*제 2회 안드로이드 세미나 \_ Track B*

**안드로이드 어플리케이션 기초 실습**

2009. 12. 5

김태호 (커니의 안드로이드 이야기)

|  |
| --- |
| Contents |

**Android Fundamentals**

* 안드로이드 어플리케이션의 구성요소
* 액티비티의 생애주기 (Activity Lifecycle)

**Exercises**

* 새 프로젝트 생성 / 생성파일 분석
* Layout 파일을 이용한 액티비티 화면 구성
* Java Code를 이용한 액티비티 화면 구성
* 새로운 액티비티 생성하기
* 인텐트를 이용하여 액티비티 호출하기

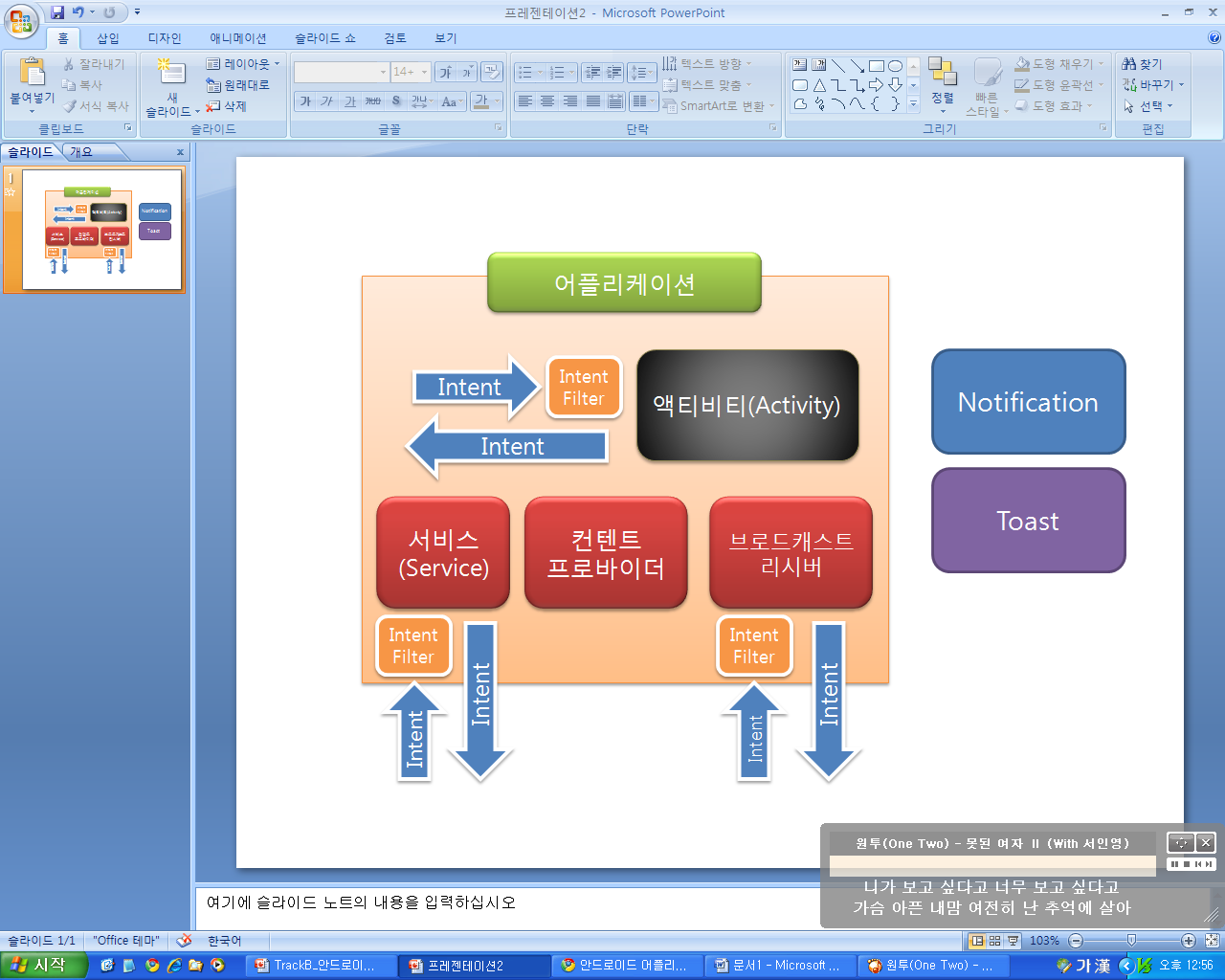
|  |
| --- |
| Android Fundamentals : 안드로이드 어플리케이션의 구성요소 |

안드로이드 어플리케이션을 만들기 전에 **안드로이드 어플리케이션은 과연 어떤 구조를 가지고 있을까?** 에 대해서 알아보도록 하겠습니다.   
  
안드로이드 어플리케이션이 어떤 구조를 가지고 있는지 알고 있어야만 앞으로 진행할 강좌도 무리없이 소화할 수 있고, 무엇보다도 처음부터 어느 정도의 기본 바탕을 잡고 공부를 하면 앞으로 배우는 것들을 이해하는 속도도 빠르고, 좀 더 머릿속에 체계적으로 정리할 수 있기 때문에 좀 지루한 감이 있더라도 잘 읽고 이해하도록 해 주세요.  

[](http://cfile1.uf.tistory.com/original/13073A1B4AA152F406E19D)

가능하다면 그냥 외우세요! 정말 중요한겁니다!!

안드로이드 어플리케이션의 구성 요소는 다음과 같이 사용자가 직접 눈으로 보며 상호작용을 하는 컴포넌트(액티비티, 알림;Notification)와 사용자의 눈으로 볼 수 없거나 백그라운드에서 실행되는 컴포넌트(컨텐트 프로바이더, 서비스, 브로드캐스트 리시버)로 나눌 수 있습니다. 사실 인텐트(Intent)와 Notificat ion은 어플리케이션 컴포넌트라고 할 수 없지만, 각 컴포넌트들과 깊은 연관을 가지고 있기에 여기에서 함께 다뤄보도록 하겠습니다.



어플리케이션을 구성하는 컴포넌트는 **액티비티, 컨텐트 프로바이더(Content Provider), 서비스(Service), Broadcast Receiver**가 있습니다. 어플리케이션은 이들 컴포넌트 중 여러 컴포넌트로 구성되며, 각 구성요소를 사용하기 위해서는 각 구성요소를 호출해야 하는데, 이를 위해 **인텐트(Intent)**를 사용합니다.   
  
인텐트도 어플리케이션 내에 포함되는것이 아니냐? 할 수도 있지만, 엄연히 따지면 "인텐트" 자체는 어플리케이션 컴포넌트들을 호출해 주는 역할을 할 뿐, 어플리케이션 내에 존재한다고 보기는 어렵습니다. 하지만, 각 컴포넌트가 호출될 수 있는 조건을 담고 있는 **인텐트 필터(Intent-Filter)**는 해당 각 컴포넌트 내에 존재합니다. 이 글에서는 어플리케이션의 '얼굴' 역할을 하는 액티비티에 대해 자세히 알아보도록 하겠습니다.

**액티비티(Activity)**는 어플리케이션의 "한 화면"을 뜻합니다, 전화번호부 어플리케이션을 예로 들자면 전화번호부를 조회하는 화면, 전화번호를 새로 추가하는 화면, 입력된 데이터를 수정하는 화면 등등이 하나의 액티비티가 됩니다.

