 통한 리스너 구현 클래스 선언
public class MyMainActivity extends Activity implements View.OnTouchListener {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button mBtn = (Button) findViewById(R.id.btn);

        // 2.리스너 등록
        mBtn.setOnTouchListener(this);
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            Toast.makeText(MyActivity.this, "Touch Event Call~~", Toast.LENGTH_SHORT).show();
            return true;
        }
        return false;
    }
}
```

### 3.3 UI 이벤트 처리 (5/7) – 리스너 인터페이스 구현 (4/5)

- ✓ 익명 내부 클래스를 통한 리스너 인터페이스를 구현하는 방식입니다.
- ✓ 선언적인 클래스 정의가 아니므로 해당 액티비티 외 다른 액티비티에서 해당 리스너 객체를 사용할 수 없습니다.
  - 단, 리스너 객체에 대한 접근지시자 또는 접근할 수 있는 별도 메소드가 제공되는 경에는 사용이 가능합니다.

```
public class MyMainActivity extends Activity {  
    // 1. 리스너 객체 생성 및 리스너 구현  
    View.OnTouchListener touchListener = new View.OnTouchListener() {  
        @Override  
        public boolean onTouch(View v, MotionEvent event) {  
            if(event.getAction() == MotionEvent.ACTION_DOWN) {  
                Toast.makeText(MyActivity.this, "Touch Event Call~~", Toast.LENGTH_SHORT).show();  
                return true;  
            }  
            return false;  
        }  
    };  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Button mBtn = (Button)findViewById(R.id.btn);  
  
        // 2. 리스너 등록  
        mBtn.setOnTouchListener(touchListener);  
    }  
}
```

### 3.3 UI 이벤트 처리 (6/7) – 리스너 인터페이스 구현 (5/5)

- ✓ 익명 내부 클래스의 임시 객체 사용을 통한 리스너 인터페이스를 구현하는 방식입니다.
- ✓ 리스너 객체에 대한 선언이 없으므로 해당 액티비티 외에 다른 액티비티에서 해당 리스너 객체를 사용할 수 없습니다.
  - 단, 리스너가 등록된 컴포넌트에서 해당 리스너 객체에 접근이 가능한 경우에는 리스너 객체 사용이 가능합니다.

```
public class MyMainActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button mBtn = (Button)findViewById(R.id.btn);

        // 1. 리스너 객체 생성 및 리스너 구현, 리스너 등록
        mBtn.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                if(event.getAction() == MotionEvent.ACTION_DOWN) {
                    Toast.makeText(MyActivity.this, "Touch Event Call~~", Toast.LENGTH_SHORT).show();
                    return true;
                }
                return false;
            }
        });
    }
}
```

### 3.3 UI 이벤트 처리 (7/7) – 레이아웃에서 이벤트 메소드 지정

- ✓ View가 정의할 수 있는 onClick 속성을 이용해 이벤트를 처리하는 방식입니다.
- ✓ 레이아웃을 정의할 때 이벤트를 처리하는 View의 속성에 onClick 속성을 지정합니다.
- ✓ onClick의 속성 값은 메소드의 이름입니다. (예:mOnClick)
- ✓ 이 방식은 이벤트 리스너 등록을 생략할 수 있습니다.

```
public class MyMainActivity extends Activity {

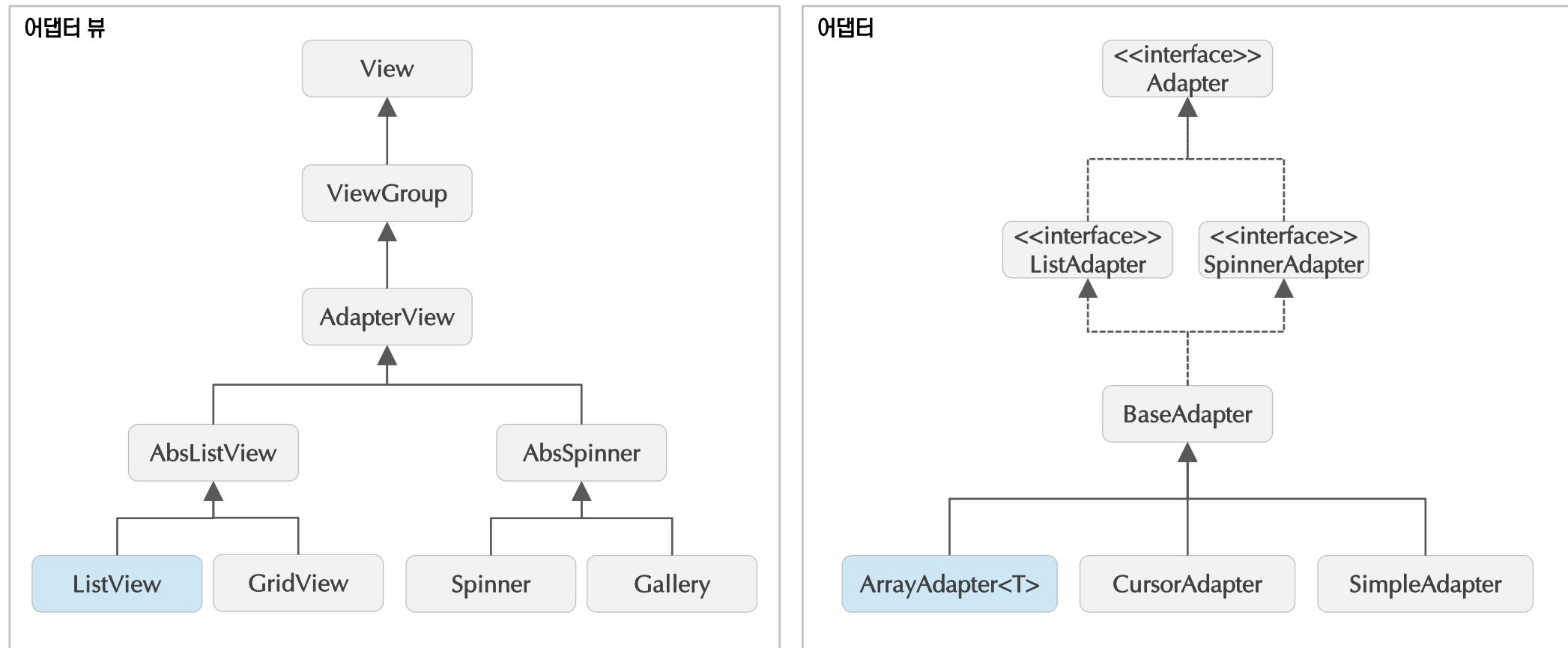
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_layout);
    }

    public void mOnClick(View v) {
        TextView textView = (TextView)findViewById(R.id.title);

        switch(v.getId()) {
            case R.id.btn1:
                textView.setText("btn1");
                break;
            case R.id.btn2:
                textView.setText("btn2");
                break;
        }
    }
}
```

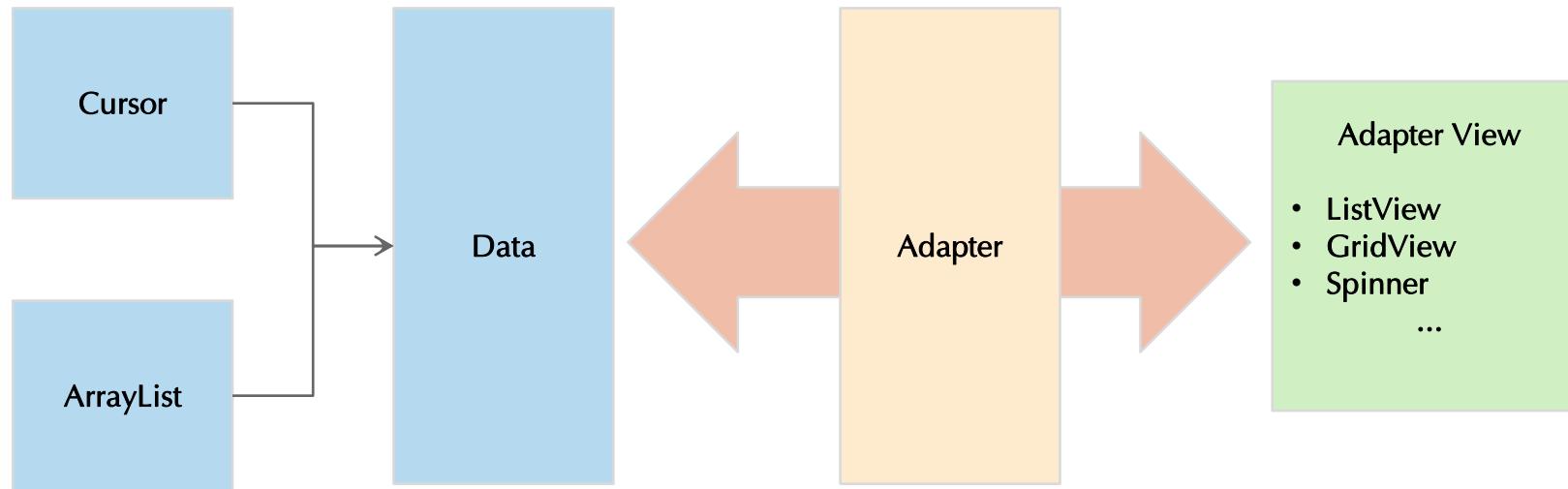
## 3.4 리스트뷰와 어댑터 (1/6) – 어댑터 뷰와 어댑터

- ✓ 어댑터 뷰는 대량의 데이터를 출력하기 위한 뷰를 뜻합니다.
- ✓ 어댑터는 대량의 데이터를 관리하며 어댑터 뷰에 데이터를 전달하는 역할을 합니다.
- ✓ 따라서, 어댑터는 표시할 항목 데이터에 대해 각 항목이 어떻게 표시될지를 지정해야 합니다.



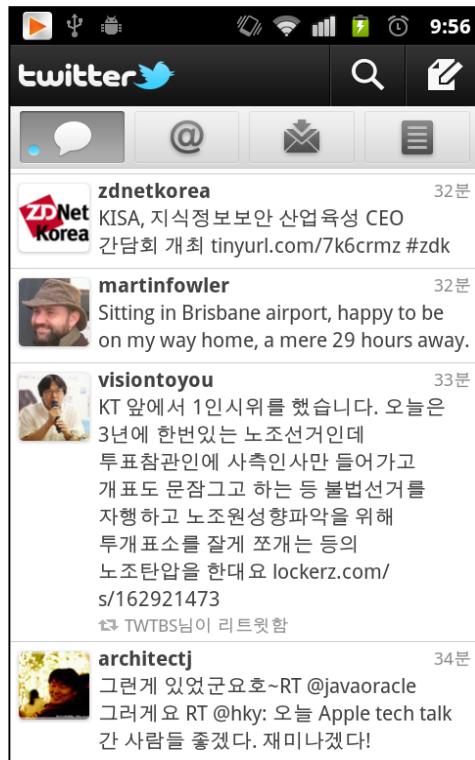
## 3.4 리스트뷰와 어댑터 [2/6] – 어댑터의 역할

- ✓ 어댑터는 출력할 데이터와 이 데이터가 올라가는 뷰를 매칭 시키는 역할을 합니다.
- ✓ 동적인 데이터의 표현을 위해서는 어댑터를 구현해야 합니다.
- ✓ API로 제공되는 어댑터의 활용뿐만 아니라 BaseAdapter 클래스를 상속하여 상황에 맞는 어댑터를 정의할 수 있습니다.

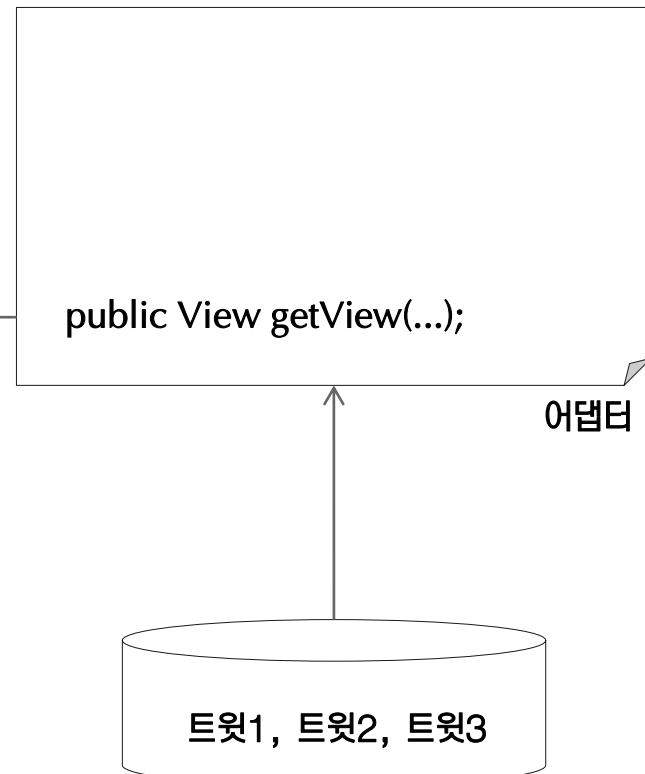
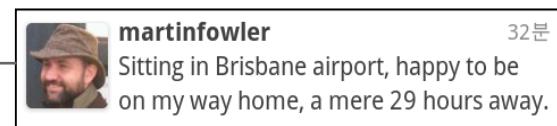


## 3.4 리스트뷰와 어댑터 (3/6) – 리스트 뷰

- ✓ 리스트뷰(ListView)는 가장 일반적으로 사용되는 어댑터뷰입니다.
- ✓ 리스트뷰는 리스트의 요소들을 수직 방향으로 정렬하여 표현합니다.
- ✓ ListActivity는 ListView를 포함하고 있는 액티비티입니다.



리스트뷰

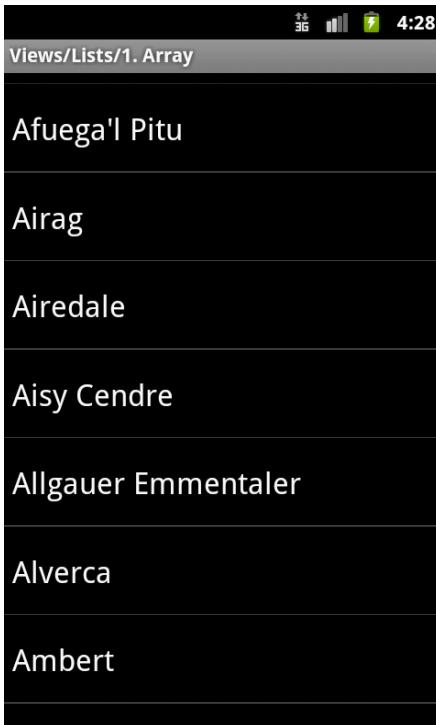


데이터

## 3.4 리스트뷰와 어댑터 (4/6) – 리스트 뷰 구현(1/2)

✓ 리스트 뷰의 속성을 활용하면 리스트 요소를 다양한 방식으로 표현할 수 있습니다.

- divider : 요소 아이템간의 구분을 위해 색상이나 이미지를 지정할 수 있습니다.
- choiceMode : 선택 불가, 단일 선택, 다중 선택을 지정할 수 있습니다.
- entries : 리스트에 표현할 요소가 정적인 배열 요소라면 array 리소스를 참조할 수 있습니다.



list.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView android:id="@+id>List"
        android:layout_width="match_parent "
        android:layout_height="match_parent "/>

</LinearLayout>
```

## 3.4 리스트뷰와 어댑터 (5/6) – 리스트 뷰 구현(2/2)

✓ 리스트 뷰를 포함하고 있는 리스트 액티비티 클래스 내에서 구현합니다.

- 리스트 뷰 구성을 위한 데이터를 조회합니다.
- 리스트 뷰의 아이템 요소 구성을 위해서 어댑터를 사용해야 합니다.
- 조회된 데이터에 대한 어댑터를 생성한 후 리스트 뷰에 어댑터를 설정합니다.

```
public class ListViewActivity extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.list);  
  
        List<String> countries= findCountries(); //데이터 조회  
        ArrayAdapter<String> adapter //어댑터 생성  
            = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, countries);  
  
        ListView listView = (ListView)findViewById(R.id.list);  
        listView.setAdapter(adapter); //리스트 뷰에 어댑터 설정  
    }  
  
    private List<String> findCountries() {  
        List<String> names = new ArrayList<String>();  
        names.add("Afarica");  
        names.add("Australia");  
  
        return names;  
    }  
}
```

## 3.4 리스트뷰와 어댑터 [6/6] – 어댑터 뷰의 퍼포먼스

- ✓ 어댑터 뷰를 새롭게 정의할 때는 표현 성능에 대해 고려해야 합니다.
- ✓ 사용자 정의 어댑터의 성능은 getView() 메소드의 재정의에서 결정합니다.
- ✓ 일반적으로 사용되는 getView() 메소드의 재정의 방식은 Recycling 방식과 ViewHolder 방식이 있습니다.

Recycling 방식

```
public View getView(int position,
    View convertView, ViewGroup parent) {

    if (convertView == null) {
        convertView =
            viewInflater.inflate(R.layout.myitem, null);
    }

    TextView textView =
        (TextView) convertView.findViewById(R.id.text);
    textView.setText(myData.get(position));

    return convertView;
}
```

ViewHolder 방식

```
static class ViewHolder {
    TextView text;
}

public View getView(int position,
    View convertView, ViewGroup parent) {
    ViewHolder holder;

    if (convertView == null) {
        convertView = viewInflater.inflate(R.layout.myitem, null);
        holder = new ViewHolder();
        holder.text =
            (TextView) convertView.findViewById(R.id.text);
        convertView.setTag(holder);
    } else {
        holder = (ViewHolder) convertView.getTag();
    }
    holder.text.setText(myData.get(position));

    return convertView;
}
```

## 3.5 프래그먼트 (1/2) – 개요

- ✓ 프래그먼트는 액티비티와 마찬가지로 사용자 화면을 구성할 때 사용합니다.
- ✓ Android 3.0(Honeycomb) 버전에서 태블릿과 같은 큰 해상도의 화면 구성을 위해 만들어 졌습니다.
- ✓ Android 4.0 부터는 태블릿과 스마트폰에 구분 없이 동일하게 적용되어 사용되고 있습니다.