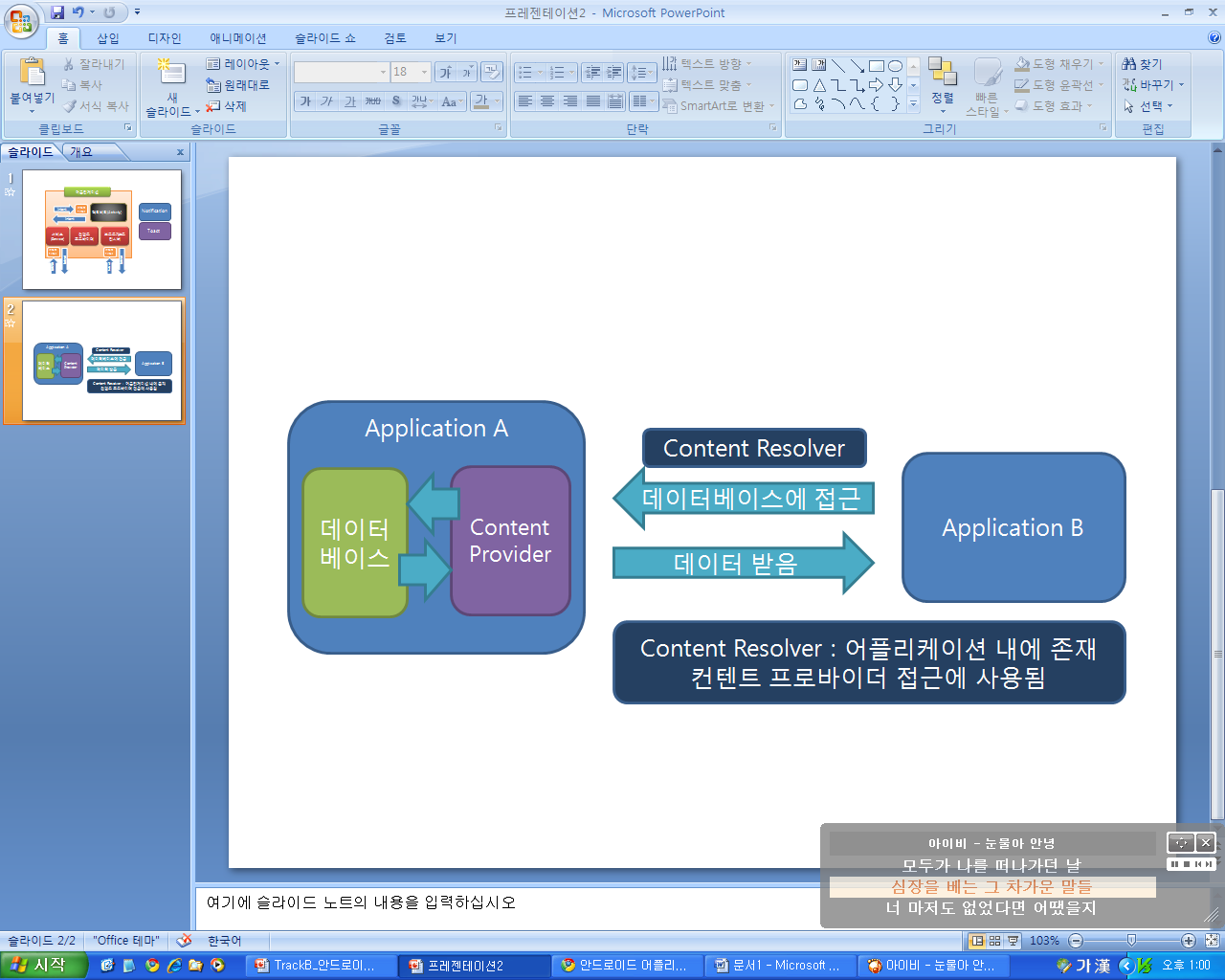
|  |  |  |
| --- | --- | --- |
| [http://cfile5.uf.tistory.com/image/1207591B4AA163800A2A11](http://cfile5.uf.tistory.com/original/1207591B4AA163800A2A11)  전화번호부 조회 화면 | [http://cfile27.uf.tistory.com/image/1807711B4AA1638108A5D8](http://cfile27.uf.tistory.com/original/1807711B4AA1638108A5D8)  전화부 편집 화면 | [http://cfile7.uf.tistory.com/image/1107691B4AA16381318E03](http://cfile7.uf.tistory.com/original/1107691B4AA16381318E03)  전화부 선택 화면 |

액티비티는 어플리케이션 컴포넌트 중 가장 중요한 역할, 사용자의 입력을 받고 사용자에게 그 결과를 보여주는 등 실질적으로 사용자와 상호작용을 하는 역할을 맡고 있습니다. 따라서, 액티비티 구성은 어플리케이션 개발에서 가장 신경써야 할 부분이기도 합니다.   
  
기본적으로 안드로이드가 휴대용 기기에서 돌아가는 것을 목적으로 하는 플랫폼이다보니, PC와 비교해 메모리가 부족할 수 밖에 없습니다. 그러다보니, 한 번에 여러 가지의 작업을 하다가 메모리가 부족해지면 우선순위가 낮은 작업은 메모리를 반환하여 다른 작업을 수행할 수 있도록 합니다.  
  
메모리 관리를 위해 액티비티는 액티비티 생애 주기(Activity Lifecycle)에 따라 사용자와 상호작용을 하지 않는 액티비티를 비활성화 시키거나 종료시킬 수 있습니다. 이러한 특성 때문에 액티비티를 구성할 때 각 액티비티의 상태변화에 따라 적절한 처리를 해주는 것이 중요합니다.

**서비스(Service)**는 화면에 표시되며 사용자와 직접 상호작용을 하는 액티비티와는 달리, 화면에 표시되지 않고 백그라운드에서 실행되는 컴포넌트입니다.   
  
앞서 언급했듯이, 액티비티는 액티비티 생애 주기에 의해 사용자와 상호작용을 하지 않는 액티비티는 비활성화 되거나 소멸될 수 있습니다. 예를 들면, 음악을 재생하는 어플리케이션에서 음악 재생을 담당하는 부분이 액티비티에 구현되어 있다면 액티비티가 화면에서 없어짐과 동시에 해당 액티비티는 더이상 사용자와 상호작용을 하지 못하므로 액티비티 생애주기에 따라 비활성화 상태(Inactive)가 되므로 음악 재생이 중지됩니다. 

[](http://cfile6.uf.tistory.com/original/1361571F4AA16C810C7201)

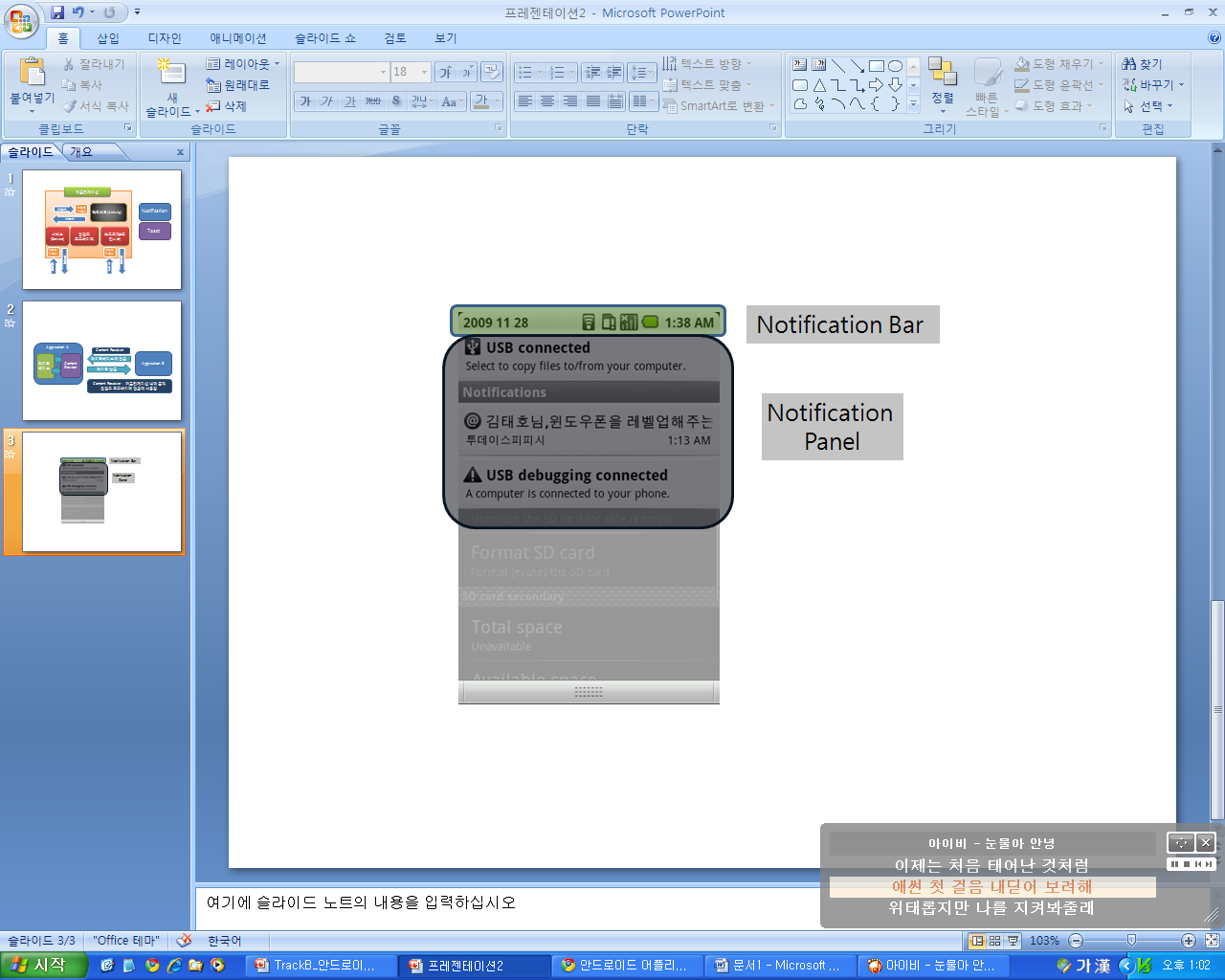
음악을 재생하는 어플리케이션은 해당 어플리케이션이 화면에 표시되고 있지 않을 때, 즉 사용자와 상호작용을 하지 않고 있더라도 백그라운드에서 계속 음악을 재생할 수 있어야 합니다. 따라서, 일반적으로 음악 재생 어플리케이션에서 음악을 재생하는 부분은 서비스 부분에 구현하여 사용자와 상호작용이 중단되더라도 지속적으로 음악을 재생할 수 있게 합니다.  
  
**컨텐트 프로바이더(Content Provider)**는 어플리케이션 내의 데이터 (전화번호부, 메모 등..)를 다른 어플리케이션과 공유할 수 있도록 해줍니다.   
  
어플리케이션 내의 데이터들은 기본적으로 해당 어플리케이션에서만 접근할 수 있고, 다른 어플리케이션은 접근할 수 없습니다. 그래서 특정 어플리케이션 내의 데이터 (예: 전화번호부 데이터)에 다른 어플리케이션에서도 접근할 수 있게 하려면 컨텐트 프로바이더를 정의하여 다른 어플리케이션에서 해당 데이터에 접근할 수 있는 범위, 방식 등을 정의해주어야 합니다. 즉, 컨텐트 프로바이더가 어플리케이션의 데이터에 접근하는 통로가 되는 셈이죠.



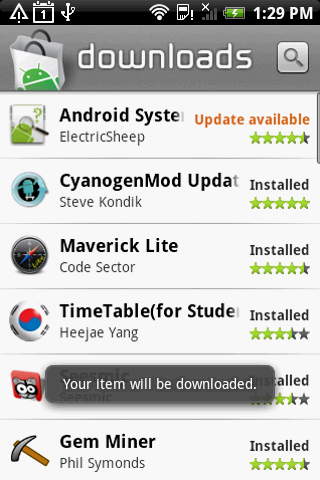
일반적으로 컨텐트 프로바이더를 이용하여 데이터를 공유할 어플리케이션에서 공유를 하고 싶은 데이터만을 선택하여 공유를 할 수 있도록 지정하게 됩니다. 이 과정에서 해당 컨텐트 프로바이더로 접근할 수 있는 고유의 주소를 정의하게 되며, 컨텐트 리졸버(Content Resolver)에 이 주소를 넘겨서 공유된 데이터에 접근할 수 있습니다.

[](http://cfile25.uf.tistory.com/original/1561AB1F4AA16E1F1E3693)

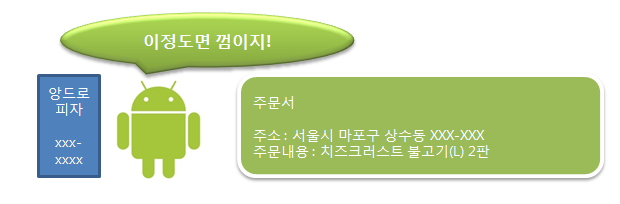
컨텐트 프로바이더는 마치 싸이월드 미니홈피와 비슷하다고 할 수 있습니다. 요즘은 사진 등을 올릴 때 최소 일촌공개로 사진을 올리는 경우가 많습니다. 물론 일촌공개인 사진 외에 자신만 볼 수 있는 비공개 사진들도 있겠구요. 즉, 일촌공개도 결국은 비공개인 사진 중 일부를 볼 수 있도록 공개해 주는 것이라 볼 수 있습니다. 이와 같이 컨텐트 프로바이더도 어플리케이션의 데이터에 접근할 수 있는 중재자 역할을 하며, 그 범위까지 지정해주는 역할을 수행합니다.  
  
 **브로드캐스트 리시버(Broadcast Receiver)**는 주로 시스템의 상태에 관련된 메시지 (배터리 부족, 언어 변경됨 등..) 혹은 어플리케이션의 메시지 (파일 다운로드 완료 등..)에 응답하는 역할을 합니다. 브로드캐스트 리시버는 해당 이벤트가 발생하였을 경우 그를 잡아내는 능력만 있을 뿐, 이를 사용자에게 알릴 방법은 없습니다. 때문에, 사용자에게 해당 이벤트를 알리기 위해 **알림(Notification) 혹은 토스트(Toast)**을 사용합니다.  
  
**알림(Notification)**은 NotificationManager를 통해 상태 표시줄 및 Notification Panel을 통해 알림 메시지를 띄워주거나,  LED를 점멸시키거나, 진동 발생 혹은 벨소리를 울리게 하여 사용자에게 해당 이벤트를 알립니다. 이러한 알림(Notification)은문 문자메시지 도착, 다운로드 완료 등을 사용자에게 알리는 데 사용됩니다.



**토스트(Toast)**는 Notification과 달리 간단한 내용을 사용자에게 알리기 위해 사용합니다. 특별한 기능 없이 단지 메시지를 화면에 잠깐 띄워주고, 곧 사라집니다.

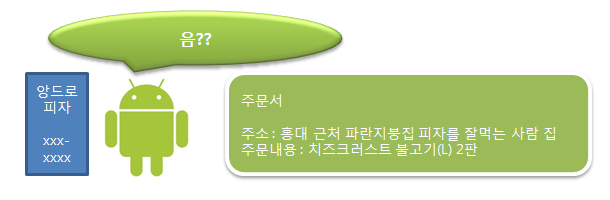


마지막으로, 인텐트(Intent)에 대해 알아보도록 하겠습니다.  
  
**인텐트(Intent)**는 어플리케이션의 컴포넌트, 구체적으로 말하자면 액티비티, 서비스, 브로드캐스트 리시버를 호출하거나 해당 컴포넌트를 호출함과 동시에 필요한 데이터를 전달해주는 역할을 합니다. 일종의 "전달 매체"라고 하면 적절하겠네요.   
  
인텐트 내부에는 대상 컴포넌트에 대한 정보가 담겨있습니다. 인텐트는 담고 있는 정보의 유형에 따라 명시적 **인텐트(Explicit Intent)**와 **암시적 인텐트(Explicit Intent)**로 나뉩니다.   
  
**명시적 인텐트(Explicit Intent)**는 호출하거나 데이터를 전달한 컴포넌트가 정확히 명시되어 있는 인텐트입니다. 대상 컴포넌트가 명확할 경우 컴포넌트의 이름을 인텐트에 넣어주면 인텐트에 의해 해당 컴포넌트가 호출되거나 해당 컴포넌트에 데이터가 전달되게 됩니다.

[](http://cfile26.uf.tistory.com/original/155DF21D4AA17C720F1861)

정확한 주소가 나와있군요!

**암시적 인텐트(Implicit Intent)**는 호출하거나 데이터를 전달할 컴포넌트를 정확히 명시하지 않고, 대상 컴포넌트의 특징들만 나열한 인텐트입니다. 예를 들면, 메모를 수정하는 인텐트에는 메모를 수정하는 기능을 가진 컴포넌트의 이름이 아닌 [메모를 수정할 수 있는 기능]이 들어가게 됩니다. 

[](http://cfile25.uf.tistory.com/original/1819FF1A4AA17C7B25910D)

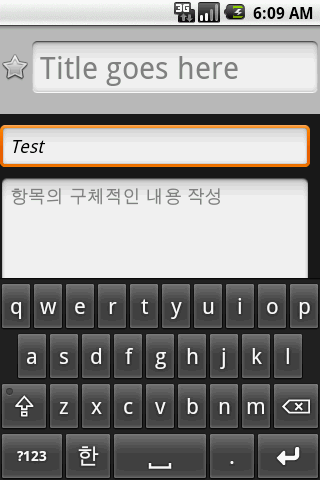
목적지의 특성이 나열되어있군요 -\_-^

이러한 암시적 인텐트를 처리하려면 각 컴포넌트에 해당 컴포넌트가 처리할 수 있는 작업을 명시해주어야 호출 대상 컴포넌트가 인텐트를 받아 호출될 수 있습니다. 이렇게 인텐트를 해석하여 적절한 컴포넌트를 찾는 과정을 **인텐트 해석(Intent resolving)**이라 합니다. 인텐트 해석(Intent resolving)에 필요한 정보는**메니페스트 파일**인 AndroidManifest.xml 의 각 컴포넌트 태그 내의 인텐트 필터 (Intent-filter) 내에 명시되며, 이곳에는 각 컴포넌트가 처리할 수 있는 작업의 유형 뿐만 아니라 해당 컴포넌트에서 처리할 수 있는 데이터의 유형까지 명시되어 있습니다.

|  |
| --- |
| Android Fundamentals : 액티비티 생애주기 (Activity Lifecycle) |

안드로이드는 모바일 기기에서 구동되는 것을 목적으로 하고 있기에 PC에서 구동되는 프로그램들에 비해 더 효율적인 메모리 관리가 필요합니다.   
  