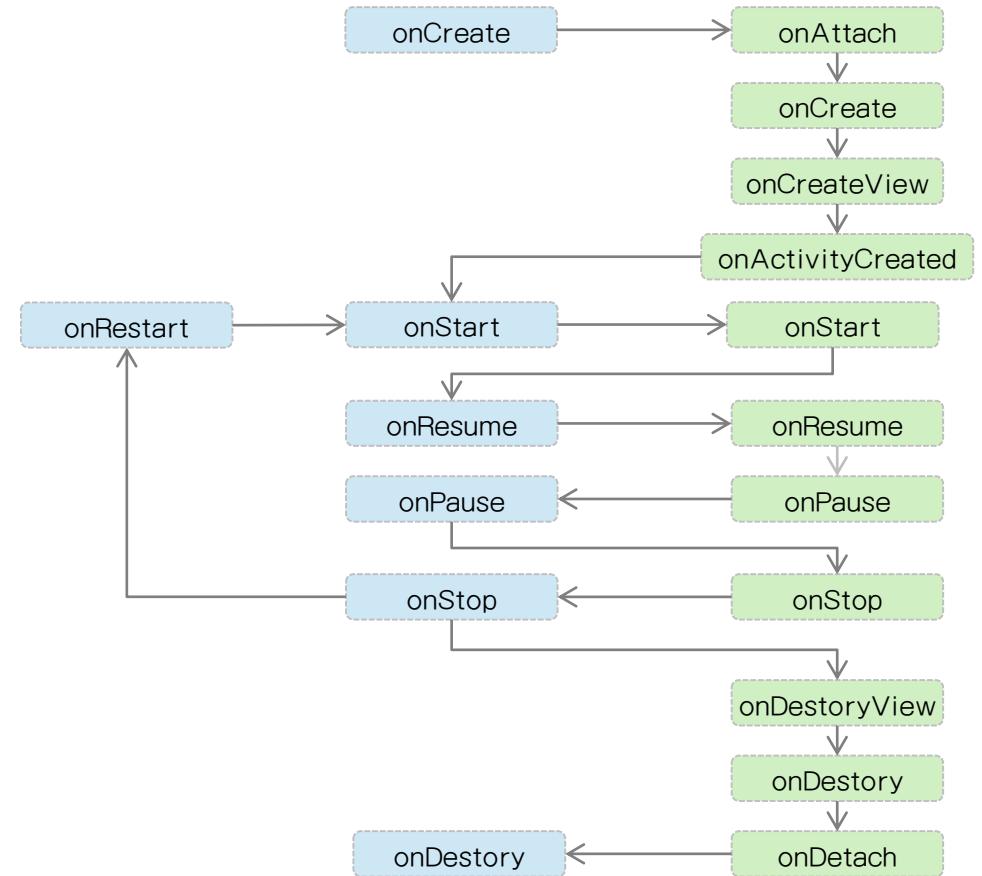


## 3.5 프래그먼트 (2/2) – 생명주기(Life-cycle)

- ✓ 프래그먼트는 액티비티에 종속적입니다.
- ✓ 프래그먼트는 액티비티 처럼 생명주기를 갖고 있으며 다수의 콜백 메소드를 정의하고 있습니다.
- ✓ 프래그먼트의 생명주기는 액티비티의 생명주기와 밀접하게 연관되어 있습니다.

메소드명	설명
onAttach()	프래그먼트가 액티비티와 연동될 때 호출됩니다.
onCreate()	프래그먼트가 생성될 때 호출됩니다.
onCreateView()	프래그먼트에 표시된 뷰를 생성하기 위해 호출됩니다.
onActivityCreated()	액티비티의 onCreate() 메소드가 종료되기 직전에 호출됩니다.
onStart()	프래그먼트가 화면에 나타날 때 호출됩니다.
onResume()	액티비티의 onResume() 메소드가 호출되고 사용자와 상호 작용이 가능할 때 호출됩니다.
onPause()	사용자와 상호 작용이 불가할 때 호출됩니다.
onStop()	프래그먼트의 동작이 중지 될 때 호출됩니다.
onDestoryView()	프래그먼트에 표시된 뷰를 제거할 때 호출됩니다.
onDestory()	프래그먼트가 종료될 때 호출됩니다.
onDetach()	프래그먼트를 종료하고 액티비티와의 연동을 끝낼 때 호출됩니다.

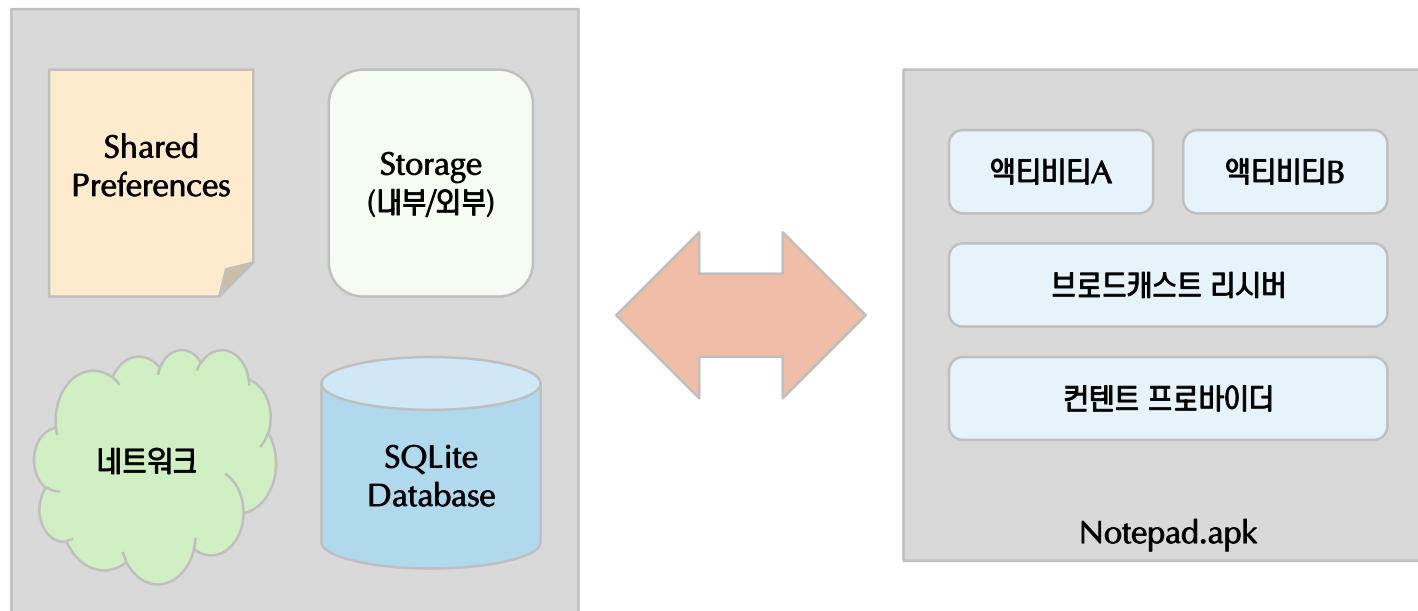
출처 : <https://developer.android.com/guide/components/fragments.html>



## 3.6 내부 데이터 관리 (1/12) – 개요

✓ 안드로이드는 내부 데이터 관리를 위해 다음과 같은 저장 방법을 제공합니다.

- SQLite를 이용한 경량의 데이터베이스
- Preference를 이용한 데이터 저장
- 제한적 사용이 가능한 파일 I/O
- 네트워크를 통한 데이터 관리



저장 장소 → /data/data/패키지이름

## 3.6 내부 데이터 관리 (2/12) – SQLite 개요

- ✓ 안드로이드는 조직화된 대량의 데이터를 다루기 위해 SQLite 내장하고 있습니다.
- ✓ 안정적이고 용량이 작아 소규모 데이터베이스에 적합하며, 단일 사용자만 지원합니다.
- ✓ 데이터 저장소가 단순한 파일이므로 별도의 서버가 필요 없습니다.
- ✓ 파일 저장경로 : /data/data/[패키지 명]/databases/[데이터베이스 명]



## 3.6 내부 데이터 관리 (3/12) – SQLiteOpenHelper 클래스

- ✓ SQLiteOpenHelper 클래스는 데이터베이스 작업을 수행하기 위해 DB 생성 및 연결을 지원하는 Helper 클래스입니다.
- ✓ 이 클래스는 데이터베이스와 테이블의 생성을 돕습니다.
- ✓ 그리고 데이터베이스를 여는 것과 기존에 데이터베이스와 테이블의 구조가 변경 되었다면 이를 갱신하는 것을 돕습니다.

`SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)`

- context : DB를 생성하는 컨텍스트
- name : DB 파일의 이름
- factory : 커스텀 커서를 사용할 때 지정함. 표준 커서를 사용할 경우는 null 지정
- version : DB 버전 생성 및 업데이트할 때 사용

메소드명	설명
onCreate()	DB가 처음 만들어질 때 호출된다. 테이블 생성문을 실행하여 테이블을 만듭니다.
onUpgrade()	DB를 업그레이드할 때 호출된다. 기존 테이블을 삭제하고 새로 만들거나 ALTER TABLE로 스키마를 수정합니다.
onOpen()	DB를 열 때 호출되는 메소드입니다.
getReadableDatabase()	읽기만 가능한 형태로 DB를 엽니다. DB가 없으면 onCreate가 호출되며 버전이 변경되면 onUpgrade가 호출됩니다.
getWritableDatabase()	읽고와 쓰기가 가능한 형태로 DB를 엽니다.
close()	DB를 닫습니다.

## 3.6 내부 데이터 관리 (4/12) – SQLite ContentValues 클래스

✓ ContentValues 는 데이터베이스 테이블에 기록할 데이터에 대응하는 객체입니다.

- 레코드 하나는 임의의 타입을 가지는 필드들의 집합이므로 테이블마다 형태가 다릅니다.
- 그래서 빈 객체를 만든 후 put 메소드를 호출하여 필드/값 쌍을 여러 개 저장합니다.
- 모든 기본 타입에 대해 오버로딩 되어 있으므로 필드의 타입에 맞는 메소드를 호출합니다.

### - 사용 예

```
ContentValues values = new ContentValues();
values.put("company", "(주) 넥스트리소프트"); // values.put(컬럼명, 문자열 데이터);
values.put("establishedYear", 15);           // values.put(컬럼명, 숫자형 데이터);
...
someDB.insert("data", null, values);
```

메소드명	설명
put (String key, Integer value)	
put (String key, String value)	
put (String key, Boolean Value)	
...	ContentValues 클래스 put() 메소드의 인자(Arguments)는 컬럼명과 해당 컬럼에 입력할 데이터로 선언합니다. 그 외 여러 타입에 대한 데이터 입력을 위한 put() 메소드가 오버로드되어 있습니다.

## 3.6 내부 데이터 관리 [5/12] – SQLiteDatabase API [1/3]

- ✓ 데이터베이스에서 레코드를 조회하여 데이터를 가져오는 것은 쿼리(Query, 질의)를 통해 이루어 집니다.
- ✓ SQLiteDatabase 클래스는 레코드 조회를 위해 query() 메소드를 제공하며, 결과는 Cursor 객체로 리턴합니다.
- ✓ query() 메소드의 다양한 매개변수를 이용해 여러 조건의 쿼리를 보낼 수 있으며 그 결과를 받을 수 있습니다.

```
public Cursor query (String table, String[] columns, String selection, String[] selectionArgs,  
String groupBy, String having, String orderBy, String limit)
```

인자 (Args)	설명
table	쿼리를 수행할 테이블 명
columns	자료를 조회할 컬럼명, null 입력시 모든 컬럼 반환 ( null 입력시 select * from AAA 에서 * 와 동일)
selection	SQL where 구분에 해당하는 조건과 값을 입력 조건이 많을 경우, 조건값을 ?로 대체 후 selectionArgs에서 값을 지정 가능
selectionArgs	selection에서 ?로 지정했던 조건의 값
groupBy	SQL group by 구문에 해당
having	groupBy를 지정했을 경우 조건을 입력
orderBy	결과값을 정렬방식 지정, null 입력시 주요 키를 기준으로 오름차순 정렬
limit	조회 결과의 개수 지정

## 3.6 내부 데이터 관리 [6/12] – SQLiteDatabase API [2/3]

✓ SQLiteDatabase 클래스는 데이터베이스의 레코드를 생성, 갱신, 삭제하기 위한 메소드를 제공합니다.

- insert 메소드의 반환 값은 신규로 생성된 Row의 ID입니다.
- update 및 delete 메소드의 반환 값은 갱신, 삭제된 Row 개수입니다.

```
public long insert(String table, String nullColumnHack, ContentValues values);  
  
public int update(String table, ContentValues values, String whereClause, String[] whereArgs);  
  
public int delete(String table, String whereClause, String[] whereArgs);
```

인자 (Args)	설명
table	쿼리를 수행할 테이블 명
nullColumnHack	row 자체가 null일 경우 대체되는 입력값
values	생성된 데이터 맵
whereClauses	조건문
whereArgs	조건문에서 사용되는 ?에 대체될 입력값

## 3.6 내부 데이터 관리 (7/12) – SQLiteDatabase API (3/3)

- ✓ 그 외 execSQL() 메소드는 select 문을 제외한 대부분의 SQL을 직접 실행합니다.
- ✓ rawQuery() 메소드는 select 문의 SQL을 실행합니다.
  - sql은 jdbc의 preparedStatement 형식으로 작성되며, SQL에서 ?에 selectionArgs 값으로 대체합니다.
- ✓ close() 메소드는 현재 열려 있는 데이터베이스를 닫습니다. 즉, 접속한 데이터베이스 연결을 종료합니다.

```
public void execSQL(String sql);  
  
public Cursor rawQuery(String sql, String[] selectionArgs);  
  
public void close();
```

### - 사용 예

- insert : execSQL("insert into 테이블명(컬럼, ...) values (데이터, ...);");
- update : execSQL("update 테이블명 set 컬럼=변경값, ... where 조건 ;");
- delete : execSQL("delete from 테이블명 where 조건;");

## 3.6 내부 데이터 관리 (8/12) – SQLiteOpenHelper 구현 (1/2)

```
public class BizCardDBManager extends SQLiteOpenHelper {

    private static BizCardDBManager instance;

    private static final String DB_BIZCARD      = "bizcard.db";
    private static final String TABLE_BIZCARD   = "bizcard";
    private static final int   DB_VERSION       = 1;

    private static final String TABLE_CREATE_SQL = "CREATE TABLE IF NOT EXISTS "
        + TABLE_BIZCARD + "(_id INTEGER PRIMARY KEY AUTOINCREMENT, "
        + "name TEXT NOT NULL, "
        + "company TEXT, "
        + "phone_no TEXT, "
        + "email TEXT)";

    private static final String TABLE_DROP_SQL = "DROP TABLE IF EXISTS " + TABLE_BIZCARD;

    public static BizCardDBManager getInstance(Context context) {
        //
        if (instance == null) {
            instance = new BizCardDBManager(context);
        }
        return instance;
    }

    private BizCardDBManager(Context context) {
        super(context, DB_BIZCARD, null, DB_VERSION);
    }
}
```

## 3.6 내부 데이터 관리 (9/12) – SQLiteOpenHelper 구현 (2/2)

```
@Override  
public void onCreate(SQLiteDatabase db) {  
    db.execSQL(TABLE_CREATE_SQL);  
}  
  
@Override  
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
    if (newVersion > oldVersion) {  
        db.execSQL(TABLE_DROP_SQL);  
    }  
}  
  
public long insert(ContentValues addRowValues) {  
    return getWritableDatabase().insert(TABLE_BIZCARD, null, addRowValues);  
}  
  
public Cursor query(String[] columns, String selection,  
    String[] selectionArgs, String groupBy, String having, String orderBy) {  
    return getReadableDatabase().query(TABLE_BIZCARD, columns, selection, selectionArgs, groupBy, having, orderBy);  
}  
  
public int update(ContentValues values, String whereClause, String[] whereArgs) {  
    return getWritableDatabase().update(TABLE_BIZCARD, values, whereClause, whereArgs);  
}  
  
public int delete(String whereClause, String[] whereArgs) {  
    return getWritableDatabase().delete(TABLE_BIZCARD, whereClause, whereArgs);  
}
```

## 3.6 내부 데이터 관리 (10/12) – Shared Preferences (1/3)

- ✓ 애플리케이션에서 환경 설정과 같이 간단한 설정값을 저장할 때 사용합니다.
- ✓ XML형태로 저장하며, 각 데이터는 key, value 형태로 저장합니다.
- ✓ 저장 경로는 “data/data/[패키지 명]/shared\_prefs/[파일명]” 입니다.
- ✓ 저장할 수 있는 데이터 유형은 다음과 같습니다.
  - Boolean
  - Integer
  - Float
  - Long
  - String

## 3.6 내부 데이터 관리 (11/12) – SharedPreferences (2/3)

- ✓ 하나의 액티비티에서만 사용하는 SharedPreferences의 생성 파일은 액티비티 명과 동일합니다.
- ✓ 특정 이름을 갖는 SharedPreferences는 애플리케이션 전체에 사용되며, 주로 설정 값 관리하는 경우에 사용합니다.
- ✓ 환경설정 액티비티에서 설정한 값이 저장된 SharedPreferences의 데이터에 접근 시 사용합니다.

```
public SharedPreferences getPreferences(int mode); // 하나의 액티비티에서 사용
```

```
public SharedPreferences getSharedPreferences(String name, int mode); // 이름을 통한 전체 애플리케이션 사용
```

모드 (Mode)	설명
MODE_PRIVATE	자기 app 내에서 사용 할 때 모드이며, 기본값이 0 입니다.
MODE_WORLD_READABLE	다른 액티비티에서 읽기 가능한 모드입니다.
MODE_WORLD_WRITEABLE	다른 액티비티에서 쓰기 가능한 모드입니다.
MODE_MULTI_PROCESS	다른 액티비티에서 읽기, 쓰기 모두 가능한 모드입니다.

```
// 선언  
PreferenceManager.getDefaultSharedPreferences(Context context);
```

## 3.6 내부 데이터 관리 (12/12) – SharedPreferences (3/3)

✓ 데이터 기록 및 조회는 putXXX() 및 getXXX() 메소드를 통해서 사용합니다.

- putXXX() 메소드는 SharedPreferences 객체 내 Editor 객체를 가져온 후 Editor의 getXXX() 메소드를 통해서 기록합니다.
- 저장 데이터 기록을 마친 뒤에는 commit() 메소드를 호출해야 변경사항을 완전히 저장합니다.
- getXXX() 메소드는 SharedPreferences 객체에서 직접 호출하여 데이터를 조회합니다.

데이터 기록

```
public void putString(key, value);
public void putBoolean(key, value);
public void putFloat(key, value);
public void putInt(key, value);
public void putLong(key, value);
```

데이터 조회

```
public String getString(key, value);
public boolean getBoolean(key, value);
public float getFloat(key, value);
public int getInt(key, value);
public long getLong(key, value);
public Map<String, ?> getAll();
```

- 사용 예

```
SharedPreferences sp =
    getSharedPreferences("myandoird_shared",
        MODE_PRIVATE);
SharedPreferences.Editor editor = sp.edit();
editor.putString("userId", userId);
editor.commit();
```

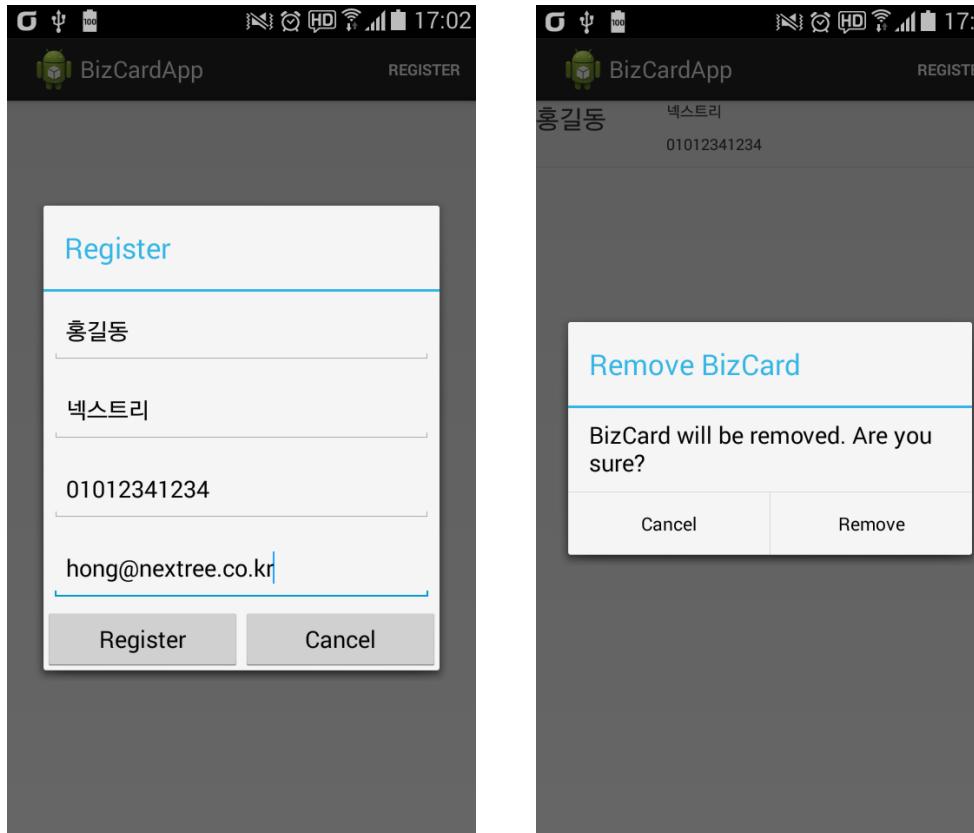
- 사용 예

```
SharedPreferences sp =
    getSharedPreferences("myandoird_shared",
        MODE_PRIVATE);

String value = sp.getString(key, null);
```

## 3.7 다이얼로그 (1/7) – 개요 (1/2)

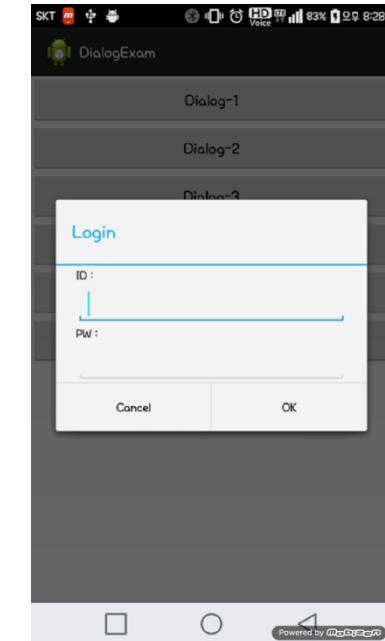
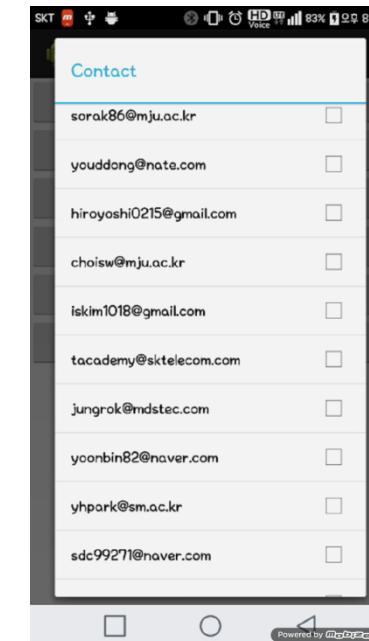
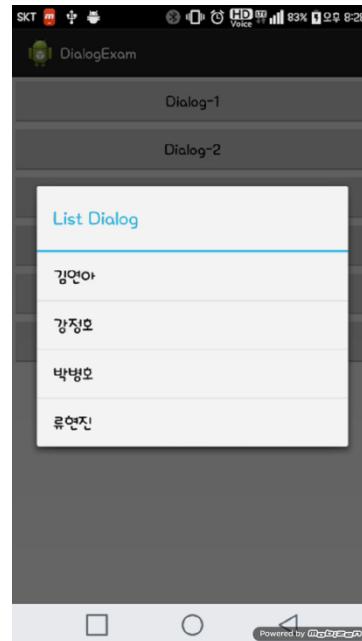
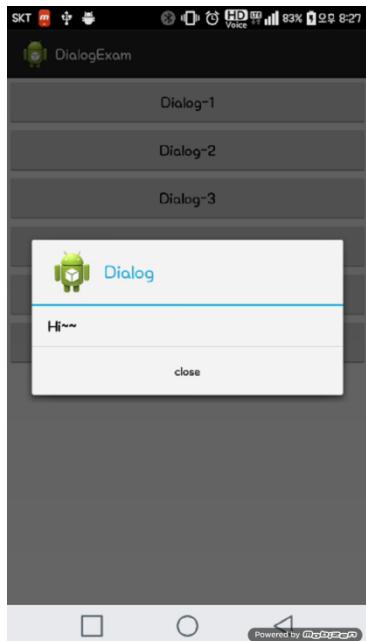
- ✓ 사용자와 상호작용을 위한 수단으로 다이얼로그를 사용할 수 있습니다.
- ✓ 다이얼로그는 정보를 전달하고, 사용자의 선택 또는 입력을 처리합니다.
- ✓ 대표적인 클래스는 AlertDialog로 문자열 메시지, 타이틀, 아이콘, 버튼을 사용할 수 있습니다.
- ✓ 간단한 설정을 통해 기존의 Activity 를 다이얼로그 형태로 띄울 수 있습니다.



## 3.7 다이얼로그 (2/7) – 개요 (2/2)

✓ 다이얼로그 클래스의 종류는 다음과 같습니다.

- AlertDialog : 가장 기본적인 다이얼 로그
- ProgressDialog : 작업의 진행률을 사용자에게 알려줄 때 사용
- DatePicker / TimePicker : 날짜 혹은 시간을 간편하게 선택할 수 있는 다이얼 로그
- ActivityDialog : 액티비티를 다이얼로그 처럼 표시



## 3.7 다이얼로그 (3/7) – AlertDialog (1/3)

- ✓ AlertDialog는 직접 생성할 수 없고 내부 클래스인 Builder를 이용해 생성합니다.
  - 즉, AlertDialog.Builder 클래스의 create() 또는 show() 메소드를 통해 AlertDialog 객체를 받을 수 있습니다.
- ✓ 아래는 AlertDialog.Builder 클래스의 AlertDialog 구성을 위한 주요 메소드입니다.

메소드	설명
public AlertDialog.Builder(Context context);	AlertDialog.Builder 클래스 생성자로서 객체를 생성합니다.
public AlertDialog.Builder setTitle(CharSequence title);	다이얼로그의 제목을 지정합니다.
public AlertDialog.Builder setTitle(int titleResourceld);	
public AlertDialog.Builder setIcon(int resourceld);	다이얼로그의 아이콘을 설정합니다.
public AlertDialog.Builder setMessage(CharSequence message);	다이얼로그의 메시지를 설정합니다.
public AlertDialog.Builder setMessage(int messageResourceld);	
public AlertDialog create();	다이얼로그를 생성합니다.
public AlertDialog show();	다이얼로그를 생성하고 화면에 보여 줍니다.

## 3.7 다이얼로그 (3/7) – AlertDialog (2/3)

✓ AlertDialog는 3개의 버튼을 지정하여 이벤트를 받을 수 있는 기본 구조가 반영되어 있습니다.

- 만약, 이벤트 리스너의 작업이 별도로 지정되어 있지 않으면 다이얼로그를 닫습니다.
- 다이얼로그에는 메시지 외에도 커스텀 뷰를 구성하여 사용자의 추가적인 입력을 처리할 수도 있습니다.

✓ 아래는 AlertDialog.Builder 클래스의 AlertDialog 이벤트 처리를 구성하는 주요 메소드입니다.

이벤트 유형	메소드
왼쪽 버튼 클릭	public void setPositiveButton(CharSequence text, DialogInterface.OnClickListener clickListener);
	public void setPositiveButton(int textResourceld, DialogInterface.OnClickListener clickListener);
가운데 버튼 클릭	public void setNeutralButton(CharSequence text, DialogInterface.OnClickListener clickListener);
	public void setNeutralButton(int textResourceld, DialogInterface.OnClickListener clickListener);
오른쪽 버튼 클릭	public void setNegativeButton(CharSequence text, DialogInterface.OnClickListener clickListener);
	public void setNegativeButton(int textResourceld, DialogInterface.OnClickListener clickListener);
메시지 하단 커스텀 뷰 추가	public void setView(View view);

## 3.7 다이얼로그 (5/7) – AlertDialog (3/3)

✓ 사용자의 입력을 필요로 할 경우 다음과 같은 다이얼로그를 직접 생성하여 입력을 처리합니다.

- 단, 안드로이드의 다이얼로그는 프로세스를 멈추고 응답을 대기하는 것이 아니므로, 사용자의 이벤트 처리는 버튼 클릭 이벤트 핸들러 내에서 처리해야 합니다.
- 이벤트 처리는 DialogInterface 클래스 내부에 정의된 OnClickListener를 구현하여 처리합니다.

### - 사용 예

```
new AlertDialog.Builder(this)
    .setTitle("명함 추가")
    .setView(view)
    .setPositiveButton("확인", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            EditText etName = (EditText) view.findViewById(R.id.ed_name);
            EditText etCompany = (EditText) view.findViewById(R.id.ed_company);

            BusinessCard card = new BusinessCard();
            card.setName(etName.getText().toString());
            card.setCompany(etCompany.getText().toString());
            ...

            cards.add(card);
        }
    }).show();
```

## 3.7 다이얼로그 (6/7) – ProgressDialog

- ✓ ProgressDialog는 작업의 진행률을 사용자에게 알려 줄 때 사용합니다.
- ✓ ProgressDialog에서 작업의 진행률은 표시되는 형태와 표시되지 않는 형태로 설정할 수 있습니다.
- ✓ 진행률의 범위는 0~10000 까지 지정할 수 있습니다.

작업의 진행률을 표시하는 형태

```
ProgressDialog pd = new ProgressDialog(this);  
  
// 진행률 표시 형태  
pd.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);  
  
pd.setTitle("Progress Dialog"); // 제목  
pd.setMessage("진행중입니다."); // 내용  
pd.setCancelable(true); // 설정시 백버튼으로 취소가 가능해진다.  
pd.setProgress(value); // 진행률 표시
```

작업의 진행률을 표시하지 않는 형태

```
ProgressDialog pd = new ProgressDialog(this);  
  
// 진행률 표시되지 않는 형태  
pd.setProgressStyle(ProgressDialog.STYLE_SPINNER);  
  
pd.setTitle("Progress Dialog"); // 제목  
pd.setMessage("진행중입니다."); // 내용  
pd.setCancelable(true); // 설정시 백버튼으로 취소가 가능해진다.
```

## 3.7 다이얼로그 (7/7) – DatePicker / TimePicker

✓ DatePicker 및 TimePicker 다이얼로그는 날짜 혹은 시간을 선택하게 할 수 있는 다이얼로그입니다.

- DatePicker 생성 : new DatePickerDialog(context, callback, year, month, day);
- TimePicker 생성 : new TimePickerDialog(context, callback, hour, minute, false);

DatePicker  
날짜 셋팅을 위한 리스너 구현

```
DatePickerDialog.OnDateSetListener dpCb= new DatePickerDialog.OnDateSetListener() { //CallBack
    @Override
    public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
        mTextView.setText(year + "년" + monthOfYear + "월" + dayOfMonth + "일");
    }
};
```

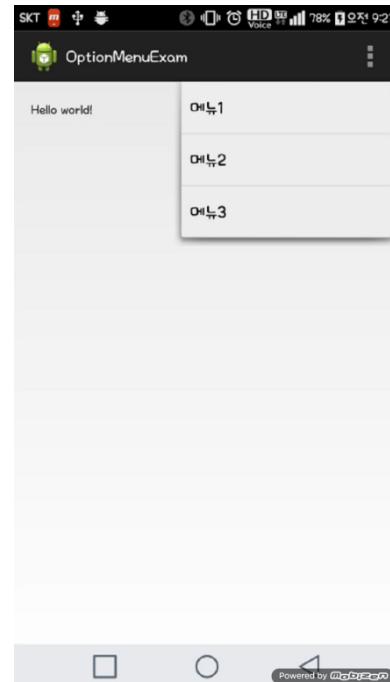
TimePicker  
시간 셋팅을 위한 리스너 구현

```
TimePickerDialog.OnTimeSetListener tpCb = new TimePickerDialog.OnTimeSetListener() { //CallBack
    @Override
    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        mTextView.setText(hourOfDay + "시" + minute + "분");
    }
};
```

## 3.8 옵션 메뉴와 컨텍스트 메뉴 (1/11) – 옵션 메뉴(1/7)

✓ 옵션 메뉴는 모바일 장치의 메뉴 버튼이나 액션 바의 메뉴 버튼을 선택했을 때 나타나는 메뉴 버튼들을 말합니다.

- 애플리케이션 화면상에 표시할 수 없는 여러 기능을 접근 하기 위한 방법으로 사용합니다.
- 하나의 메뉴는 한 단계의 서브 메뉴를 가질 수 있습니다.
- 메뉴에 단축키를 지정하여 해당 메뉴를 호출 할 수도 있습니다.



## 3.8 옵션 메뉴와 컨텍스트 메뉴 [2/11] – 옵션 메뉴 [2/7]

- ✓ 옵션 메뉴는 onCreateOptionsMenu() 메소드를 이용해 생성할 수 있습니다.
- ✓ 이 메소드는 안드로이드의 버전에 따라 호출 시점이 다릅니다.
  - Android 2.3 이하 버전의 경우 사용자가 처음 메뉴를 열었을 때 호출합니다.
  - Android 3.0 이상 버전의 경우 액티비티를 시작할 때 작업 모음의 표시를 위해 호출합니다.

```
// 메뉴 생성
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);

    menu.add(0, 0, Menu.NONE, "추가").setIcon(android.R.drawable.ic_menu_add);
    menu.add(0, 1, Menu.NONE, "설정").setIcon(android.R.drawable.ic_menu_preferences);

    return true; // 메뉴를 보이게 하려면 반드시 true 를 반환해야 한다.
}
```

## 3.8 옵션 메뉴와 컨텍스트 메뉴 [3/11] – 옵션 메뉴 [3/7]

- ✓ 옵션 메뉴의 add() 메소드는 메뉴 항목을 추가할 때 호출하는 메소드입니다.
  - 메뉴 항목을 추가 시 메뉴 그룹 및 메뉴 고유ID 부여, 순서 등을 정의해야 합니다.
  - 다음에서 설명하는 매개 변수를 통해 메뉴들을 그룹화하거나 메뉴의 이름을 설정합니다.

//아이콘을 사용하는 메뉴

```
menu.add(groupId, menuId, int order, int titleRes).setIcon(int resourceId)  
menu.add(groupId, int menuId, int order, CharSequence title).setIcon(int resourceId)
```

//메뉴 선택시 다른 액티비티 직접 실행

```
menu.add(groupId, int menuId, int order, int resourceId).setIcon(int resourceId)  
.setIntent(new Intent(this, OtherActivity.class));
```

인자 (Args)	설명
groupId	메뉴의 그룹을 구성할 때 사용하며 지정하지 않을 경우는 MENU.NONE(0)을 지정합니다.
menuId	해당 메뉴의 고유한 ID이며 이 ID를 통해 선택된 메뉴를 선별합니다.
order	메뉴가 표현되는 순서를 지정할 때 사용하며 MENU.NONE로 지정할 경우 추가한 순서로 표시됩니다.
titleRes 또는 title	메뉴 항목에 나타나는 이름을 지정할 때 사용합니다.

## 3.8 옵션 메뉴와 컨텍스트 메뉴 (4/11) – 옵션 메뉴 (4/7)

✓ 옵션 메뉴의 이벤트 처리는 다음과 같습니다.

- 옵션 메뉴의 이벤트는 Activity.onOptionsItemSelectedSelected () 메소드에서 처리합니다.
- 옵션 메뉴가 선택되면 onOptionsItemSelected() 메소드가 콜백 되며 이때 선택된 메뉴 항목을 선별하여 이벤트 처리합니다.

```
// 메뉴 선택시 처리 작업
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);
    switch (item.getItemId()) {
        case MENU_ADD:
            // 추가 버튼이 선택되었을 경우 이벤트 처리
            break;
        case SUBMENU_PROFILE:
            // 설정의 서브메뉴, 개인정보 메뉴 선택시 처리
            break;
        case SUBMENU_HELP:
            // 설정의 서브메뉴, 도움말 메뉴 선택시 처리
            break;
    }
    return true;
}
```

## 3.8 옵션 메뉴와 컨텍스트 메뉴 (5/11) – 옵션 메뉴 (5/7)

✓ 옵션 메뉴는 하위 메뉴를 가질 수 있습니다.

- 하위 메뉴는 addSubMenu() 메소드를 이용해 생성하며 이 메소드는 SubMenu 객체를 반환합니다.
- 옵션 메뉴의 하위 메뉴는 리스트를 이용한 다이얼로그와 동일한 구성을 갖고 있습니다.

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    menu.add(0, 0, Menu.NONE, "추가").setIcon(android.R.drawable.ic_menu_add);  
    SubMenu subMenu =  
        menu.addSubMenu(0, 1, Menu.NONE, "설정").setIcon(android.R.drawable.ic_menu_preferences);  
  
    subMenu.add(0, 0, Menu.NONE, "개인설정");  
    subMenu.add(0, 1, Menu.NONE, "도움말");  
  
    return true;  
}
```

## 3.8 옵션 메뉴와 컨텍스트 메뉴 [6/11] – 옵션 메뉴 [6/7]

✓ 옵션 메뉴는 XML을 통해서도 생성 가능합니다.

- 메뉴의 경우 반복적으로 사용될 경우가 존재하기 때문에 xml을 통해 선언적 구성이 가능합니다.
- menu.xml 을 작성하여 다음과 같이 Menu를 구성할 수 있습니다.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/menu_add" android:title="추가"
          android:icon="@android:drawable/ic_menu_add" />

    <item android:id="@+id/menu_preference" android:title="설정"
          android:icon="@android:drawable/ic_menu_preferences">

        <menu>
            <item android:id="@+id_submenu_profile" android:title="개인정보" />
            <item android:id="@+id_submenu_help" android:title="도움말" />
        </menu>
    </item>
</menu>
```

## 3.8 옵션 메뉴와 컨텍스트 메뉴 [8/11] – 옵션 메뉴 (7/7)

✓ XML을 통한 옵션 메뉴의 이벤트는 View의 onClick 속성을 이용하는 방식과 동일합니다.

- item 요소에 onClick 속성의 값을 지정합니다.(속성값 : 이벤트 호출 메소드명)
- 옵션 메뉴가 적용된 액티비티 클래스 내부에서 item의 속성값에 지정된 메소드 명을 이용해 이벤트 메소드를 정의 합니다.

XML 정의

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android=
    "http://schemas.android.com/apk/res/android">

    <item android:id="@+id/about"
        android:title="프로그램소개"
        android:onClick="mOnClick" />

    <item android:id="@+id/help"
        android:title="도움말"
        android:onClick="mOnClick" />

</menu>
```

이벤트 메소드 구현

```
public void mOnClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.about:
            Toast.makeText(this,
                "이벤트처리", Toast.LENGTH_SHORT).show();
            break;
        case R.id.help:
            Toast.makeText(this,
                "이벤트처리", Toast.LENGTH_SHORT).show();
            break;
        default:
            break;
    }
}
```

## 3.8 옵션 메뉴와 컨텍스트 메뉴 [9/11] – 컨텍스트 메뉴 (1/3)

✓ 컨텍스트 메뉴는 특정 View 요소를 3초 이상 누르고 있을 경우 나타나는 메뉴입니다.

- 선택된 항목에 대해 추가적인 동작을 제시하는 경우 사용합니다.
- 컨텍스트 메뉴를 생성한 이후에는 컨텍스트 메뉴를 제공할 View에 등록하는 과정이 필요 합니다.
- 컨텍스트 메뉴의 등록은 Activity.registerForContextMenu(view) 메소드를 이용합니다.