안드로이드 어플리케이션을 구성하는 구성요소 중 하나인 액티비티(Activity)도 효율적인 메모리 관리를 위해 액티비티가 생성되고 소멸되는 조건인 생애주기(Lifecycle)을 가지고 있습니다. 액티비티 뿐만 아니라 브로드캐스트 리시버(Broadcast Receiver), 서비스(Service)도 일정한 생애주기를 갖고 있지만, 가장 자주 쓰이는 구성요소인 액티비티의 생애주기부터 알아보도록 하겠습니다.  
  
액티비티의 상태는 크게 **활성(Active), 일시정지(Paused), 정지(Stopped)** 이 3가지로 나눌 수 있으며, 각 상태에 해당하는 생애주기들이 존재합니다. 우선, 액티비티의 상태에 대해 알아보도록 하겠습니다.

* **활성(Active)**  
  현재 화면에 액티비티가 표시되는 상태이며, 사용자와 상호작용(버튼을 누르거나 글자를 입력하는 등..)을 할 수 있는 상태를 말합니다. 일반적으로 액티비티가 화면에 표시되고 있을 때 액티비티는 활성 상태입니다.

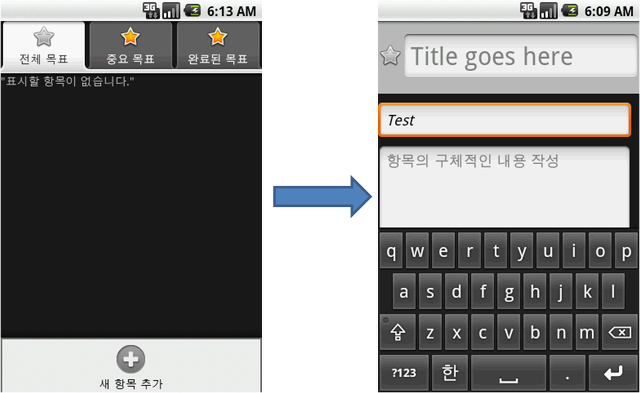
[](http://cfile1.uf.tistory.com/original/1539F80F4AAC0DD52BBD4A)

액티비티 내의 EditText 위젯에 텍스트를 입력하는 등, 사용자와 상호작용이 가능합니다.

* **일시정지(Paused)**  
  화면에서 액티비티가 보이지만, 사용자와 상호작용은 할 수 없는 상태입니다. 즉, 배경이 투명한 액티비티나 화면 전체를 가리지 않는 다른 액티비티에 의해 액티비티의 일부가 가려진 경우가 해당됩니다. 다이얼로그 또한 액티비티의 일부를 가리지만, 다이얼로는 액티비티의 일부이기 때문에 이 때에는 일시정지 상태로 바뀌지 않습니다.

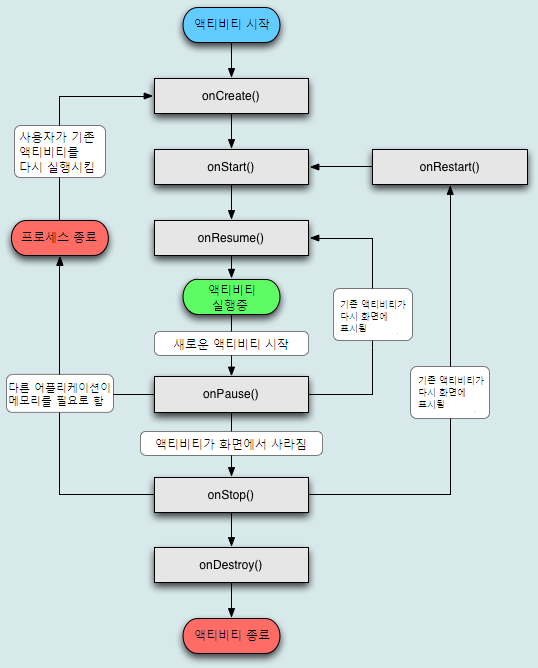
|  |  |
| --- | --- |
| [http://cfile24.uf.tistory.com/image/147E4F0E4AAC0D562D3EFE](http://cfile24.uf.tistory.com/original/147E4F0E4AAC0D562D3EFE)  다이얼로그가 화면을 가린 경우에는 Paused 상태가 되지 않습니다. | [http://cfile27.uf.tistory.com/image/123691274B10C22838BCB0](http://cfile27.uf.tistory.com/original/123691274B10C22838BCB0)  배경이 투명한 액티비티가 화면을 가린 경우 가려진 액티비티는 Paused 상태가 됩니다. |

* **정지(Stopped)**  
  다른 액티비티에 의해 완전히 가려진 상태입니다. 액티비티 A가 액티비티 B를 호출해서 액티비티 B가 화면에 표시된 경우, 액티비티 A의 모습은 화면에서 더이상 보이지 않게 되겠지요? 따라서 액티비티 A는 정지 상태가 됩니다.

[](http://cfile4.uf.tistory.com/original/1748CF0C4AAC0FA02BF84B)

액티비티의 상태는 위와 같이 크게 3가지로 나뉘지만, 안을 살펴보면 아래와 같이 7개의 상태로 나눠지게 됩니다. 아래의 메소드들이 호출됨에 따라 액티비티의 상태들이 바뀌게 되며, 액티비티를 구현할 때 해당 메소드를 오버라이드한 후 각 상태에 따라 처리해줄 작업을 지정해주면 됩니다.

* **void onCreate(Bundle savedInstanceState)**
* **void onStart()**
* **void onRestart()**
* **void onResume()**
* **void onPause()**
* **void onStop()**
* **void onDestroy()**

[](http://cfile25.uf.tistory.com/original/154FD6274AA18CD7208CC0)

액티비티 생애주기(Activity Lifecycle)

위 메소드들의 이름을 통해 onCreate, onStart.. 등 이 메소드들이 호출되는 조건들을 어렴풋이 짐작할 수 있습니다. 하지만 어렴풋이 아는 것 만으로는 안되겠죠?   
  
액티비티는 액티비티의 생애주기에 따라 상태가 변하게 되는데, 액티비티의 상태를 변화시키기 위해 각 상태에 해당하는 메소드가 호출됩니다. 따라서, 어플리케이션을 작성할 때 액티비티의 생애주기에 따라 알맞는 작업을 처리해주기 위해 각 상태에 해당하는 메소드를 오버라이드하여 처리할 작업을 넣어주게 됩니다.   
  
예를 들면, 데이터베이스를 사용하는 어플리케이션에서는 메인 액티비티가 실행될 때 데이터베이스를 열고 사용할 준비를 해야 합니다. 그리고, 데이터베이스를 사용하는 액티비티가 소멸될 때에는 사용하던 데이터베이스를 닫아줘야 합니다. 만약 이러한 작업을 적절히 처리해주지 않으면 데이터베이스를 닫아주지 않아 에러가 발생하는 등 여러가지 문제가 생길 것입니다.

그럼, 액티비티의 생애주기 내에서 액티비티의 상태가 어떻게 변하는지, 어떤 메소드가 호출되면 액티비티의 상태가 변하게 되는지 알아보도록 합시다.

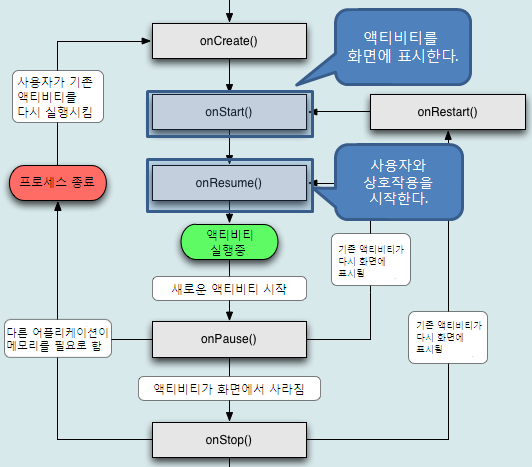
**액티비티의 시작**

액티비티를 처음 시작하려면, 액티비티를 만들어 줘야겠지요? 객체를 사용하려면 객체의 생성자를 이용하여 새로운 인스턴스를 만들어주듯이, 액티비티 또한 이러한 과정이 필요합니다.

액티비티의 생성은 **onCreate(Bundle savedInstanceState)** 메소드가 호출을 통해 이루어집니다. 액티비티가 생성되니까 onCreate()메소드가 호출되는 것은 알겠는데, 인자로 받는 **savedInstanceState**는 뭔지 잘 모르겠다구요? - 요놈은 잠시 후에 설명할테니, 궁금해도 조금만 참아주세요. ^^;

**onCreate()** 메소드에서는 **액티비티의 "초기화"**를 담당하는 코드들을 처리해줍니다. 액티비티에서는 기본적으로 각종 버튼이나 리스트, 체크박스, 입력 표시줄(EditText) 위젯들이 배치되어 있는 레이아웃을 구성하므로, 그러한 위젯들이 사용자와 상호작용을 할 수 있도록 해주는 코드들을 담고 있어야 합니다. 예를 들면, 버튼을 누르면 다른 액티비티가 뜨게 하거나, 아니면 입력 표시줄(EditText)에서 받은 데이터를 처리하는 것 등이 될 수 있겠죠.