```
public class MainActivity extends Activity {  
    private static final int MENU_EDIT = 1;  
    private static final int MENU_REMOVE = 2;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        setContentView(R.layout.activity_main);  
        mText = (TextView) findViewById(R.id.text);  
        registerForContextMenu(mText);  
    }  
}
```

## 3.8 옵션 메뉴와 컨텍스트 메뉴 [10/11] – 컨텍스트 메뉴 [2/3]

- ✓ 컨텍스트 메뉴 생성은 Activity.onCreateContextMenu() 메소드를 이용하여 생성합니다.
- ✓ 컨텍스트 메뉴는 모든 View에 적용할 수 있습니다.
- ✓ 컨텍스트 메뉴도 옵션 메뉴와 마찬가지로 XML을 이용한 등록이 가능합니다.

```
public void onCreateContextMenu(ContextMenu menu, View view, ContextMenuItemInfo menuInfo) {  
    super.onCreateContextMenu(menu, v, menuInfo);  
  
    // 선택된 view에 따라 다른 종류의 컨텍스트 메뉴를 제공한다.  
    switch (view.getId()) {  
        case R.id.text:  
            menu.add(0, MENU_EDIT, Menu.NONE, "수정");  
            menu.add(0, MENU_DELETE, Menu.NONE, "삭제");  
            break;  
    }  
}
```

## 3.8 옵션 메뉴와 컨텍스트 메뉴 [11/11] – 컨텍스트 메뉴 [3/3]

✓ 컨텍스트 메뉴의 이벤트는 Activity.onOptionsItemSelected() 메소드를 이용해 처리합니다.

- 컨텍스트 메뉴의 버튼이 선택되면 onOptionsItemSelected() 메소드를 다시 호출(callback)합니다.
- 이 메소드에서 전달 인자로 받는 item을 통해 이벤트를 구분하여 이벤트를 처리합니다.

```
// 메뉴 선택시 처리 작업
@Override
public boolean onOptionsItemSelected(MenuItem item) {

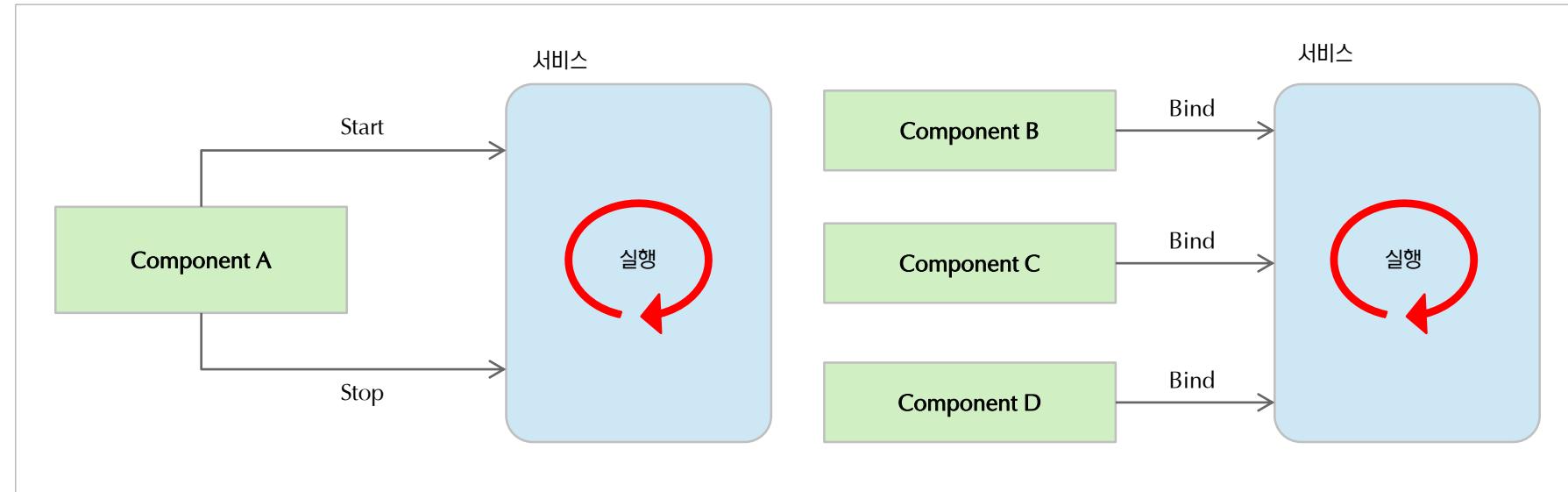
    switch (item.getItemId()) {
        case MENU_EDIT:
            //추가 버튼이 선택되었을 경우 이벤트 처리
            return true;
        case MENU_PREFERENCE :
            //설정 버튼이 선택되었을 경우 이벤트 처리
            return true;
    }

    return super.onOptionsItemSelected(item);
}
```

## 3.9 서비스 [1/9] – 개요

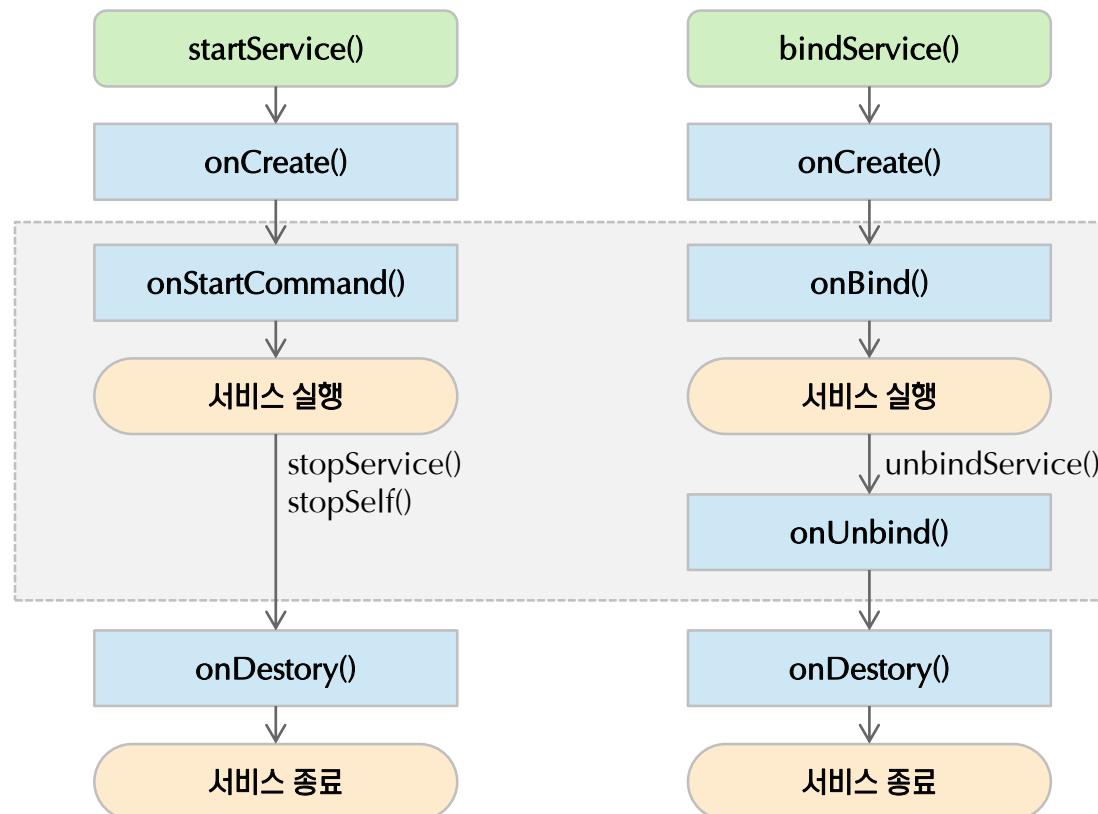
✓ 서비스는 사용자 UI를 갖지 않는 안드로이드 컴포넌트로서 다른 컴포넌트가 생성하고 실행합니다.

- 서비스를 생성하는 컴포넌트는 액티비티, 브로드캐스트 리시버 등이 있습니다.
- 서비스를 활용하면 액티비티가 화면에서 사라지거나, 비활성화되거나, 종료된 경우에도 지속적인 처리나 규칙적인 처리 혹은 이벤트 처리를 수행할 수 있습니다.
- 서비스는 동작 방식에 따라 시작 서비스, 바인드 서비스로 구분합니다.



## 3.9 서비스 [2/9] – 생명주기(Life Cycle)

- ✓ 서비스는 인텐트를 통해 동작하며 생명주기(Life-cycle)를 갖고 있습니다.
- ✓ startService() 메소드를 통해 서비스를 호출할 경우 서비스 객체를 생성합니다.
- ✓ onCreate() 메소드는 서비스 객체가 생성될 때 단 한번만 호출합니다.
- ✓ onBind() 메소드는 서비스 객체가 생성되어 이미 실행된 이후에 호출할 수 있습니다.



출처 : <https://developer.android.com/guide/components/services.html>

## 3.9 서비스 [3/9] – 구현 [1/4]

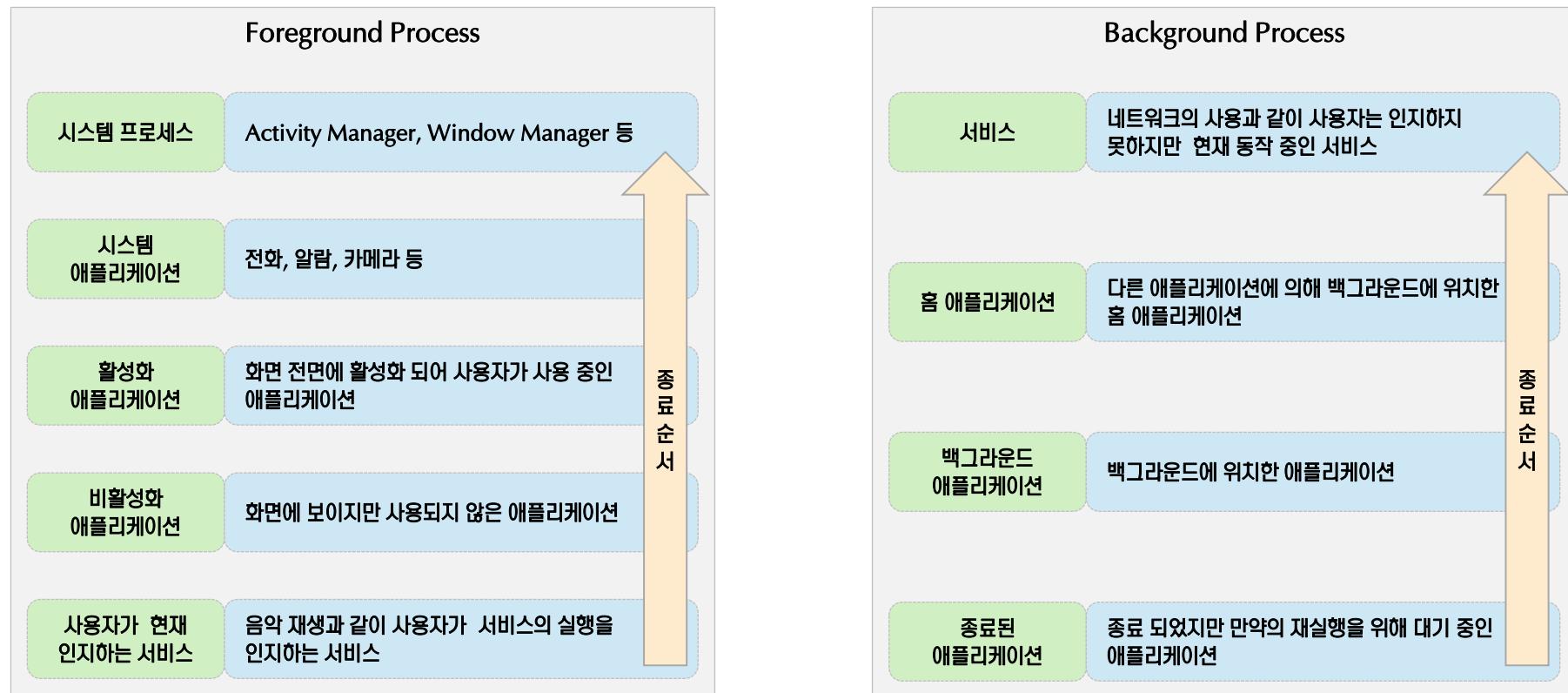
✓ 서비스는 Service 클래스를 상속하여 구현 합니다.

- onCreate() 메소드를 통해 서비스 생성 시 수행할 동작을 구현 합니다.
- onStartCommand() 메소드는 stopService 및 stopSelf를 통해 서비스가 종료되기 이전에 시스템이 서비스를 종료할 경우 서비스 재시작 방식을 설정할 수 있습니다.

```
public class MyService extends Service {  
  
    @Override  
    public void onCreate() {  
        // TODO : 서비스 생성시 수행할 동작.  
    }  
  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        // TODO : 처리를 수행할 백그라운드 스레드를 실행  
        return Service.START_STICKY;  
    }  
}
```

## 3.9 서비스 (4/9) – 구현 (2/4)

- ✓ 안드로이드 시스템은 메모리가 부족할 경우 Low Memory Killer를 이용해 프로세스를 종료합니다.
- ✓ Low Memory Killer는 각 프로세스의 종류에 따라 우선 순위에 따라 프로세스를 종료합니다.
- ✓ 프로세스의 우선순위는 크게 포그라운드(Foreground) 프로세스와 백그라운드(Background) 프로세스로 구분합니다.
- ✓ 메모리가 부족할 경우 백그라운드 프로세스들을 우선 종료합니다.



## 3.9 서비스 (5/9) – 구현 (3/4)

- ✓ Service.onCreate() 메소드는 서비스를 처음 생성하고 실행할 때만 호출합니다.
- ✓ 서비스가 다시 시작될 때에는 Service.onStartCommand() 메소드를 호출합니다.
- ✓ Service.onStartCommand() 메소드의 반환값을 통해 서비스를 다시 시작할 경우의 동작 방식을 설정할 수 있습니다.

```
@Override  
public int onStartCommand(Intent intent, int flags, int startId) {  
    ...  
    return Service.START_STICKY;  
}
```

속성 (Constnts)	설명
START_STICKY	가장 많이 사용되는 방식이며, 만약 메모리가 부족하여 Low Memory Killer에 의해 서비스가 종료 되었다면 이후에 메모리가 확보 되었을 경우 다시 실행 됩니다. 따라서, 서비스의 생존이 중요한 애플리케이션의 경우 아 값으로 반환하여야 합니다.
START_NOT_STICKY	특정 상황에 의해 서비스가 종료된 경우라 할지라도 다시 실행되지 않습니다. 서비스가 불필요하게 실행되는 것을 방지 할 수 있는 가장 안전한 방식입니다.
START_REDELIVER_INT	START_STICKY 방식과 동일한 방식입니다. 차이는 onStartCommand() 메소드가 다시 호출 될 때 START_STICKY 방식과 달리 파라미터로 전달되는 인텐트가 처음 호출되었을 때와 같이 전달됩니다. 즉, 서비스의 생존이 중요하고 서비스가 다시 실행 될 때 전달되는 인텐트의 전달 값이 중요하게 사용될 경우 이 값으로 반환 합니다.

출처 : <https://developer.android.com/guide/components/services.html>  
<https://developer.android.com/reference/android/app/Service.html>

## 3.9 서비스 [6/9] – 구현 [4/4]

- ✓ Service.onStartCommand() 메소드의 flags 매개 변수를 통해 서비스의 재실행 상태를 확인할 수 있습니다.
- ✓ 일반적으로 서비스가 종료되었거나 다시 실행 되는 경우에 이 flags의 값으로 0을 전달합니다.
- ✓ 서비스가 비 정상적으로 종료 되었다면 flags에 START\_FLAG\_REDELIVERY 값이 전달되며 이때 이전의 인텐트도 함께 전달 인자로 받습니다.

```
@Override  
public int onStartCommand(Intent intent, int flags, int startId) {  
    if ((flags & start_FLAG_RETRY) == 0) {  
        // TODO : 재시작 된경우에 원하는 작업을 수행  
    }  
    else {  
        // TODO : 처리를 수행할 백그라운드 스레드를 실행  
    }  
    return Service.START_STICKY;  
}
```

속성 (Constnts)	설명
START_FLAG_REDELIVERY	매개변수로 전달된 인텐트가 재전송된 인텐트임을 나타냅니다.
START_FLAG_RETRY	서비스가 비정상적인 종료 후 재시작 되었음을 나타냅니다.

출처 : <https://developer.android.com/reference/android/app/Service.html>

## 3.9 서비스 (7/9) – Service 등록

- ✓ 서비스는 안드로이드의 4대 주요 컴포넌트 중에 하나입니다.
- ✓ 서비스도 다른 주요 컴포넌트와 마찬가지로 AndroidManifest.xml에 등록해야 합니다.
- ✓ 서비스의 실행은 인텐트를 통해 이루어지며 인텐트 필터를 이용한 암묵적 호출이 가능합니다.

```
// Syntex
<service android:enabled=[ "true" | "false"]
          android:exported=[ "true" | "false"]
          android:icon="drawable resource"
          android:isolatedProcess=[ "true" | "false"]
          android:label="string resource"
          android:name="string"
          android:permission="string"
          android:process="string" >

</service>
```

```
// 사용 예
<service android:enabled="true" android:name=".MyService" />
```

출처 : <https://developer.android.com/guide/topics/manifest/service-element.html>

## 3.9 서비스 [8/9] – 서비스 시작 및 제어 [1/2]

✓ 서비스 시작 : startService()를 호출하여 서비스를 시작합니다.

- 암시적 : 브로드캐스트 리시버를 가진 서비스를 호출하는 방식입니다.
- 명시적 : 원하는 서비스 클래스를 이용한 서비스를 호출하는 방식입니다.
- 실행 권한이 없으면 SecurityException을 호출합니다.

```
// 암시적인 방법
```

```
Intent myIntent = new Intent("com.namoosori.myandroid.MyService");
startService(myIntent);
```

```
// 명시적인 방법
```

```
startService(new Intent(this, MyService.class));
```

## 3.9 서비스 [9/9] – 서비스 시작 및 제어 [2/2]

✓ 서비스 중지는 stopService()와 stopSelf() 메소드를 통해 할 수 있습니다.

- stopService() 메소드는 명시적으로 서비스 이름을 통해 중지합니다.
- stopSelf() 메소드는 서비스 스스로 종료할 때 사용합니다.