**onCreate()** 메소드를 통해 액티비티를 시작하기 위한 준비를 모두 끝마쳤다면, 액티비티를 시작해야겠지요? 액티비티의 시작은onStart() 메소드를 통해 이루어집니다. 하지만, 이 상태는 완전히 액티비티가 시작된 상태가 아니라 액티비티가 화면에 표시되기 "직전"의 상태입니다. 이어서 onResume()메소드가 호출되면 액티비티가 화면에 표시되고, 사용자와 상호작용을 하기 직전의 상태가 됩니다. 액티비티가 생성된 후 바로 액티비티를 실행하면 간단할 것 같은데, 왜 굳이 복잡하게 onStart()와 onResume()으로 나누었을까요?

[](http://cfile24.uf.tistory.com/original/1377F21B4AB2836A0F6748)

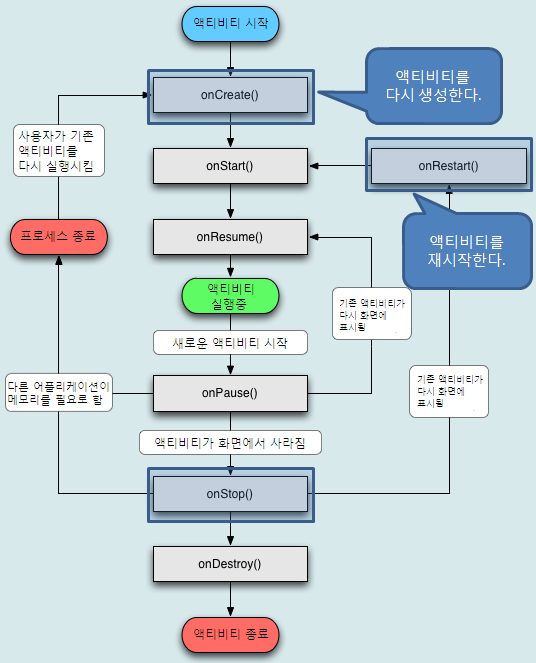
onStart()와 onResume()의 비교

 액티비티 생애주기 그림을 다시 보도록 하죠. onStart()와 onResume()의 위치가 다른 것을 볼 수 있습니다.   
  
onStart()는 액티비티가 실행되다가 다른 액티비티가 화면에 표시되어 정지 상태가 되면서 onStop()이 호출되었다가, 메모리에서 사라지기 전에 다시 정지되었던 액티비티가 호출되어 onRestart()를 거쳐 오거나, 혹은 메모리가 부족하여 프로세스가 종료되었다가 다시 액티비티를 생성하고, (onCreate() 호출) 액티비티를 화면에 표시하면서 onStart()를 호출하게 됩니다. 즉, onStart()에서는 정지(Stopped) 상태에서 액티비티가 복귀될 때 호출된다고 할 수 있습니다. 이는 한마디로 **액티비티가 화면에서 보이지 않다가 다시 나타나는 것**을 의미합니다.

반면, onResume()은 액티비티가 일시정지(Paused) 상태에서 복귀될 때 호출됩니다. 즉, 이는 **액티비티가 사용자와 상호작용을 하지 못하다가 다시 상호작용을 할 수 있게 되는 것**을 의미합니다.  
  
이렇게, 액티비티가 대기 상태에 있다가 복귀되는 시점 및 복귀되는 기능이 다르기 때문에 onStart()와 onResume()처럼 시작 과정이 복잡하게 나누어져 있는 것입니다.

**액티비티의 정지/일시정지**

액티비티가 화면에 표시되고, 사용자와 상호작용을 할 수 있는 활성(Active)상태에서 다른 액티비티에 의해 가려지거나, 화면에서 사라지게 되면 그 액티비티는 **정지(Stopped)** 상태가 됩니다. onStop()메소드가 호출되면 해당 액티비티는 정지 상태가며, 이곳에서부터 액티비티의 운명은 갈리게 됩니다.

[](http://cfile30.uf.tistory.com/original/115CD61A4AB28B862122FA)

일반적으로 정지된 액티비티는 사용자가 다시 사용할 것을 대비하여 메모리서 바로 사라지지 않고 올라가 있다가 메모리가 부족하게 되면 메모리 확보를 위해 강제로 종료되게 되죠. 그 후, 사용자가 다시 그 액티비티를 실행시키게 되면 다시 액티비티를 생성한 후(onCreate() ) 액티비티를 실행시키게 됩니다. 반면에, 메모리 부족 등이 발생하지 않아 액티비티가 대기 상태로 메모리에 올려져 있을 때 다시 그 액티비티를 호출하게 되면 액티비티를 다시 만들 필요 없이 다시 화면에 표시하면(onRestart()) 되므로, 액티비티를 다시 시작하기만 하면 됩니다.  
  
액티비티가**일시정지(Paused)**되는 것은 액티비티가 화면에는 표시되고 있지만, 사용자와 상호작용을 하지 않아서입니다. 일시정지 상태에는 이것 외에도 중요한 것이 하나 더 있습니다. 위의 생애주기에서도 확인할 수 있듯이, 일시정지 상태에서도 메모리가 부족하면 액티비티가 종료될 수 있습니다.   
  
일시정지 상태에서는 대부분 사용자가 어떤 내용을 입력하고 있는 등, "어떠한 작업이 진행 중"일 가능성이 높습니다. 그런데, 메모리가 부족해 액티비티를 종료시키게 된다면 사용자가 입력했던 데이터들 (예:작성중이던 메시지 등..)이 손실됩니다.   
  
이를 방지하기 위해 메모리 부족 등으로 "어쩔 수 없이" 액티비티를 종료해야 하는 경우 onSaveInsstanceState() 메소드가 호출되며 액티비티의 상태를 Bundle 형태로 저장하게 됩니다. 따라서, onSavedInstanceState() 메소드는 항상 호출되지 않으며, 만약 호출된다면 프로세스가 종료되기 전, onPause() 이후 혹은 onStop() 이후에 호출될 수 있습니다.

onSaveInstanceState() 메소드를 통해 저장된 상태는 Bundle 객체 형태로 저장되어 있다가, 나중에 해당 액티비티를 다시 실행할 때 onCreate()메소드 및 onRestoreInstanceState()메소드에서 액티비티가 강제로 종료될 때의 상태로 액티비티를 복원하게 됩니다.

onCreate() 메소드와 onRestoreInstanceState() 모두 Bundle 객체를 인자로 받고 있으며, 실제로 액티비티가 강제 종료될 때 저장된 상태정보가 담겨있는 Bundle 객체의 정보를 동일하게 전달받게 됩니다. 따라서 액티비티 상태의 복구는 두 메소드 모두에서 처리해줄 수 있으며, 상황에 따라 적합한 메소드에서 상태를 복원해주는 코드를 넣어주면 됩니다.

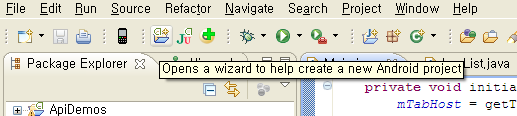
onRestoreInstanceState() 메소드 역시 항상 호출되는 메소드는 아니며, 호출된다면  onCreate() 메소드 이후에 호출되게 됩니다.

즉, 위의 두 메소드는 액티비티가 "강제로" 종료되었을 때 혹은 그 후 액티비티를 다시 실행시킬 때만 실행되므로, 만약 액티비티가 시작하고나 종료될 때 해 주어야 할 작업이 있다면 저 두 메소드를 오버라이드하는 대신 onCreate(), onStop() 및 onDestroy()메소드를 오버라이드한 후 그 안에 코드를 구현해야 안정적으로 해당 작업을 처리해줄 수 있습니다.