```
Intent intent = new Intent(this, MyService.class);
stopService(intent);
```

- 이미 실행중인 서비스의 startService()를 실행하면 해당 서비스의 onStartCommand() 핸들러가 실행되며, 중첩 호출되지 않습니다.
- 여러 번의 startService()를 호출한 경우에 관계 없이 한번의 stopService()를 호출하면 서비스가 종료됩니다.

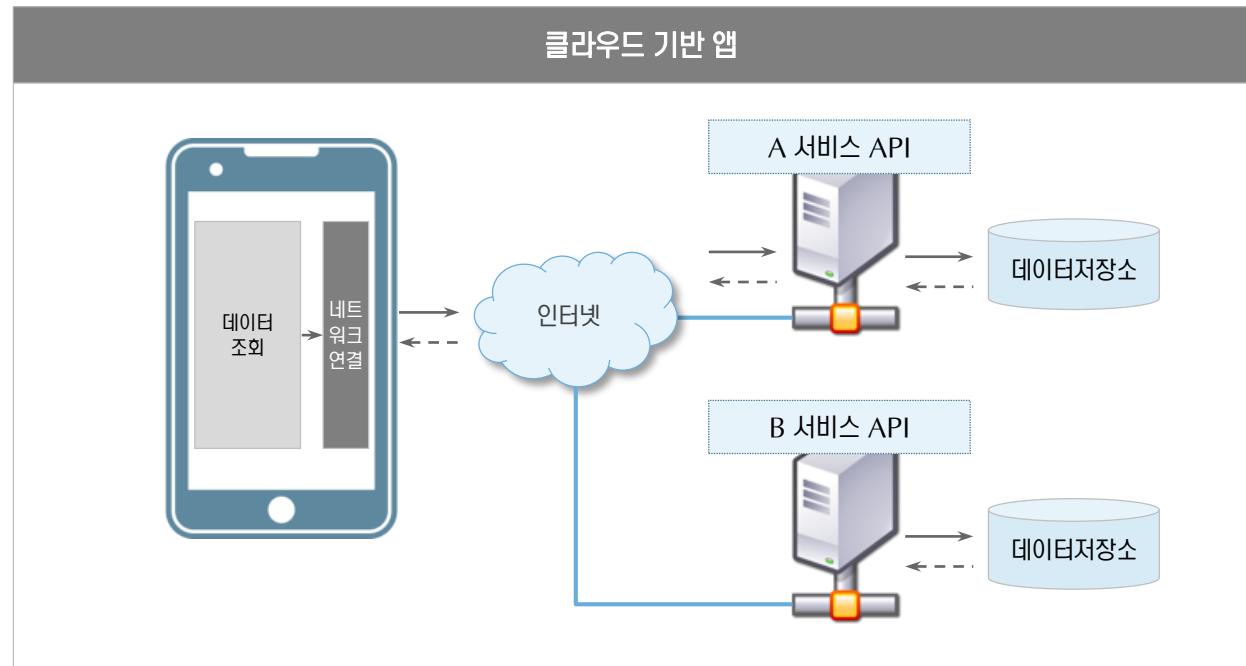


## 4. 클라우드 기반 애플리케이션 개발

- 
- 4.1 클라우드 기반 애플리케이션
  - 4.2 안드로이드 쓰레드 처리
  - 4.3 AsyncTask를 이용한 비동기 처리
  - 4.4 안드로이드 네트워크
  - 4.5 전문 변환

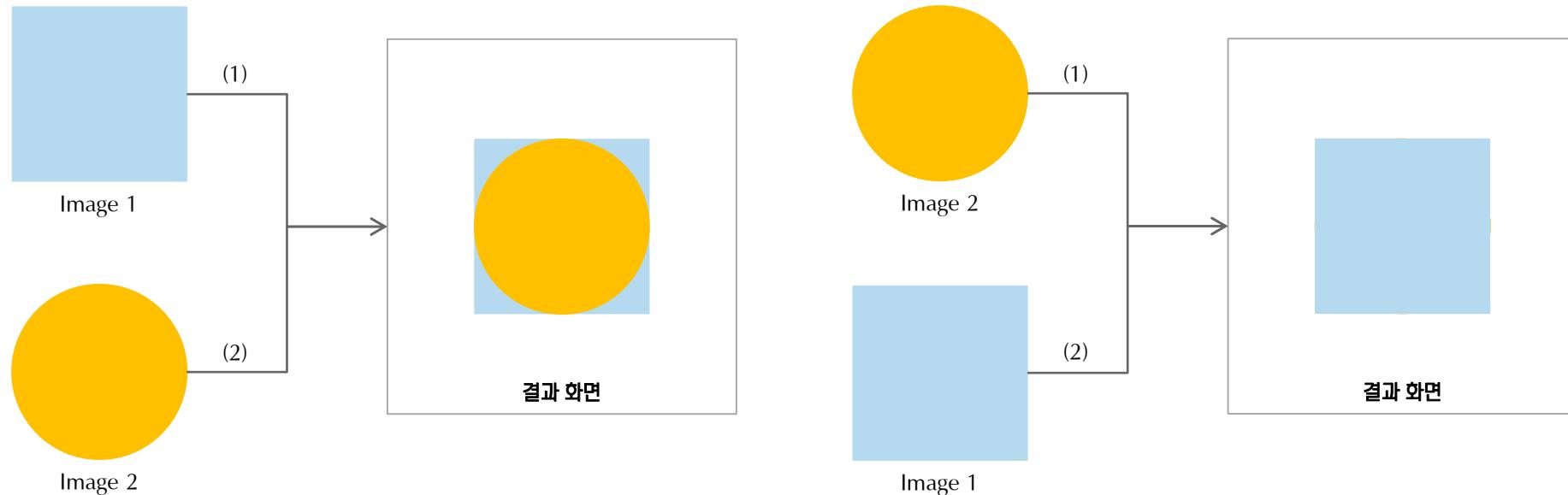
## 4.1 클라우드 기반 애플리케이션

- ✓ 클라우드 기반 애플리케이션은 독립 구동형 앱과는 달리 인터넷 상에 존재하는 서비스를 활용하여 구현하는 방식입니다.
- ✓ 클라우드 서버에 서비스 API를 배포하고, 애플리케이션에서는 HTTP 통신을 통해 서비스를 호출합니다.
- ✓ 이미지 업로드 및 이미지 조회는 이미지 관련 클라우드 서비스를 활용합니다.
- ✓ 본 장에서는 네트워크 통신을 처리하는 방법과 메시지 전문을 변환하는 과정을 살펴봅니다.



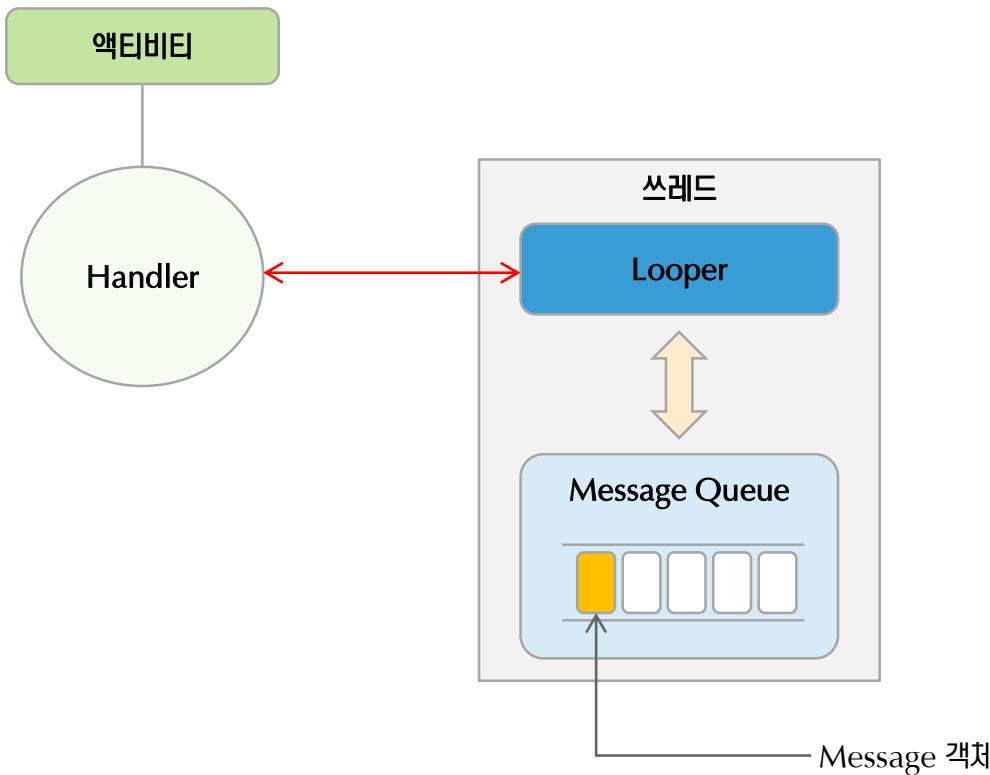
## 4.2 안드로이드 쓰레드 처리 (1/2) – 메인 쓰레드의 이해

- ✓ 안드로이드에는 메인 쓰레드, 바인더 쓰레드, 백그라운드 쓰레드 3가지 유형의 쓰레드가 있습니다.
- ✓ 메인 쓰레드는 UI 쓰레드라고도 하며 사용자 화면에 대한 모든 권한을 갖는 쓰레드입니다.
- ✓ 안드로이드는 메인 쓰레드 이외의 쓰레드가 UI를 변경하는 것을 허용하지 않습니다.



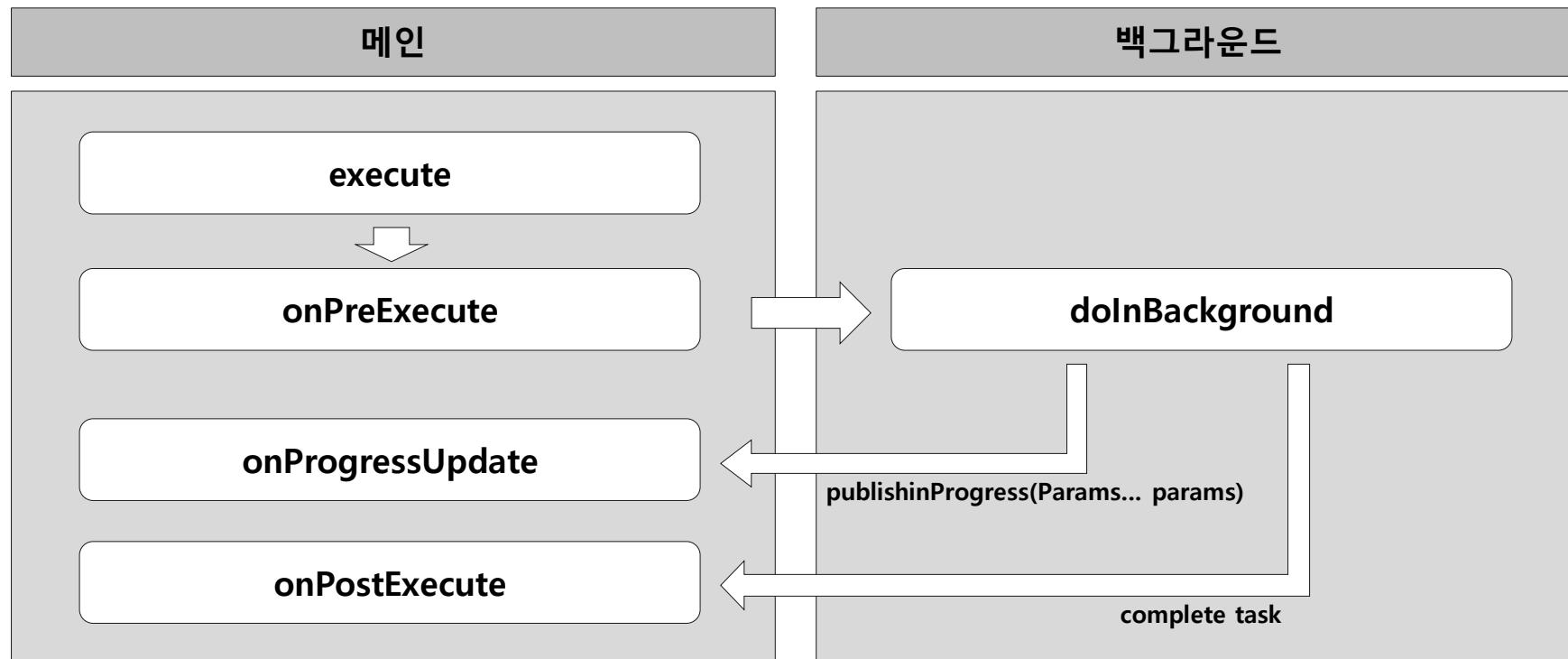
## 4.2 안드로이드 쓰레드 처리 (2/2) – Handler를 통한 쓰레드 처리

- ✓ Looper 객체는 메인 쓰레드가 처리해야 하는 일들을 반복적으로 수행하는 객체입니다.
- ✓ Looper 객체는 처리해야 하는 일들이 저장되어 있는 메시지큐(Message Queue)를 갖고 있습니다.
- ✓ 메시지 큐에 저장되는 하나의 작업은 Message 객체로 표현합니다.
- ✓ Handler 객체는 메시지 큐에 Message 객체 형태의 작업을 넣고 관리하는 객체입니다.



## 4.3 AsyncTask를 이용한 비동기 처리 (1/5)

- ✓ AsyncTask 클래스를 이용하면 직접 쓰레드의 생성이나, 핸들러 구현 없이 쓰레드를 구현할 수 있습니다.
- ✓ AsyncTask 클래스 내부적으로 쓰레드를 생성하여 필요할 때마다 메인 쓰레드에서 실행되는 콜백 메서드를 호출합니다.
- ✓ AsyncTask 클래스를 상속하여 생성하고 콜백 메서드 내부에 쓰레드로 구현하고자 하는 코드를 작성합니다.



## 4.3 AsyncTask를 이용한 비동기 처리 (2/5)

---

### ✓ AsyncTask를 이용한 비동기 처리 장점은 ...

- 메인 스레드와 동기화되어 있는 편리한 이벤트 핸들러들을 제공합니다.
- 스레드의 생성, 관리, 동기화와 관련된 모든 일들을 처리합니다.
- 백그라운드에서 수행할 비동기 작업을 생성, 작업 완료시 UI를 업데이트 할 수 있습니다.

### ✓ AsyncTask는 한번만 실행할 수 있고, 기 오출된 AsyncTask를 또 오출하면 예외가 발생합니다.

### ✓ 또한, 호출한 Activity가 AsyncTask를 종료하지 않고 종료되어도 AsyncTask를 계속 실행합니다.

## 4.3 AsyncTask를 이용한 비동기 처리 (3/5)

✓ AsyncTaks를 이용한 비동기 처리를 위한 사용법은 AsyncTask 클래스를 확장하여 구현합니다.

- AsyncTask 클래스는 <[입력 파라메터 타입], [진행 상황 보고 타입], [결과 타입]>와 같이 타입을 정의합니다.
- 만약 각 타입이 없을 경우에는 java.lang.Void로 지정합니다.

// 사용 예

```
class MyAsyncTask extends AsyncTask<Integer, Integer, Integer> {  
    ...  
}
```

타입 정의	설명
1 <sup>st</sup> 타입 – 입력 파라메터 타입	doInBackground() 메소드의 파라메터들의 타입을 정의합니다.
2 <sup>nd</sup> 타입 – 진행 상황보고 타입	onProgressUpdate() 메소드의 파라메터들의 타입을 정의합니다.
3 <sup>rd</sup> 타입 – 결과 타입	doInBackground() 메소드의 결과 타입 및 onPostExecute() 메소드의 파라메터의 타입을 정의합니다.

## 4.3 AsyncTask를 이용한 비동기 처리 (4/5)

✓ **AsyncTask** 클래스를 확장하여 구현할 메소드는 다음과 같습니다.

- `doInBackground()` 메소드는 백그라운드 작업을 수행합니다.
- `onProgressUpdate()` 메소드는 작업 경과를 표시합니다.
- `onPostExecute()` 메소드는 백그라운드 작업 후 결과 처리를 합니다.

타입 정의	설명
<code>dolnBackground</code>	<ul style="list-style-type: none"><li>• <code>AsyncTask</code>의 입력 파라미터 타입 부분에 지정된 타입으로 된 매개변수들을 전달 받습니다.</li><li>• 백그라운드 스레드에서 실행되므로 UI 객체와 상호작용하려 해서는 안됩니다.</li><li>• <code>publishProgress</code> 메소드를 이용하여 <code>onProgressUpdate</code> 진행사항을 UI에 전달합니다.</li><li>• 작업이 완료되면 <code>onPostExecute</code> 핸들러에 리턴하여 이를 UI에 보고합니다.</li></ul>
<code>onProgressUpdate</code>	<ul style="list-style-type: none"><li>• 중간 진행 상황을 UI 스레드에 전달합니다.</li><li>• <code>dolnBackground</code>에서 <code>publishProgress</code>에 전달한 파라미터들을 전달 받습니다.</li><li>• GUI 스레드와 동기화 되므로 UI 요소들을 안전하게 변경할 수 있습니다.</li></ul>
<code>onPostExecute</code>	<ul style="list-style-type: none"><li>• <code>dolnBackground</code> 작업을 마치고 리턴된 값을 전달 받습니다.</li><li>• 비동기 작업을 마치고 UI를 업데이트 하기 위해 사용합니다.</li><li>• GUI 스레드와 동기화되므로 UI 요소들을 안전하게 변경할 수 있습니다.</li></ul>

## 4.3 AsyncTask를 이용한 비동기 처리 (5/5)

✓ AsyncTask 클래스의 타입 정의와 구현 메소드와의 관계에 대한 사용 예입니다.

- 입력 파라메터 타입 (Integer) – doInBackground(Integer... params)
- 진행 상황보고 타입 (Integer) – onProgressUpdate(Integer... progress)
- 결과 타입 (Integer) – onPostExecute(Integer result), Integer doInBackground()

```
class MyAsyncTask extends AsyncTask<Integer, Integer, Integer> {  
    protected void onPreExecution(){  
        // TODO 사전작업  
    }  
    protected Integer doInBackground(Integer ... params){  
        Integer result;  
        // TODO 백그라운드 작업  
        // publishProgress(Integer... progress) 를 통해 onProgressUpdate()메서드를 호출  
        return result;  
    }  
    protected void onProgressUpdate(Integer... progress){  
        // TODO 작업 경과를 표시하기 위해 호출  
    }  
    protected void onPostExecute(Integer result){  
        // TODO 백그라운드 작업이 끝난 후 UI 쓰레드에서 실행할 작업  
    }  
    protected void onCancelled (){  
        // TODO cancel 메서드로 작업을 취소하였을 경우 실행  
    }  
}
```

## 4.4 안드로이드 네트워크 (1/8) – 연결관리자 (1/3)

- ✓ 안드로이드는 다양한 네트워크 장비를 이용한 연결을 위한 연결 관리자를 제공합니다.
- ✓ 연결 관리자(ConnectivityManager)는 모바일 장비의 네트워크 상태 및 정보를 관리하는 객체입니다.
- ✓ 연결 관리를 위해서는 안드로이드 매니페스트에 네트워크를 사용하기 위한 권한(Permission)을 추가해야 합니다.

```
<users-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

### 주요 기능

- 사용 가능한 네트워크 정보를 조회합니다.
- 각 연결방법의 현재 상태를 조회합니다.
- 연결상태가 변경되면 모든 애플리케이션에 인텐트로 공지합니다.
- 연결에 실패하면 대체 가능한 다른 연결을 조회합니다.

## 4.4 안드로이드 네트워크 [2/8] – 연결관리자 [2/3]

- ✓ 연결 관리자 사용을 위해 시스템 서비스로부터 ConnectivityManager 객체를 획득하여 사용합니다.
  - 즉, ConnectivityManager는 시스템이며, getSystemService() 메서드를 통해서 객체를 전달 받습니다.
- ✓ 또한, ConnectivityManager 클래스는 NetworkInfo 객체를 제공하는 메소드가 정의되어 있습니다.