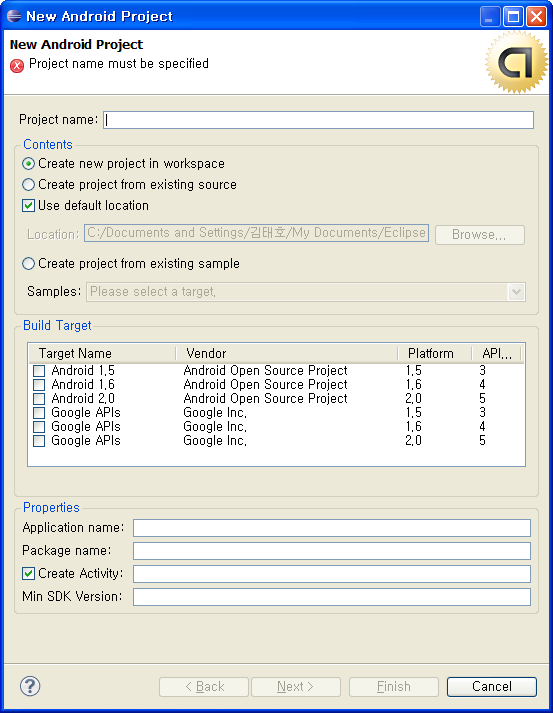
아래에 액티비티 생애 주기를 구성하는 주요 메소드의 역할을 간단히 정리해 보았습니다.  
[](http://cfile7.uf.tistory.com/original/160CDF244AB291630D6E28)

|  |
| --- |
| Exercises : 새 안드로이드 프로젝트 생성 |

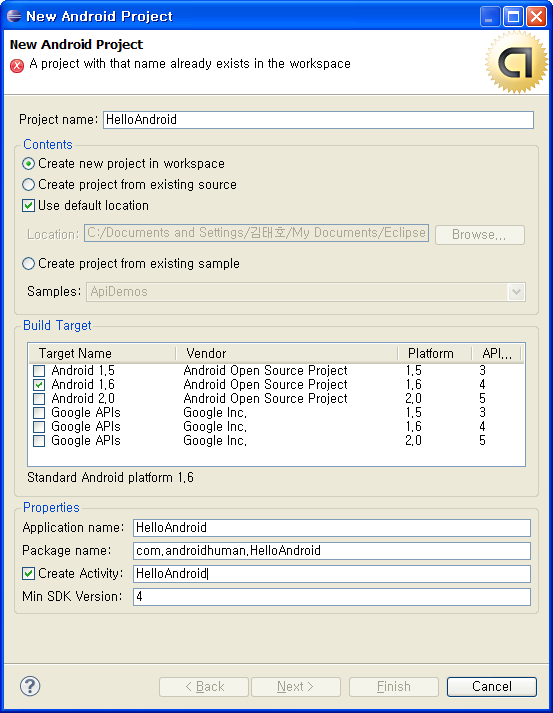
안드로이드 SDK도 설치하고, 이클립스도 설치하고,ADT까지 다 설치했다면?  
-> 이제 본격적으로 개발에 들어가는 것만 남았겠죠?   
  
이 글에서는 최근 출시된 안드로이드 SDK 2.0 과 이클립스용 안드로이드 개발 플러그인인 ADT 0.9.3 버전을 기준으로 하여 새로운 프로젝트를 만드는 과정에 대해 알아보도록 하겠습니다.  
  
자, 우선 새로운 프로젝트를 생성하려면 ... 이클립스부터 켜야겠지요? (...)

[](http://cfile24.uf.tistory.com/original/1613F1104AB553D6299C21)

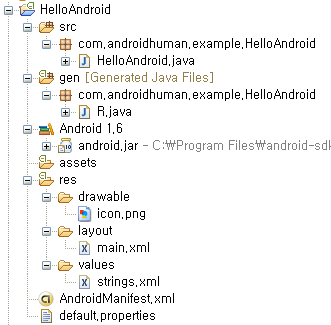
이클립스를 켠 후, New Android Project 버튼을 누릅니다.

[](http://cfile29.uf.tistory.com/original/1759A21A4B134C02A90DA2)

새로운 프로젝트를 생성하는 창이 나오면, 내용들을 하나하나씩(?) 입력해주면 됩니다.  
그리 어려운 내용이 아니니... 하나하나 보도록 하죠.

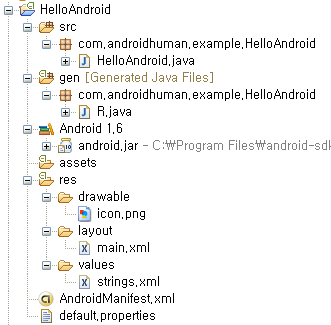
[](http://cfile9.uf.tistory.com/original/16569E1E4B134C143AA3F8)

**Project Name :**이름 그대로 프로젝트 이름을 입력해주시면 됩니다.  
  
**Build Target :**해당 프로젝트에서 사용할 SDK의 버전을 지정합니다. 좀 더 정확히 말하면 AVD(Android Virtual Device)의 버전을 선택하는 것입니다. Android 1.5/1.6은 해당 버전의 시스템 이미지를 의미하고, Google APIs는 해당 버전의 시스템 이미지에 Google maps가 포함되어 있음을 뜻합니다.  
  
**Application name :**  
어플리케이션 이름을 지정합니다.  
  
**Package name :**   
패키지명을 지정합니다. 패키지명을 짓는 원리에 대해서는 [2009/01/12 - [프로그래밍 이야기/끄적끄적] - 자바 패키지 이름을 짓는 원리??](http://androidhuman.tistory.com/entry/자바-패키지-이름을-짓는-원리) 를 참고하세요.  
  
**Create Activity :**  
체크표시를 한 후, 액티비티 명을 입력하면 프로젝트 생성과 동시에 액티비티가 만들어집니다.  
  
**Min SDK Version :**  
프로젝트의 최소 SDK 요구버전입니다. SDK의 업데이트가 계속 되고 있으므로, 최신 기술을 사용하는 경우 구버전의 시스템 이미지에서는 정상적으로 동작을 보장하지 않기에 이곳에 최소 버전을 명시하여야 합니다. 이 정보는 Target을 설정하게 되면 자동으로 입력됩니다.  
  
  
모든 정보를 입력한 후, Finish 버튼을 누르면 프로젝트가 생성됩니다.  
프로젝트 생성과 동시에 같이 생성된 파일들은 다음과 같습니다.

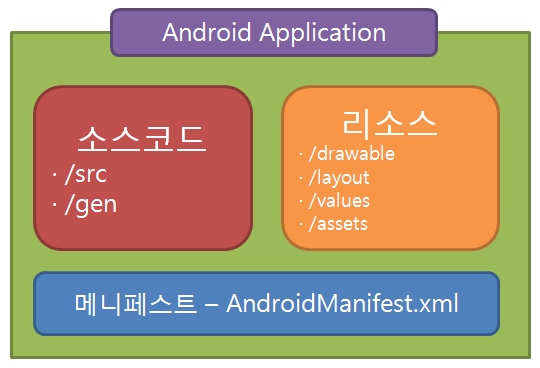
[](http://cfile10.uf.tistory.com/original/130AB0144AB552D30DA9DD)

위에서 생성된 파일들을 크게 구분한다면 소스파일과 리소스 파일로 구분할 수 있습니다.  
소스 파일은 우리가 실제로 어플리케이션에서 행해야 할 동작들을 명시하게 되고, 리소스 파일들은 여러 가지 데이터(그림, 음악, 레이아웃, 문자열 등등...) 의 데이터를 가지고 있습니다. 하나의 어플리케이션을 만드려면 이 두가지가 적절히 조합되어야 하죠.

|  |
| --- |
| Exercises : 프로젝트 생성파일 분석 |

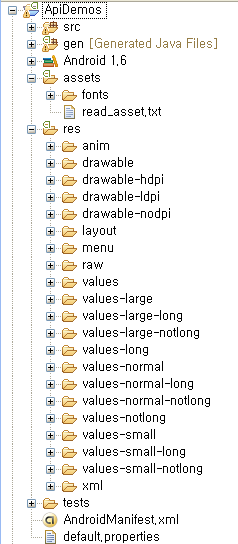
[[](http://cfile7.uf.tistory.com/original/131081224AC3CB4D42938D)](http://cfile7.uf.tistory.com/original/131081224AC3CB4D42938D)

새 프로젝트를 생성하게 되면, 위와 같은 폴더와 파일들이 생성됩니다. 폴더는 src, gen, assets, res 로 나누어지고, 그 외에 어플리케이션에 대한 전반적인 정보를 담고 있는 AndroidManifest.xml 파일 및 안드로이드 라이브러리를 볼 수 있습니다.   
  
안드로이드 어플리케이션을 [이론적 관점에서 보면](http://androidhuman.tistory.com/entry/안드로이드-어플리케이션의-구성요소)액티비티, 서비스, 컨텐트 프로바이더, Broadcast Receiver로 나눠지지만, 실제 개발환경 관점에서 보면 크게 **[(자바)소스코드], [리소스], [메니페스트]**로 나눌 수 있습니다.

[](http://cfile25.uf.tistory.com/original/164ECA1F4AC3CEEA110713)

개발환경 관점에서 본 안드로이드 어플리케이션의 구조

위의 그림에서는 리소스에 drawable, layout, values, assets만 표시하였지만, 실제로 안드로이드 어플리케이션에서 사용할 수 있는 리소스의 종류는 이보다 더 많습니다. SDK 폴더의 예제에 포함된 안에 있는 ApiDemos 어플리케이션의 리소스 구조만 봐도 알 수 있죠. 엄청 복잡합니다. -ㅁ-...

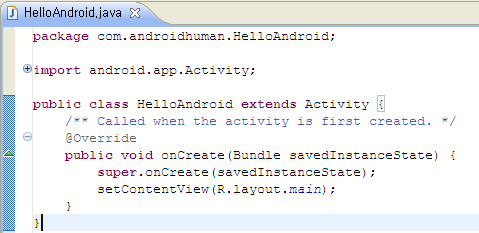
[](http://cfile8.uf.tistory.com/original/194ED41F4AC3D0331B5C90)

한눈에 봐도 복잡하기만 한 ApiDemos의 폴더구조..

폴더들을 보면, values, drawable 폴더도 있지만 values-large나 drawable-hdpi 등 다소 수상한(?) 제목을 가지고 있는 폴더들도 보이는군요. 이러한 폴더들은 **단말의 환경**(고해상도 액정, 저해상도 액정, 단말 로케일 등등..)에 따라 다른 리소스를 사용하기 위해 각 환경에 맞는 리소스들을 별도로 가지고 있는 것들입니다. 리소스에 대한 자세한 설명은 추후에 하기로 하고, 일단은 새로 만든 프로젝트에서 생성한 파일들에 대해 알아보도록 하겠습니다.

**소스코드(/src, /gen)**

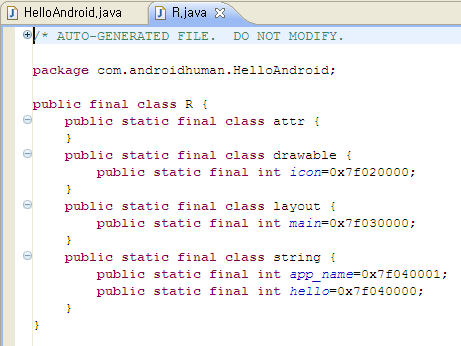
**1. 어플리케이션 컴포넌트 / 기타 클래스의 소스코드(/src)**  
일반 소스코드는 액티비티, 서비스, 컨텐트 프로바이더 혹은 기타 어플리케이션 구성에 필요한 클래스들의 소스 파일들로 구성되어 있습니다. 새 프로젝트를 생성한 경우, 일반적으로 자동으로 생성된 액티비티의 소스코드가 포함되죠. 우리가 만든 HelloAndroid 프로젝트를 보면, 자동으로 생성된 액티비티의 소스코드인 HelloAndroid.java가 있는 것을 확인할 수 있습니다.

[](http://cfile10.uf.tistory.com/original/16505C134AC3CA2832A250)

HelloAndroid 액티비티의 소스코드가 담긴 HelloAndroid.java 파일

**2. 자동 생성 파일(/gen)**  
  
안드로이드 어플리케이션은 소스코드 뿐만 아니라 레이아웃, 그림, 문자열 등 여러 가지 리소스들로 구성됩니다. 소스 코드를 작성하다 보면 레이아웃 파일에 정의된 위젯(예: TextView 등..)에 접근하여 작업을 수행(예:표시될 텍스트 내용을 변경)하는 등 리소스에 접근해야 하는 경우가 생깁니다.   
  
소스 코드에서 리소스 파일에 접근하기 위해 이클립스 ADT 플러그인에서는 리소스에 변동이 있을 때마다 리소스의 주소를 업데이트하여 R.java 파일에 기록해줍니다. 이러한 과정은 리소스가 추가/삭제되거나 변경된 후 해당 리소스가 "저장" 될 때 이루어지게 됩니다.

주의할 것은, 이러한 과정은 모두 자동으로 이루어지며, R.java 파일에는 리소스들의 주소가 담겨 있으므로 R.java 파일의 내용을 임의로 변경해서는 안됩니다. 

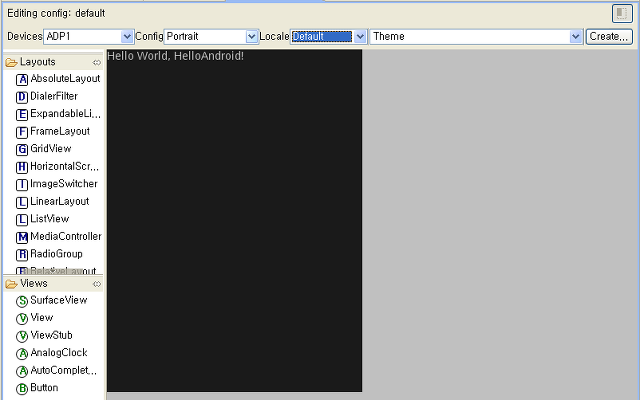
[](http://cfile9.uf.tistory.com/original/17505C134AC3CA283346E8)

R.java 파일의 내부 모습

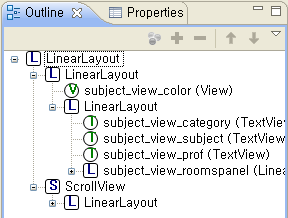
**리소스(/drawable, /layout, /values)**

**1. 그려질 수 있는 객체 (/drawable)**  
  
이 폴더에는 어플리케이션에서 사용할 그림 파일 등을 보관합니다. jpg, png 등의 파일 형식을 지원합니다. 우리가 위에서 만든 HelloAndroid 프로젝트에는 어플리케이션의 아이콘인 icon.png 만 들어가있습니다.   
리소스를 추가시키게 되면 해당 파일의 이름을 리소스명으로 등록하게 되는데, 리소스 이름으로 소문자 및 일부 한정된 특수문자만 지원하기에 **그림 파일 이름은 모두 소문자 및 일부 한정된 특수문자로만 이루어져야([a-z0-9\_.])**합니다. 이는 비단 그림파일에만 해당되는 것이 아니라, **리소스 전체(레이아웃 등)에 해당됩니다**,  
  
**2. 레이아웃 (/layout)**  
  
어플리케이션의 화면을 구성하는 레이아웃 파일들을 포함하고 있습니다. HelloAndroid 프로젝트에는 HelloAndroid 액티비티의 레이아웃인 main.xml 파일이 있는 것을 확인할 수 있습니다. 이클립스 ADT((Android Development Tools)를 통해 레이아웃 파일이 화면에 어떻게 표시되는지 볼 수도 있으며, Outline 및 속성 창을 이용하여 레이아웃을 쉽게 구성할 수도 있습니다. 아래는 main.xml 레이아웃 파일을 ADT를 통해 열어본 모습입니다.

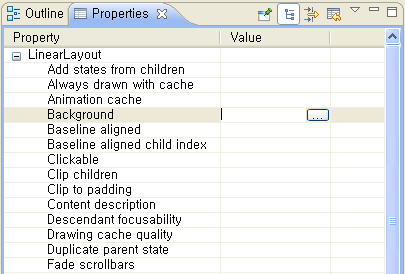
안드로이드 SDK가 2.0으로 업데이트됨에 따라, ADT의 버전도 0.9.4로 업데이트면서 레이아웃 에디터의 모습이 조금 달라졌습니다. 기존의 잡다한 옵션들 (사실 어떻게 쓰는지도 잘 모르겠....)은 싹 사라지고, 각 장치 및 환경에 해당 레이아웃이 적용된 모습을 미리 볼 수 있도록 바뀌었습니다. 현재는 ADP1 (Android Dev Phone 1)과 Ion (HTC Magic)만 있습니다.

[](http://cfile10.uf.tistory.com/original/1375EE154AE756601AC2D7)

ADT 0.9.4의 레이아웃 에디터

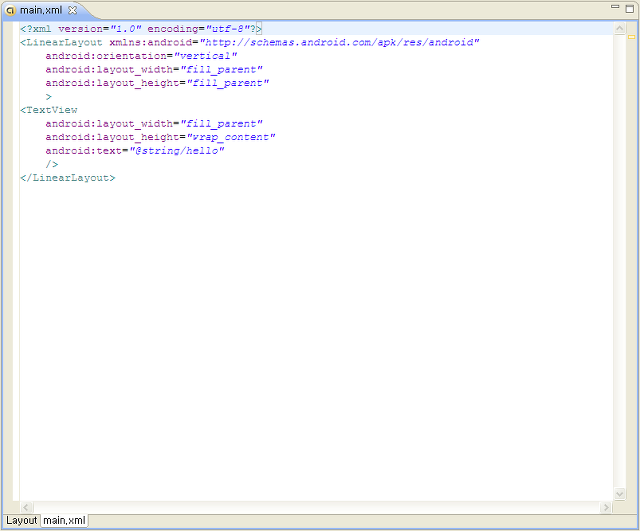
[](http://cfile4.uf.tistory.com/original/18222D0B4B06C8470CCBE1)

레이아웃의 구조가 표시되는 Outline 창

[](http://cfile26.uf.tistory.com/original/117CA00B4B06C848314189)

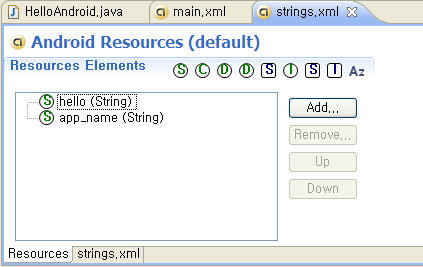
각 객체의 속성을 표시해주는 Properties 창

안드로이드 레이아웃 에디터ㅜ래부분의 탭의 [Layout]탭을 클릭하면 레이아웃 미리보기를, [main.xml]탭을 클릭하면 해당 레이아웃의 코드를 볼 수 있습니다.