메소드	설명
public NetworkInfo[] getAllNetworkInfo();	모든 네트워크의 연결 방법에 대한 정보를 받습니다.
public NetworkInfo getActiveNetworkInfo();	현재 액티브 상태인 네트워크의 연결 방법에 대한 정보를 받습니다.
public NetworkInfo getNetworkInfo(int networkType);	특정 타입을 갖는 네트워크의 연결방법에 대한 정보를 받습니다.

### NetworkInfo 클래스의 속성 조사 메소드

- public boolean isAvailable();
- public boolean isConnected();
- public boolean isRoaming();
- public NetworkInfo.State getState();

## 4.4 안드로이드 네트워크 (3/8) – 연결관리자 (3/3)

- ✓ 연결관리자를 통해 모바일 망 및 WIFI 망에 연결되었는지 확인을 위해서는 다음의 속성을 확인합니다.
  - ConnectivityManager.TYPE\_MOBILE, ConnectivityManager.TYPE\_WIFI 연결 타입으로 구분합니다.
- ✓ 다음의 코드는 메소드를 이용해 모바일의 연결 상태를 확인하는 코드 예제입니다.

```
/**  
 * 모바일 연결여부를 확인합니다.  
 */  
public static boolean isConnectedByMobile(Context context) {  
    ConnectivityManager systemService =  
        (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);  
    NetworkInfo mobile = systemService.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);  
    return mobile.isConnected();  
}
```

```
/**  
 * WIFI 연결여부를 확인합니다.  
 */  
public static boolean isConnectedByWiFi(Context context) {  
    ConnectivityManager systemService =  
        (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);  
    NetworkInfo mobile = systemService.getNetworkInfo(ConnectivityManager.TYPE_WIFI);  
    return mobile.isConnected();  
}
```

## 4.4 안드로이드 네트워크 (4/8) – HTTP 통신 (1/5)

- ✓ HTTP 요청은 웹의 기본 프로토콜이며, 자바의 API를 이용합니다.
- ✓ 자바의java.net 패키지에서 제공하는 클래스들을 이용하여 HTTP 요청을 구현합니다.
- ✓ HTTP 요청을 위해서는 안드로이드 매니페스트에 인터넷 사용하기 위한 권한(Permission)을 추가해야 합니다.

```
<users-permission android:name="android.permission.INTERNET" />
```

### HTTP 통신을 위한 클래스

- 인터넷 자원을 지칭하는 URL 클래스 및 URL 연결을 위한 URLConnection 클래스가 있습니다.

## 4.4 안드로이드 네트워크 [5/8] – HTTP 통신 [2/5]

✓ URL 클래스는 인터넷 자원을 지칭하며, 보통 문자열 하나로 표현하거나 분리해서 URL 객체의 인자로 생성 가능합니다.

- URL 클래스의 메소드는 URL을 구성하는 각 요소 별로 분리하여 값을 받을 수 있도록 제공합니다.
- 만약, 규칙에 맞지 않은 유효하지 않은 URL을 설정하면 MalformedURLException 예외가 발생합니다.

// 생성자

```
URL(String spec)  
URL(String protocol, String host, int port, String file);
```

메소드	설명	반환 값 (예)
String getProtocol()	통신 방법을 정의하는 프로토콜을 리턴 합니다.	http
int getDefaultPort()	프로토콜이 정의하는 디폴트 포트를 리턴 합니다.	80
int getPort()	URL에 정의된 포트, 없을 경우 -1이 리턴 합니다.	-1 (포트가 없을 경우)
String getHost()	서버 주소를 리턴 합니다.	www.kosta.com
String getFile()	서버 내의 경로(Path)와 쿼리를 리턴 합니다.	/main.do?user=admin
String getPath()	서버 내의 경로(Path)를 리턴 합니다.	/main.do
String getQuery()	서버로 전달되는 쿼리 변수 값을 리턴합니다.	user=admin

## 4.4 안드로이드 네트워크 [6/8] – HTTP 통신 [3/5]

- ✓ URLConnection 클래스는 추상 클래스이며 openConnection()를 통해 네트워크 연결 객체를 받습니다.
- ✓ 프로토콜에 따라 다양한 연결 객체를 반환합니다.
- ✓ 웹에 대한 접속은 HttpURLConnection 객체가 반환되며, 이를 캐스팅하여 사용합니다.

메소드	설명
setConnectTimeout(int timeout)	연결 제한 시간을 1/1000초 단위로 지정 합니다. 만약, 0으로 지정할 경우 무한 대기 합니다.
setReadTimeout(int timeout)	읽기 제한 시간을 지정 합니다. 만약, 0이면 지정할 경우 무한 대기 합니다.
setUseCaches(boolean newValue)	캐시 사용 여부를 지정 합니다.
setRequestProperty(String field, String newValue)	요청 헤더에 값을 설정 합니다.

## 4.4 안드로이드 네트워크 [7/8] – HTTP 통신 [4/5]

✓ 웹 접속에 따른 HttpURLConnection을 통한 연결일 경우에는 요청 방식을 지정해야 합니다.

- 요청 방식(HTTP Method)는 GET, POST, PUT 등이 있습니다.
- 모든 설정이 완료되면 서버로 요청을 보내 후, 서버로 부터 응답에 대한 결과 코드를 받아 확인할 수 있습니다.
- 또한, 서버로 부터 요청에 대한 응답 결과를 스트림을 통해서 받을 수 있습니다.

```
// HTTP Method 설정  
// GET, POST ...  
  
void HttpURLConnection.setRequestMethod(String method)
```

```
// 원격지 서버로 부터의 응답 결과  
// 요청 성공일 경우 HTTP_OK (200), 요청에 대한 자원이 없을 경우 HTTP_NOT_FOUND (400) ...  
  
int getResponseCode()
```

```
// 서버로 부터 전송된 결과를 읽을 입력 스트림  
  
InputStream getInputStream()
```

## 4.4 안드로이드 네트워크 (8/8) – HTTP 통신 (5/5)

```
public static String get(Context context, String path) {  
    StringBuilder sb = new StringBuilder();  
  
    try {  
        URL url = new URL(path);  
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();  
        if (conn != null) {  
            conn.setRequestMethod(HTTP_GET);  
            conn.setConnectTimeout(10000);  
            conn.setUseCaches(false);  
  
            if (conn.getResponseCode() == HttpURLConnection.HTTP_OK) {  
                BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));  
  
                String content;  
                while ((content = br.readLine()) != null) {  
                    sb.append(content);  
                }  
                br.close();  
            }  
            conn.disconnect();  
        }  
    } catch (MalformedURLException e) {  
        e.printStackTrace();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    return sb.toString();  
}
```

## 4.5 전문 변환 (1/10) – XML 파서 개요

✓ XML은 웹 서비스의 기본 데이터 포맷으로 서버와 클라이언트의 주요 통신 수단입니다.

- 서버는 클라이언트의 요청을 받아들여 처리하고 그 결과를 XML로 반환하며 클라이언트는 XML을 분석하여 처리 결과를 얻습니다.
- 웹 서비스가 제공하는 기능을 활용하기 위해서는 서버로부터 반환된 XML 문서를 빠르고 정확하게 변환할 수 있어야 합니다.
- XML 자체는 단순한 텍스트 포맷이지만 규칙이 엄격해서 정확한 정보를 획득하는 것이 쉽지 않습니다.
- 정확한 위치의 문자열을 읽기 위해 사용하는 것이 파서(Parser)입니다.

### DOM Parser

- 트리 형식으로 문서를 읽어서 전체 구조를 파악한 후 정보를 구합니다.
- 메모리를 많이 사용하는 반면 성능이 좋습니다.
- XML의 동적 구조 변경이 쉽습니다.

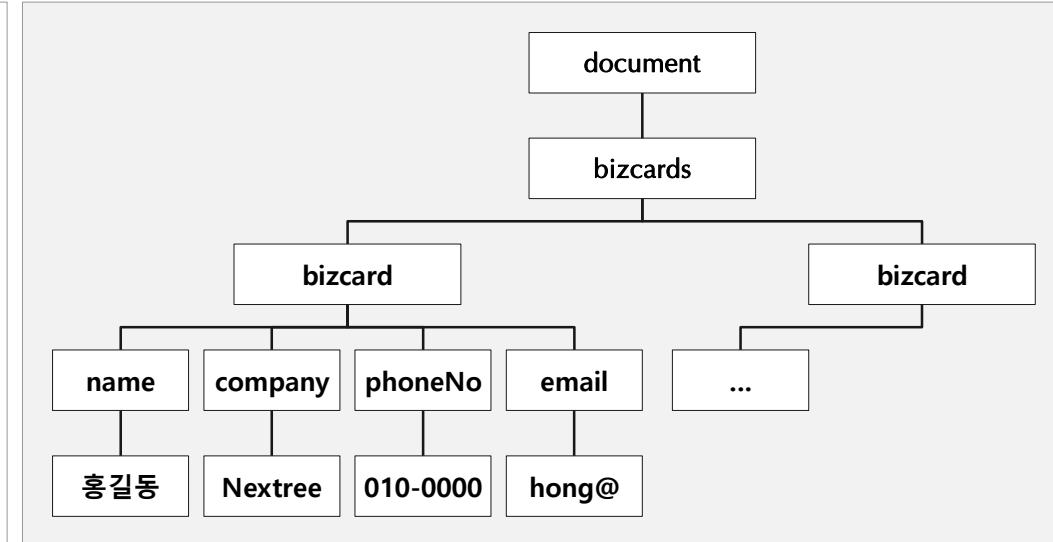
### SAX Parser

- 순차적으로 문서를 읽으면서 정보를 구합니다.
- 속도는 느리지만 메모리를 사용이 적습니다.
- 한번만 읽는다면 DOM 파서 보다 빠릅니다.

## 4.5 전문 변환 (2/10) – XML DOM 파서 (1/4)

- ✓ DOM은 DocumentObjectModel의 약어이며, XML을 문서 객체 구조로 구조화 합니다.
  - 즉, DOM 파서를 통한 XML 데이터 파싱을 위해서는 DocumentObjectModel의 구조에 대한 이해가 필요합니다.
- ✓ DOM은 전체 트리 형식으로 문서를 구성하고 전체 구조를 파악하여 정보를 얻는 방식입니다.

```
<?xml version="1.0" encoding="utf-8"?>
<bizcards>
  <bizcard>
    <name>홍길동</name>
    <company>Nextree</company>
    <phoneNo>010-0000-0000</phoneNo>
    <email>hong@namoosori.com</email>
  </bizcard>
</bizcards>
```



## 4.5 전문 변환 (3/10) – XML DOM 파서 (2/4)

✓ DOM 파서를 통한 XML 데이터 파싱 및 결과 값을 받기 위해서는 DOM API를 활용해야 합니다.

- DOM 파싱은 XML을 DOM 구조로 파싱하는 영역과 DOM 구조로 부터 데이터를 추출하는 영역으로 구분하여 구현합니다.
- DOM 파싱은 노드의 검색, 수정, 구조의 변경이 빠르고 쉽습니다.
- 직관적이고 SAX 파싱보다 방식이 단순하지만 XML 문서의 모든 내용을 메모리에 로딩한 이후에 데이터를 얻어야 하는 단점이 있습니다.

```
// XML을 읽어 DOM 구조로 파싱합니다.  
// DocumentBuilder를 통해 DOM 구조화된 Document 객체를 받습니다.  
  
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
DocumentBuilder builder = factory.newDocumentBuilder();  
InputStream istream = new ByteArrayInputStream(xml.getBytes("utf-8")); //charset 지정  
Document doc = builder.parse(istream);  
  
// DOM 구조화된 Document 객체로 부터 ROOT 요소를 받은 후  
// 요소로 부터 NodeList 받아 하위의 Node로 부터 데이터를 추출 합니다.  
// 하위 Node로 부터 데이터를 추출하기 위해 parseBizCards()라는 사용자 정의 메소드를 구현합니다.  
  
Element root = doc.getDocumentElement();  
NodeList bizCardNodes = root.getElementsByTagName("bizcard");  
List<BizCard> bizCards = parseBizCards(bizCardNodes);
```

## 4.5 전문 변환 (4/10) – XML DOM 파서 (3/4)

✓ NodeList로 부터 하위 Node 별로 데이터를 추출하기 위한 메소드를 추가하여 구현합니다.

- 각 Node 객체는 getChildNodes() 메소드를 통해 하위 NodeList를 받을 수 있습니다.
- 하위 NodeList로 부터 item() 메소드를 통해 Node 객체를 받아 값을 추출합니다.

```
public List<BizCard> parseBizCards(NodeList bizCardNodes) {  
    List<BizCard> bizCards = new ArrayList<BizCard>();  
    int length = bizCardNodes.length;  
    for (int i = 0; i < length; i++) {  
        Node cardNode = bizCardNodes.item(i);  
  
        NodeList cardNodeList = cardNode.getChildNodes();  
        // 첫번째 node를 name node로 판단.  
        Node nameNode = cardNodeList.item(0);  
        Node nameTextNode = nameNode.getFirstChild();  
        String name = nameTextNode.getNodeValue();  
        // 두번째 node를 company node로 판단.  
        Node companyNode = cardNodeList.item(0);  
        Node companyTextNode = companyNode.getFirstChild();  
        String company = companyTextNode.getNodeValue();  
        ...  
        bizCards.add(new BizCard(name, company, phoneNo, email));  
    }  
    return bizCards;  
}
```

## 4.5 전문 변환 (5/10) – XML DOM 파서 (4/4)

✓ XML 구성이 속성으로 정의된 데이터는 attribute를 통해 접근합니다.

- Node 클래스의 getAttributes() 메소드를 통해서 NamedNodeMap 객체를 받습니다.
- 속성 값은 NamedNodeMap 객체를 통해 속성 명칭에 해당하는 Node를 찾아 값을 추출합니다.

```
<?xml version="1.0" encoding="utf-8"?>
<bizcards>
    <bizcard name="홍길동" company="Nextree" phoneNo="010" email="xxxx@nextree.co.kr"/>
</bizcards>
```

```
public List<BizCard> parseBizCards(NodeList bizCardNodes) {
    List<BizCard> bizCards = new ArrayList<BizCard>();
    int length = bizCardNodes.length;
    for (int i = 0; i < length; i++) {
        Node cardNode = bizCardNodes.item(i);
        NamedNodeMap cardNodeMap = cardNode.getAttributes();
        Node nameAttr = cardNodeMap.getNamedItem("name");
        String name = nameAttr.getNodeValue();

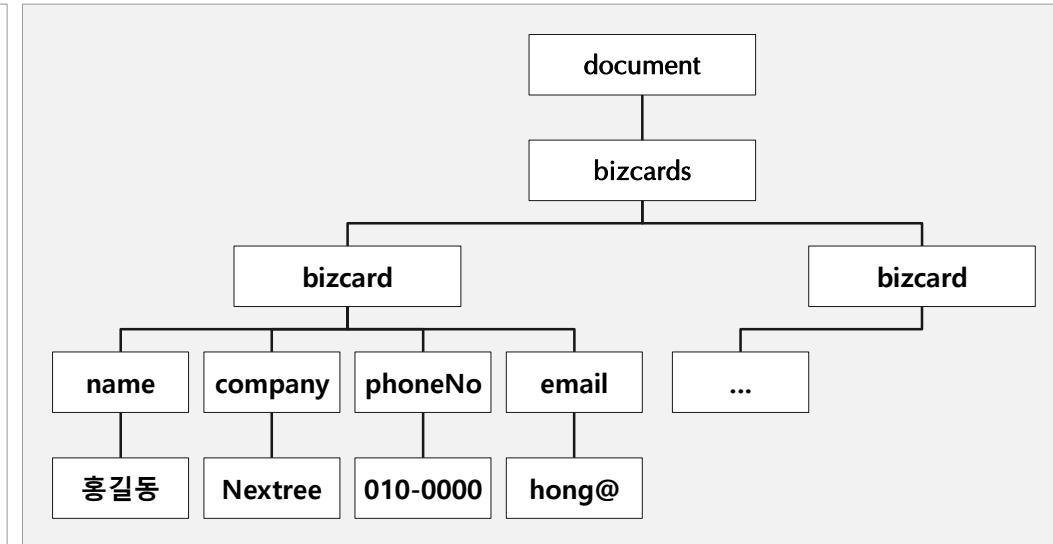
        ...
        bizCards.add(new BizCard(name, company, phoneNo, email));
    }
    return bizCards;
}
```

## 4.5 전문 변환 (6/10) – XML SAX 파서 (1/4)

✓ SAX 파서는 문자열 스트림 기반으로 XML을 분석합니다.

- 즉, 문자열 스트림으로부터 한 줄씩 읽어 들여, XML 태그에 대한 요소 및 값을 분석합니다.
- SAX 파싱은 메모리의 사용량이 적고 데이터를 읽는 용도로 사용될 경우 속도가 빠릅니다.

```
<?xml version="1.0" encoding="utf-8"?>
<bizcards>
  <bizcard>
    <name>홍길동</name>
    <company>Nextree</company>
    <phoneNo>010-0000-0000</phoneNo>
    <email>xxxx@nextree.co.kr</email>
  </bizcard>
</bizcards>
```



## 4.5 전문 변환 (7/10) – XML SAX 파서 (2/4)

✓ SAX 파서를 통한 XML 데이터 파싱 및 결과 값을 받기 위해서는 SAX API를 활용해야 합니다.

- SAX 파싱은 XML을 스트림으로 부터 읽으면서, 이를 SAX API를 통해 데이터를 추출하도록 구성되어 있습니다.
- SAX 파싱을 위해서는 SAXParserFactory 클래스를 통해 SAXParser 객체를 얻어야 합니다.
- SaxHandler 클래스는 SAX 파서 API의 DefaultHandler 클래스를 확장한 클래스입니다.

```
// XML을 읽어 파싱할 SAXParser 및 XMLReader 객체를 생성합니다.  
SAXParserFactory factory = SAXParserFactory.newInstance();  
SAXParser parser = factory.newSAXParser();  
XMLReader reader = parser.getXMLReader();  
  
// XML을 스트림으로 읽어 처리할 Handler를 생성한 후 등록합니다.  
SaxHandler handler = new SaxHandler();  
reader.setContentHandler(handler);  
  
// XML을 스트림으로 부터 읽고 파싱을 수행 합니다.  
// XML 파싱을 수행하면서 내부에서 등록된 Handler를 호출하면서 데이터를 추출하도록 요청합니다.  
InputStream istream = new ByteArrayInputStream(xml.getBytes("utf-8"));  
reader.parse(new InputSource(istream));  
  
// 파싱된 최종 결과를 Handler로 부터 받습니다.  
List<BizCard> cards = handler.getCards();
```

## 4.5 전문 변환 (8/10) – XML SAX 파서 (3/4)

✓ SAX 파싱을 위해 org.xml.sax.helpers.DefaultHandler 클래스를 확장하여 구현 합니다.

- SAX API는 문서 시작 – 요소 시작 – 요소 간 문자 읽기 – 요소 종료 – 문서 종료 등을 처리하기 위한 메소드를 제공합니다.
- startElement() 메소드를 재정의 하여 XML 요소를 파악하여 데이터 추출을 위한 처리를 합니다.

```
class SaxHandler extends DefaultHandler {  
    private List<BizCard> cards = new ArrayList<BizCard>();  
    private BizCard card;  
    private StringBuilder sb = new StringBuilder();  
  
    public List<BizCard> getCards() {  
        return this.cards;  
    }  
  
    public void startDocument() {}  
  
    public void endDocument() {}  
  
    public void startElement(String uri, String localName, String qName, Attributes atts) {  
        if (localName.equals("bizcard")) {  
            card = new BizCard();  
        }  
    }  
  
    // 다음 페이지에 계속
```

## 4.5 전문 변환 (9/10) – XML SAX 파서 (4/4)

✓ SAX 파싱을 위해 `org.xml.sax.helpers.DefaultHandler` 클래스를 확장하여 구현해야 합니다.

- SAX API는 문서 시작 – 요소 시작 – 요소 간 문자 읽기 – 요소 종료 – 문서 종료 등을 처리하기 위한 메소드를 제공합니다.
- `endElement()` 메소드를 재정의 하여 XML 요소의 데이터를 읽어 원하는 데이터 형태로 변환합니다.

```
public void endElement(String uri, String localName, String qName) {  
    if (localName.equals("bizzard")) {  
        cards.add(card);  
    } else if (localName.equals("name")) {  
        card.setName(sb.toString());  
    }  
    ...  
}  
  
public void characters(char[] chars, int start, int length) {  
    sb.setLength(0);  
    sb.append(chars, start, length);  
}  
}
```

## 4.5 전문 변환 (10/10) – JSON 파서

- ✓ JSON은 JavaScript Object Notation으로 경량의 데이터 교환 방식입니다.
- ✓ JSON 파서는 JSON 문자열을 읽어 객체 형태로 변환 합니다.
  - JSONArray 및 JSONObject 클래스로 부터 객체 형태로 변화하여 데이터 값을 읽을 수 있습니다.

- JSON 문자열 : [ 1, 2, 3, 4, 5]

```
// JSONArray 클래스는 JSON문자열에서 배열을 읽어 들여 내부 메모리에 배열 형태로 저장 합니다.  
// JSONArray의 요소는 JSONObject로 구성 될 수 있습니다.
```

```
JSONArray jsonArray = new JSONArray(jsonResponse);  
jsonArray.getInt(0);
```

- JSON 문자열 :
- { "name": "홍길동" , "company": "Nextree" , "phoneNo": "010-0000-0000" , "email": "xxxx@nextree.co.kr" }

```
// JSONObject 클래스는 JSON 문자열에서 객체를 읽어 들입니다.
```

```
JSONObject jsonObject = new JSONObject (jsonResponse);
```

```
String name = jsonObject.getString("name");           // 매핑되는 대상이 없을경우 JSONException 발생  
String company = jsonObject.optString("company"); // 매핑되는 대상이 없을경우 지정된 기본값 반환
```

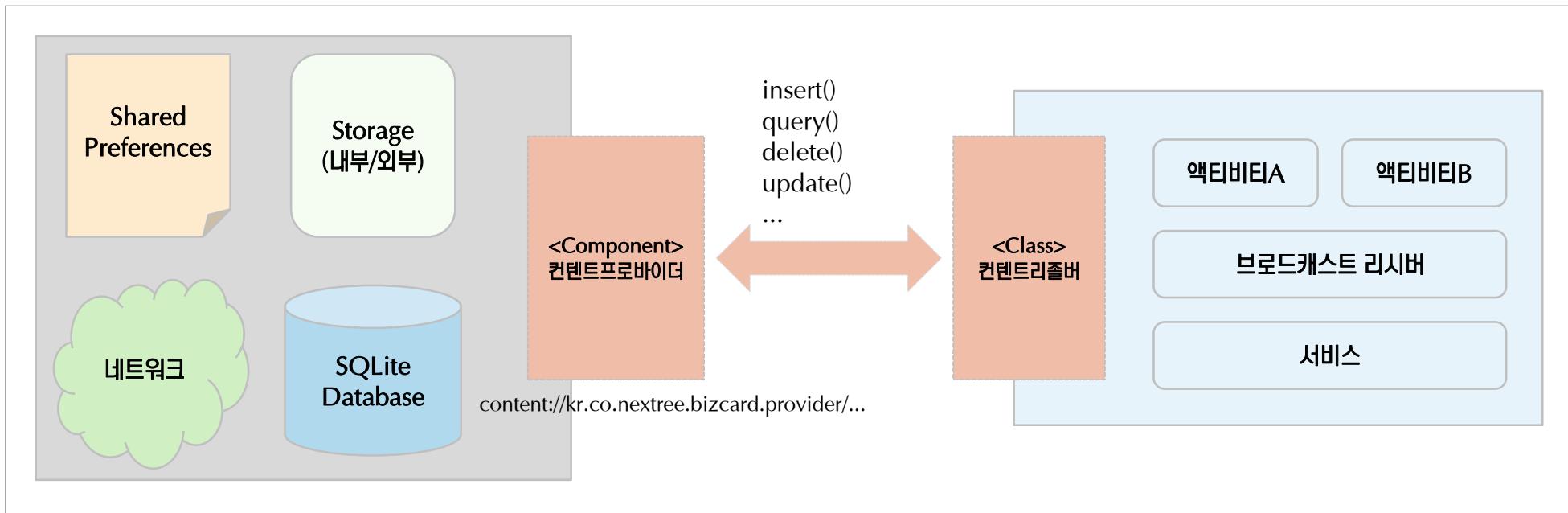


## 5. 안드로이드 활용

- 
- 5.1 컨텐트 프로바이더
  - 5.2 브로드캐스트 리시버
  - 5.3 알림

## 5.1 컨텐트 프로바이더 (1/14) – 개요 (1/2)

- ✓ 안드로이드는 애플리케이션 보안 제약으로 다른 애플리케이션이 갖고 있는 데이터에 접근 할 수 없습니다.
- ✓ 다른 애플리케이션과의 정보 공유를 위해서는 컨텐트 프로바이더를 사용해야 합니다.
- ✓ 컨텐트 프로바이더는 다른 애플리케이션의 정보 접근을 위한 통로를 제공합니다.



## 5.1 컨텐트 프로바이더 [2/14] – 개요 [2/2]

✓ 컨텐트 프로바이더는 URI(Uniform Resource Identifier) 형식의 접근방식 제공합니다.

- URI는 정보의 고유한 명칭으로서 웹상의 주소를 나타내는 URL보다 더 상위의 개념입니다.
- 누가 어떤 정보를 제공하는지, 또 어떤 정보를 원하는지에 대한 정보를 URI에 표현합니다.
- 적합한 URI를 판단하기 위해 UriMatcher 클래스를 사용합니다.

국제 표준(RFC 2396)에 URI를 작성하는 방식 [ content://authority/path/id ]

- content://는 URI임을 나타내는 접두어이며 무조건 붙입니다.
- authority는 정보 제공자의 명칭이며 중복을 허용하지 않습니다. 따라서 패키지명을 사용할 것을 권장합니다.
- path는 정보의 종류를 지정하는 가상의 경로입니다.
- id는 구체적으로 어떤 정보를 원하는지 지정하며 전체 정보를 다 읽을 때는 생략 가능합니다.

예) content://kosta.android.bizcard/bizcard  
content://kosta.android.bizcard/bizcard/1

\* id까지 있으면 단수, path까지만 있으면 복수를 뜻합니다.

- 임의의 내용을 가질 수 있는 문자열 형태이지만 제공하는 자와 제공을 받는자 간의 약속이므로 형식을 맞춰야 합니다.
- 적합한 URI를 판단하기 위해 UriMatcher 클래스를 사용합니다.
- 컨텐트 프로바이더 내부의 데이터베이스 생성을 위해서는 SQLiteOpenHelper 클래스를 사용 합니다.
- 컨텐트 프로바이더는 안드로이드 애플리케이션 컴포넌트이기 때문에 AndroidManifest.xml에 노드를 등록해야 합니다.

## 5.1 컨텐트 프로바이더 (3/14) – UriMatcher

✓ UriMatcher 클래스는 컨텐트 프로바이더에서 처리할 수 있는 URI를 등록 합니다.

- UriMatcher 클래스는 리졸버에서 전달한 Uri를 분석하여 요청을 구분하기 위해 사용합니다.
- URI를 등록하기 위해서는 authority와 path를 code에 매칭하여 정의해야 합니다.
- Uri 등록을 위해서는 addURI() 메소드를 이용합니다.

UriMatcher에 등록하기 위한 authority – path – code 매칭 정의

- # : Any Number
- \$ : Any String

예)

content://kosta.android.bizcard/bizcard	: 명함 영역 작업
content://kosta.android.bizcard/bizcard/1	: 명함 단건 작업

```
private static final int BIZ_CARD = 0;
private static final int BIZ_CARD_ID = 1;

private static UriMatcher matcher;
static {
    matcher = new UriMatcher(UriMatcher.NO_MATCH);
    matcher.addURI(BIZCARD.AUTHORITY, "bizcard", BIZ_CARD);
    matcher.addURI(BIZCARD.AUTHORITY, "bizcard/#", BIZ_CARD_ID);
}
```

## 5.1 컨텐트 프로바이더 (4/14) – MIME 타입

- ✓ 컨텐트 프로바이더의 `getType(uri)` 메서드를 통해 데이터의 MIME 타입을 반환합니다.
- ✓ 반환되는 MIME 타입을 통해 데이터의 타입을 판별합니다.
- ✓ `<type>/<sub type>` 형식으로 간편하게 데이터 타입을 구분할 수 있습니다.

복수 : vnd.회사명.cursor.dir/타입

단수 : vnd.회사명.cursor.item/타입

- UriMatcher에 대응되는 MIME 타입을 반환하도록 지정한다.

예)

vnd.kosta.cursor.dir/vnd.kosta.android.bizcard : 명함 목록 타입

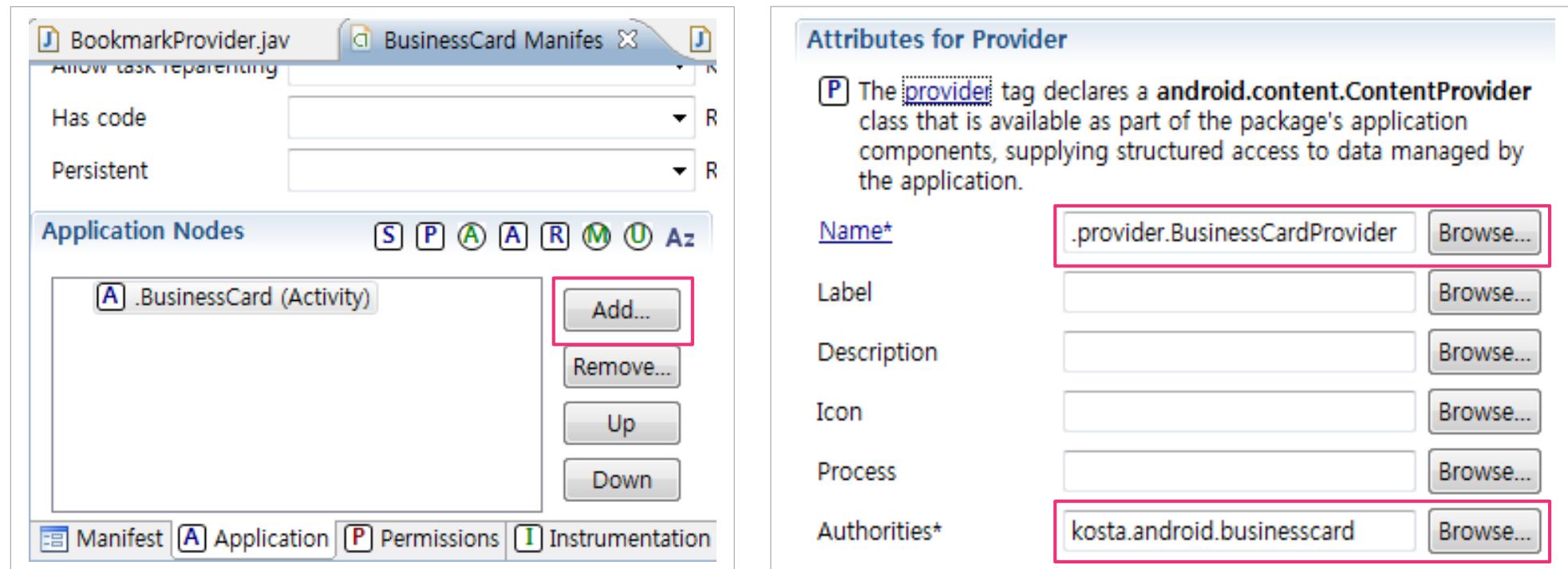
vnd.kosta.cursor.item/vnd.kosta.android.bizcard : 명함 단건 타입

```
public String getType(Uri uri){  
    switch(matcher.match(uri)) {  
        case BUSINESS_CARDS : return "vnd.kosta.cursor.dir/vnd.kosta.android.bizcard";  
        case BUSINESS_CARD_ID : return "vnd.kosta.cursor.item/vnd.kosta.android.bizcard";  
    }  
}
```

## 5.1 컨텐트 프로바이더 (5/14) – 컨텐트 프로바이더 등록

### ✓ 컨텐트 프로바이더 등록

- AndroidManifest.xml의 Application 탭에서 Add 버튼을 클릭하여 Provider를 추가합니다.
- Provider의 Name속성을 Browse... 버튼을 클릭, 클래스 선택하여 추가합니다.
- Authorities는 ContentProvider의 AUTHORITY를 작성합니다.



## 5.1 컨텐트 프로바이더 [6/14] – 컨텐츠 리졸버 (1/2)

- ✓ 컨텐츠 리졸버(Content Resolver)를 통해서 데이터베이스 작업을 수행합니다.
- ✓ 컨텐츠 리졸버(Content Resolver)는 데이터 조회를 위해 `query()` 메소드와 `managedQuery()` 메소드를 제공합니다.
  - `query()` 메소드는 컨텐트 프로바이더의 주소를 사용하여 목록을 커서 형태로 조회합니다.
  - `managedQuery()` 메소드는 Activity의 메서드, 액티비티의 라이프 사이클에 맞춰 자동으로 Cursor를 닫습니다.

```
• public Cursor query(String uri, String projection,  
                      String selection, String[] selectionArgs, String sortOrder);
```

// 사용 예

```
ContentResolver cr = getContentResolver();  
Cursor cursor = cr.query(BizCardTables.BizCard.CONTENT_URI, null, null, null, null);
```

```
• public Cursor managedQuery(String uri, String projection,  
                           String selection, String[] selectionArgs, String sortOrder);
```

// 사용 예

```
Cursor cursor = managedQuery(BizCardTables.BizCard.CONTENT_URI, null, null, null, null);
```

## 5.1 컨텐트 프로바이더 (7/14) – 컨텐츠 리졸버 (2/2)

- ✓ 컨텐츠 리졸버(Content Resolver)는 데이터베이스 작업을 위해 insert(), update(), delete() 메소드를 제공합니다.
  - insert() 메소드는 데이터를 추가하고 추가된 Row의 URI를 반환합니다.
  - update() 및 delete() 메소드는 데이터 변경 및 삭제를 하고 이에 영향 받은 Row의 개수를 반환합니다.

### Method Signature

- `public Uri insert(String uri, ContentValues values);`
- `public final int update(String uri, ContentValues values, String where, String[] selectionArgs);`
- `public final int delete(String uri, String where, String[] selectionArgs);`

## 5.1 컨텐트 프로바이더 [8/14] – 주소록 가져오기 실습 [1/3]

- ✓ 주소록 가져오기 위해서는 안드로이드 매니페스트에 읽기 접근 권한(Permission)을 추가해야 합니다.
- ✓ 주소록을 읽어 오기 위한 컨텐츠 리졸버를 Activity.getContentResolver() 메소드를 통해 반환 받습니다.
- ✓ ContentResolver.query() 메소드를 이용해 주소록 데이터를 Cursor 형태로 반환 받습니다.

```
<users-permission android:name="android.permission.READ_CONTACTS" />
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_read_contact);

    ContentResolver cr = getContentResolver();
    Cursor cursor = cr.query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null);

    int idIdx = cursor.getColumnIndex(ContactsContract.Contacts._ID);
    int nameIdx = cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME);

    StringBuilder result = new StringBuilder();

    // 다음 페이지에 계속
```

## 5.1 컨텐트 프로바이더 (9/14) – 주소록 가져오기 실습 (2/3)

- ✓ Cursor가 참조하고 있는 데이터들을 추출하기 위한 반복 과정을 수행합니다.
- ✓ 주소록에서 휴대전화 번호와 집전화 번호 데이터를 추출하기 위한 두 번째 Cursor를 준비합니다.
- ✓ 두 번째 커서를 이용해 추출한 데이터들은 문자열 형태로 관리합니다.

```
while (cursor.moveToNext()) {  
    result.append(cursor.getString(nameIdx) + " :");  
    String id = cursor.getString(idIdx);  
  
    Cursor cursor2 = cr.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,  
        ContactsContract.CommonDataKinds.Phone.CONTACT_ID + "=?", new String[] { id }, null);  
  
    int type = cursor2.getColumnIndex(ContactsContract.CommonDataKinds.Phone.TYPE);  
    int numId = cursor2.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER);  
    while (cursor2.moveToNext()) {  
        String num = cursor2.getString(numId);  
        switch (cursor2.getInt(type)) {  
            case ContactsContract.CommonDataKinds.Phone.TYPE_MOBILE:  
                result.append(" Mobile:" + num);  
                break;  
            case ContactsContract.CommonDataKinds.Phone.TYPE_HOME:  
                result.append(" Home:" + num);  
                break;  
  
        // 다음 페이지에 계속  
    }  
}
```

## 5.1 컨텐트 프로바이더 (10/14) – 주소록 가져오기 실습 (3/3)

- ✓ 주소록 데이터를 읽기 위해 사용했던 자원들을 반납합니다.
- ✓ 추출한 데이터의 내용을 TextView에 담아 화면에 표시합니다.

```
        case ContactsContract.CommonDataKinds.Phone.TYPE_WORK:  
            result.append(" Work:" + num);  
            break;  
        default:  
            break;  
        }  
    }  
    cursor2.close();  
    result.append("\n");  
}  
cursor.close();  
  
TextView mContactView = (TextView) findViewById(R.id.contactView);  
mContactView.setText(result);  
}
```

## 5.1 컨텐트 프로바이더 (11/14) – 통화기록 가져오기 실습 (1/4)

- ✓ 통화 기록을 가져오기 위해서는 안드로이드 매니페스트에 통화 로그 읽기 권한(Permission)을 추가해야 합니다.
- ✓ 또한, 통화 대상자, 통화 시점, 통화 시간, 사용자가 입력한 번호 등에 대한 통화 필드도 파악해야 합니다.
- ✓ 권한 추가 후 onCreate() 메소드에 컨텐츠 리졸버(Content Resolver)를 통해 통화 기록을 가져옵니다.

```
<users-permission android:name="android.permission.READ_CALL_LOG" />
```

- 통화 필드

필드 속성	설명
CACHED_NAME	통화 대상자의 이름 (통화 시점의 이름)
DATE	통화 시점 (1/1000초의 단위의 절대 시간)
DURATION	통화 시간이며 초 단위
NUMBER	사용자가 입력한 전화번호
TYPE	통화의 종류. INCOMING_TYPE은 수신한 전화, OUTGOING_TYPE은 발신, MISSED_TYPE은 부재 중 전화

## 5.1 컨텐트 프로바이더 [12/14] – 통화기록 가져오기 실습 [2/4]

- ✓ 통화 기록을 가져오기 위해서는 안드로이드 매니페스트에 통화 로그 읽기 권한(Permission)을 추가해야 합니다.
- ✓ 또한, 통화 대상자, 통화 시점, 통화 시간, 사용자가 입력한 번호 등에 대한 통화 필드도 파악해야 합니다.
- ✓ 권한 추가 후 onCreate() 메소드에 컨텐츠 리졸버(Content Resolver)를 통해 통화 기록을 가져옵니다.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_read_contact);

    ContentResolver cr = getContentResolver();
    Cursor cursor = cr.query(CallLog.Calls.CONTENT_URI,
        null, null, null, CallLog.Calls.DATE + " DESC");

    int nameIdx = cursor.getColumnIndex(CallLog.Calls.CACHED_NAME);
    int dateIdx = cursor.getColumnIndex(CallLog.Calls.DATE);
    int numIdx = cursor.getColumnIndex(CallLog.Calls.NUMBER);
    int durIdx = cursor.getColumnIndex(CallLog.Calls.DURATION);
    int typeIdx = cursor.getColumnIndex(CallLog.Calls.TYPE);

    StringBuilder result = new StringBuilder();
    SimpleDateFormat formatter = new SimpleDateFormat("MM/dd HH:mm");
    result.append("총 기록 개수 : " + cursor.getCount() + "개\n");
    int count = 0;

    // 다음 페이지에 계속
```

## 5.1 컨텐트 프로바이더 (13/14) – 통화기록 가져오기 실습 (3/4)

- ✓ 통화 기록을 가져오기 위해서는 안드로이드 매니페스트에 통화 로그 읽기 권한(Permission)을 추가해야 합니다.
- ✓ 또한, 통화 대상자, 통화 시점, 통화 시간, 사용자가 입력한 번호 등에 대한 통화 필드도 파악해야 합니다.
- ✓ 권한 추가 후 onCreate() 메소드에 컨텐츠 리졸버(Content Resolver)를 통해 통화 기록을 가져옵니다.

```
while (cursor.moveToNext()) {  
    String name = (name != null) ? cursor.getString(nameIdx) : cursor.getString(numIdx);  
    result.append(name);  
  
    int type = cursor.getInt(typeIdx);  
    String sType;  
    switch (type) {  
        case CallLog.Calls.INCOMING_TYPE:  
            sType = "수신";  
            break;  
        case CallLog.Calls.OUTGOING_TYPE:  
            sType = "발신";  
            break;  
        case CallLog.Calls.MISSED_TYPE:  
            sType = "부재중";  
            break;  
        case 14:  
            sType = "문자보냄";  
            break;  
    }  
    // 다음 페이지에 계속
```

## 5.1 컨텐트 프로바이더 (14/14) – 통화기록 가져오기 실습 (4/4)

- ✓ 통화 기록을 가져오기 위해서는 안드로이드 매니페스트에 통화 로그 읽기 권한(Permission)을 추가해야 합니다.
- ✓ 또한, 통화 대상자, 통화 시점, 통화 시간, 사용자가 입력한 번호 등에 대한 통화 필드도 파악해야 합니다.
- ✓ 권한 추가 후 onCreate() 메소드에 컨텐츠 리졸버(Content Resolver)를 통해 통화 기록을 가져옵니다.

```
        case 13:
            sType = "문자받음";
            break;
        default:
            sType = "기타:" + type;
            break;
    }
    result.append("(" + sType + ") : ");

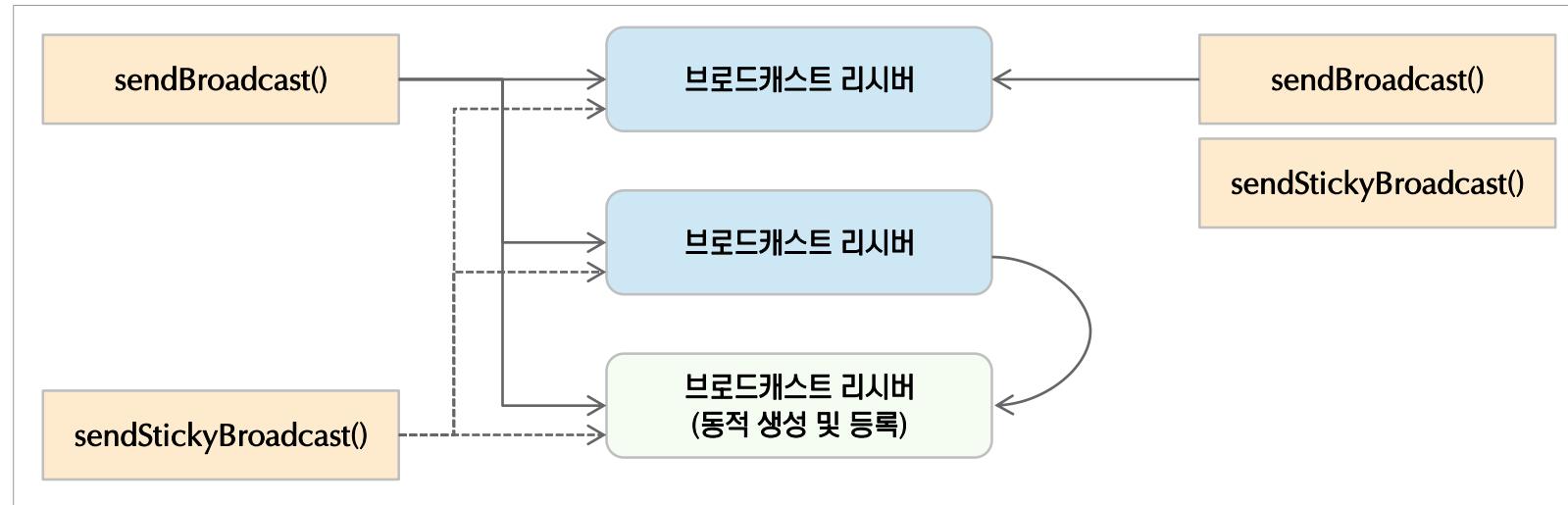
    long date = cursor.getLong(dateIdx);    // 통화 날짜
    String sdate = formatter.format(new Date(date));
    result.append(sdate + ",");

    int duration = cursor.getInt(durIdx);   // 통화시간
    result.append(duration + "초\n");

    // 최대 100개까지만
    if (count++ == 100) break;
}
}
```

## 5.2 브로드캐스트 리시버 (1/7) – 개요

- ✓ 브로드캐스트 리시버는 시스템 혹은 다른 애플리케이션에서 변화에 대한 신호를 보내면 그 신호를 받기 위한 방법입니다.
- ✓ 인텐트에 의해 동작하며 메시지 수신 이후 10초 후에 자동으로 종료합니다.
- ✓ 방송되는 방식에 따라 방송 수신의 방식을 구분합니다.



## 5.2 브로드캐스트 리시버 (2/7) – 구현 (1/3)

✓ BroadcastReceiver를 상속한 Receiver 클래스를 작성합니다.

- void onReceive(Context context, Intent intent) – 방송이 수신되면 onReceive 메소드를 호출합니다.
- context는 BR01 실행되는 컨텍스트이며, intent는 수신된 방송 내용입니다.

```
public class ReceiverTest extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

        String message = intent.getStringExtra("message");
        Log.i("ReceiverTest1", message);

        Toast.makeText(context, "Receiver Test : " + message, Toast.LENGTH_LONG).show();
    }
}
```

## 5.2 브로드캐스트 리시버 (3/7) – 구현 (2/3)

✓ 작성한 Receiver 클래스를 안드로이드 매니페스트에 등록합니다.

- 등록된 Receiver 클래스에 대해 intent-filter의 action속성에 어떤 수신에 반응 할지 지정합니다.
- 또한, intent-filter의 action은 여러 개 등록 할 수 있습니다.

```
<receiver android:name="com.myexample.myandroid.ReceiverTest">
    <intent-filter>
        <action android:name="com.myexample.myandroid.MESSAGERECEIVER" />
    </intent-filter>
</receiver>
```

- com.myexample.myandroid.MESSAGERECEIVER로 방송을 하면 ReceiverTest의 onReceiver를 호출합니다.
- intent-filter의 android:priority속성으로 우선순위를 변경할 수 있으며 높은 값일 수록 우선 순위가 높아집니다.

## 5.2 브로드캐스트 리시버 (4/7) – 구현 (3/3)

✓ Activity에서 동적으로 BroadcastReceiver 구성할 수 있습니다.

- registerReceiver() 메소드를 통해 리시버를 등록합니다.
- BroadcastReceiver의 등록을 취소하기 위해서는 unregisterReceiver() 메소드를 이용합니다.

```
public class ReceiverTestActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_receiver_test);  
    }  
    public void onResume() {  
        super.onResume();  
        IntentFilter filter = new IntentFilter();  
        filter.addAction("kr.kosta.mysample.RECEIVERTEST");  
        registerReceiver(mTestReceiver, filter); //리시버를 등록한다.  
    }  
    public void onPause() {  
        super.onPause();  
        unregisterReceiver(mTestReceiver); // 리시버를 해제한다.  
    }  
    BroadcastReceiver mTestReceiver = new BroadcastReceiver() {  
        public void onReceive(Context context, Intent intent) {  
            Toast.makeText(context, "테스트", Toast.LENGTH_LONG).show();  
        }  
    };  
}
```

## 5.2 브로드캐스트 리시버 [5/7] – 방송

- ✓ 방송 메시지는 sendBroadcast(Intent intent) 메소드 등을 통해 발송할 수 있습니다.
- ✓ 방송 메시지는 해당 방송을 수신하는 컴포넌트가 다양할 경우에 유용하기 때문에 자주 사용되지는 않습니다.
- ✓ 방송 메시지를 보내는 구현 보다는 BroadcastReceiver를 통한 방송 수신이 더욱 중요합니다.

```
Intent intent = new Intent();
intent.putExtra("message", "ReceiverTest~~~~~");
intent.setAction("com.myexample.myandroid.MESSAGERECEIVER ");
sendBroadcast(intent);
```

## 5.2 브로드캐스트 리시버 [6/7] – SMS 수신하기 [1/2]

- ✓ SMS 수신을 받기 위해서는 안드로이드 매니페스트에 통화 SMS 수신 권한(Permission)을 추가해야 합니다.
- ✓ SMS 수신 시 시스템에서 보내는 방송을 수신하는 Receiver 클래스를 구현합니다.
- ✓ 작성한 Receiver를 매니페스트에 등록합니다.
  - intent-filter에 안드로이드에서 지정한 액션 명을 작성합니다.

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

```
public class SmsReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        // SMS 메시지를 파싱합니다.
        Bundle bundle = intent.getExtras();
        Object messages[] = (Object[]) bundle.get("pdus");
        SmsMessage smsMessage[] = new SmsMessage[messages.length];

        for (int i = 0; i < messages.length; i++) {
            // PDU 포맷으로 되어 있는 메시지를 복원합니다.
            smsMessage[i] = SmsMessage.createFromPdu((byte[]) messages[i]);
        }

        // 다음 페이지에 계속
    }
}
```

## 5.2 브로드캐스트 리시버 [7/7] – SMS 수신하기 [2/2]

- ✓ SMS 수신을 받기 위해서는 안드로이드 매니페스트에 통화 SMS 수신 권한(Permission)을 추가해야 합니다.
- ✓ SMS 수신 시 시스템에서 보내는 방송을 수신하는 Receiver 클래스를 구현합니다.
- ✓ 작성한 Receiver를 매니페스트에 등록합니다.
  - intent-filter에 안드로이드에서 지정한 액션 명을 작성합니다.

```
// SMS 수신 시간 확인  
Date curDate = new Date(smsMessage[0].getTimestampMillis());  
Log.d("문자 수신 시간", curDate.toString());  
  
// SMS 발신 번호 확인  
String origNumber = smsMessage[0].getOriginatingAddress();  
  
// SMS 메시지 확인  
String message = smsMessage[0].getMessageBody().toString();  
Log.d("문자 내용", "발신자 : " + origNumber + ", 내용 : " + message);  
Toast.makeText(context, "message===> " + message, Toast.LENGTH_LONG).show();  
}  
}
```

```
<receiver android:name="com.myexample.myandroid.SmsReceiver">  
    <intent-filter android:priority="9999">  
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />  
    </intent-filter>  
</receiver>
```

## 5.3 알림(Notification) (1/7) – 개요

- ✓ 상태 변화나 특정 사건을 알리기 위한 방법을 알림을 사용하여 백그라운드 프로세스가 사용자와 통신할 수 있는 방법입니다.
- ✓ 화면 제일 위쪽의 상태 표시 영역의 왼쪽에 알림 아이콘이 나타납니다.
- ✓ 알림은 최초 잠깐만 표시되며 토스트와 달리 사용자가 확인하기 전에는 아이콘을 계속 표시합니다.
- ✓ 알림을 출력하려면 알림 관리자(NotificationManager)와 알림 객체(Notification)를 사용합니다.



## 5.3 알림 [Notification] (2/7) – 구현 (1/5)

- ✓ Notification 클래스는 내부 Builder 클래스를 통해서 생성합니다.
- ✓ Notification.Builder 클래스는 Notification 객체 내에 정의할 속성들을 정의해야 합니다.
  - 알림 속성은 기본 속성과 알림 뷰 관련 속성, 고급 속성이 있습니다.

### Notification.Builder 생성자 및 Notification 객체 생성

- Notification.Builder(Context context) / Notification build()

### Notification.Builder 기본 속성

- Notification.Builder setTicker(CharSequence tickerText [, RemoteViews views])  
// 통지 영역에 아이콘이 처음 나타날 때 잠시 출력될 짧은 문자열입니다.
- Notification.Builder setWhen(long when)  
// 통지가 발생한 시간을 지정합니다. 디폴트는 현재 시간입니다.
- Notification.Builder setSmallIcon(int icon [, int level])  
// 상태란 왼쪽에 티커 텍스트와 함께 표시합니다.
- Notification.Builder setLargeIcon(Bitmap icon)  
// 알림 뷰에 표시합니다.

## 5.3 알림 (Notification) (3/7) – 구현 (2/5)

- ✓ Notification 클래스는 내부 Builder 클래스를 통해서 생성합니다.
- ✓ Notification.Builder 클래스는 Notification 객체 내에 정의할 속성들을 정의해야 합니다.
  - 알림 속성은 기본 속성과 알림 뷰 관련 속성, 고급 속성이 있습니다.

### Notification.Builder 알림 뷰 관련 속성

- Notification.Builder setContentTitle(CharSequence title)  
// 상단제목을 설정 합니다.
- Notification.Builder setContentText(CharSequence text)  
// 중간의 내용을 설정 합니다.
- Notification.Builder setSubText(CharSequence text)  
// 하단의 서브 텍스트를 설정 합니다.
- Notification.Builder setContentIntent(PendingIntent intent)  
// 뷰를 클릭했을 때의 동작을 설정 합니다.

## 5.3 알림 (Notification) (4/7) – 구현 (3/5)

- ✓ Notification 클래스는 내부 Builder 클래스를 통해서 생성합니다.
- ✓ Notification.Builder 클래스는 Notification 객체 내에 정의할 속성들을 정의해야 합니다.
  - 알림 속성은 기본 속성과 알림 뷰 관련 속성, 고급 속성이 있습니다.

### Notification.Builder 고급 속성

- Notification.Builder setLights(int argb, int onMs, int offMs)  
// 알림 발생시 장비의 LED를 깜박거리며 색상과 주기를 지정 합니다.
- Notification.Builder setNumber(int number)  
// 알림과 숫자 하나를 보여 주는데 여러 가지 의미로 사용합니다.
- Notification.Builder setOngoing(boolean ongoing)  
// 일회적인 알림이 아니라 음악 재생이나 싱크 등의 작업이 진행 중임을 나타냅니다.
- Notification.Builder setSound(Uri sound [, int streamType])  
// 알림과 함께 출력할 소리 지정 합니다.
- Notification.Builder setVibrate(long[] pattern)  
// 진동 방식을 지정 합니다.

## 5.3 알림 (Notification) (5/7) – 구현 (4/5)

✓ PendingIntent 클래스는 인텐트를 래핑하며 다른 응용 프로그램으로 전달하여 실행 권한을 넘깁니다.

- 알림 등록과 알림 뷰를 관리하는 애플리케이션이 다른 애플리케이션이기 때문에 일반적인 방법으로는 인텐트를 전달 할 수 없습니다.
- 펜딩 인텐트는 시스템이 관리하며 인텐트를 만든 응용 프로그램이 종료되어도 유효합니다.

- PendingIntent getActivity (Context context, int requestCode, Intent intent, int flags)  
// 액티비티 호출 시 펜딩 인텐트를 가져옵니다.
- PendingIntent getBroadcast (Context context, int requestCode, Intent intent, int flags)  
// 브로드캐스트 호출 시 펜딩 인텐트를 가져옵니다.
- PendingIntent getService (Context context, int requestCode, Intent intent, int flags)  
// 서비스 호출 시 펜딩 인텐트를 가져옵니다.

- 알림 뷰에서는 주로 액티비티를 활성화하며 이 경우 인텐트에는 FLAG\_ACTIVITY\_NEW\_TASK 플래그를 지정하여 새로운 태스크에서 시작하도록 해야 합니다.

## 5.3 알림 (Notification) (6/7) – 구현 (5/5)

### ✓ 알림 관리자(Notification Manager)는 알림을 등록하거나 취소합니다.

- 알림 등록을 위한 `notify()` 메소드와 알림 취소를 위한 `cancel()` 메소드를 제공합니다.
- 알림 관리자는 시스템 서비스이므로 `getSystemService()` 메소드를 통해 `NotificationManager` 객체를 받을 수 있습니다.

#### 알림 관리자 (Notification Manager) 가져오기

- `NotificationManager manager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);`

#### 알림 등록

- `void notify(int id, Notification notification)`
  - `id` : 고유 식별번호
  - `notification` : 알림 객체

#### 알림 취소

- `void cancel(int id)`
  - `setAutoCancel(true)` 메소드로 알림 선택시 자동 취소를 할 수 있습니다.

## 5.3 알림 [Notification] (7/7) – 예제

✓ onCreate() 메소드에서 알림 관리자를 호출하고, 필요한 곳에서 알림을 생성하여 알림을 등록하는 예제입니다.

- Notification.Builder 클래스를 통해 Notification 객체를 생성합니다.
- Notification 객체에는 알림에 대한 내용을 설정합니다. 이때 PendingIntent도 함께 설정합니다.
- 완성된 Notification 객체는 NotificationManager를 통해 사용합니다.

```
NotificationManager mNotiManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

Intent intent = new Intent(NotificationSampleActivity.this, MainActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
PendingIntent content = PendingIntent.getActivity(NotificationSampleActivity.this, 0, intent, 0);

Notification noti =
    new Notification.Builder(NotificationSampleActivity.this).setTicker("일어나세요")
        .setContentTitle("기상 시간").setContentText("일어나! 일할 시간이야")
        .setSubText("일을 해야 돈을 벌고 돈을 벌어야 밥먹고 살지!")
        // .setVibrate(new long[]{1000, 1000, 500, 500, 200, 200, 200, 200})
        .setDefaults(
            Notification.DEFAULT_VIBRATE | Notification.DEFAULT_SOUND | Notification.FLAG_SHOW_LIGHTS)
        // .setLights(Color.BLUE, 300, 100)
        .setLights(0xff00ff00, 300, 100)
        .setWhen(System.currentTimeMillis())
        .setAutoCancel(true).setSmallIcon(R.drawable.icon)
        .setContentIntent(content)
        .build();

mNotiManager.notify(ALARM_NOTI, noti);
```

# 토론

---

- ✓ 질문과 대답
- ✓ 토론