[](http://cfile5.uf.tistory.com/original/142F290C4AC3DCB849BD26)

main.xml 탭을 클릭하면 나오는 코드 편집창

**3. 기타 여러가지 값들 (/values)**  
  
values 폴더에는 문자열, 배열 등 기타 어플리케이션에서 사용하는 여러가지 값들이 보관됩니다. HelloAndroid 프로젝트에는 문자열을 저장하는 파일인 strings.xml 파일만 있는 것을 확인할 수 있습니다. strings.xml 파일을 열어보면 아래와 같이 현재 리소스에 등록되어 있는 문자열과 함께 문자열들을 추가하거나 삭제할 수 있는 화면이 표시됩니다.

[](http://cfile26.uf.tistory.com/original/18505C134AC3CA283475E3)

또한, 레이아웃과 마찬가지로 아래의 strings.xml 탭을 선택하면 직접 XML 코드를 작성할 수 있는 화면이 표시됩니다.

[](http://cfile6.uf.tistory.com/original/19505C134AC3CA2835273B)

리소스 폴더에 구체적으로 어떤 파일들이 저장되는지 자세한 내용이 궁금하신 분은 [여기](http://developer.android.com/guide/topics/resources/resources-i18n.html)를 참조하세요.  
  
  
**4. 메니페스트 파일(AndroidManifest.xml)**  
  
메니페스트 파일인 AndroidManifest.xml 파일은 각 어플리케이션에 필수로 있어야 하는 파일입니다. 이 파일은 어플리케이션의 이름, 버전부터 시작하여 사용 권한, 컴포넌트 정의 등 어플리케이션의 뼈대를 구성하는 정보를 포함하고 있습니다.

|  |
| --- |
| Exercises: Layout 파일을 이용한 액티비티 화면 구성 |

[HelloAndroid.java]

01.import android.app.Activity;

02.import android.os.Bundle;

03.

04.public class HelloAndroid extends Activity {

05./\*\* Called when the activity is first created. \*/

06.@Override

07.public void onCreate(Bundle savedInstanceState) {

08.super.onCreate(savedInstanceState);

09.setContentView(R.layout.main);

10.}

11.}

위의 코드는 HelloAndroid 프로젝트를 만들면서 기본으로 만들어진 코드입니다. 위의 코드를 실행하면 아래와 같은 화면이 표시되게 됩니다.

[](http://cfile5.uf.tistory.com/original/1625282A4AE2070C10FF4A)

어떻게 해서 위와 같은 화면이 나오는지 아직은 어리둥절할지도 모릅니다. 하지만 그리 어렵지 않으니, 한줄한줄 차근차근 알아가보도록 하죠.

1.public class HelloAndroid extends Activity

HelloAndroid 클래스가 Activity 클래스를 상속하는 것을 볼 수 있습니다. 새 프로젝트를 만들게 되면 기본적으로 액티비티가 하나 생성되게 되는데, 그 액티비티가 여기에서는 HelloAndroid 입니다. 따라서, HelloAndroid 클래스는 Activity 클래스를 상속받게 됩니다.

1.@Override

2.public void onCreate(Bundle savedInstanceState) {

3.super.onCreate(savedInstanceState);

4.setContentView(R.layout.simplelayout);

5.}

Activity 클래스 내의 메소드 중 하나인 onCreate() 메소드를 오버라이드합니다. [2009/09/18 - 액티비티의 생애주기(Lifecycle)](http://androidhuman.tistory.com/entry/액티비티의-생애주기Lifecycle)

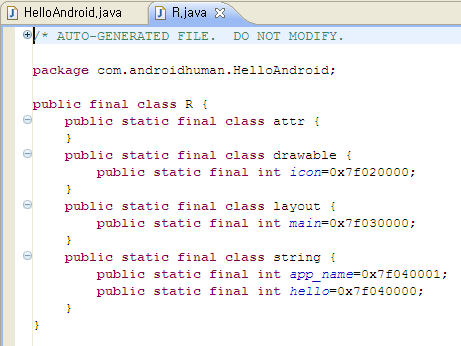
에서 다뤘듯이, onCreate() 메소드가 호출되면서 액티비티를 생성하게 됩니다. 따라서, 이곳에서 Hello, Android 액티비티의 초기화 작업, 텍스트를 화면에 표시해주는 작업을 수행해 주어야 합니다.

액티비티 클래스의 onCreate() 메소드를 오버라이드 하여 기본적인 onCreate()에서 하는 일 외에 우리가 하고자 하는 일을 추가하는 것이기 때문에, 상위 클래스의 onCreate()메소드를 호출(super.onCreate(savedInstanceState))하여 상위 클래스의 메소드 내에 정의된 작업을 먼저 수행하게 됩니다.

상위 클래스 메소드의 호출이 끝난 후에, setContentView() 메소드를 사용한 것을 볼 수 있습니다. 이 메소드는 리소스(레이아웃 파일)를 이용하여 액티비티의 화면을 구성할 때 필수적인 메소드이며, 인자로 레이아웃 리소스 혹은 View 객체를 받을 수 있습니다. 이 메소드에서는 인자로 받은 레이아웃 혹은 View 객체를 액티비티의 화면에 표시해주는 역할을 수행합니다.

**View(뷰)** (android.view.View)  
뷰는 액티비티, 즉화면을 구성하는 기본 단위입니다. 텍스트를 표시하는 TextView, 이미지를 표시하는 ImageView, 리스트를 표시하는 ListView 등 특정 데이터를 화면에 표시해주는 객체, Button, RadioButton 등 사용자의 동작에 반응할 수 있는 객체, LinearLayout, RelativeLayout 등 각자가 포함하고 있는 View 객체들이 어떻게 표시할지를 결정해주는 레이아웃 객체들로 구성됩니다.

위에서는 액티비티의 화면에 레이아웃 파일을 표시하도록 하고 있으며, 소스코드 내에서 레이아웃 파일에 접근하기 위해 리소스들의 주소가 담긴 R 클래스 (R.java)를 이용하는 것을 볼 수 있습니다.

[](http://cfile5.uf.tistory.com/original/194CA00B4AE20C64182048)

각 리소스를 참조할 수 있는 주소들이 담겨있다.

그럼, 이제 액티비티의 화면을 구성하게 될 main.xml 레이아웃 파일이 어떻게 구성되어있는지 보도록 하겠습니다.

01.<?xml version="1.0" encoding="utf-8"?>

02.<LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"

03.android:orientation="vertical"

04.android:layout\_width="fill\_parent"

05.android:layout\_height="fill\_parent"

06.>

07.<TextView

08.android:layout\_width="fill\_parent"

09.android:layout\_height="wrap\_content"

10.android:text="@string/hello"

11./>

12.</LinearLayout>

레이아웃 파일의 구조를 보면, **LinearLayout이 TextView를 감싸는 구조**를 하고 있음을 볼 수 있으며, 이는 LinearLayout이 포함하는 다른 View들이 어떻게 화면에 배치될지를 담당하기 때문입니다. LinearLayout의 속성 중 하나, orientation에 따라 레이아웃 내부의 View 수평 혹은 수직으로 배치하며, LinearLayout 외에도 포함하는 View들간의 위치 관계들을 정의한 후, 그에 따라 각 객체들을 표시해주는 RelativeLayout, 레이아웃의 왼쪽 위를 기준으로 포개듯이 객체들을 배치해주는FrameLayout 등 여러 레이아웃이 존재합니다.

layout\_height 및 layout\_width 속성은 LinearLayout 뿐만 아니라 TextView에도 있는 속성으로, 액티비티 화면 내에서 표시될 뷰의 너비 및 높이를 지정해줍니다.

fill\_parent 속성은 화면에 꽉 차게끔 View를 배치하고, wrap\_content 속성은 각 View를 그리는 데 필요한 만큼만 그 View가 화면을 차지함을 뜻합니다. 여기에서 유의해야 할 것은 fill\_parent는 해당 View가 속하는 View, 즉 여기에서 TextView의 경우 View들을 담고있는 View(LinearLayout)이 화면에서 차지하는 만큼 화면을 차지할 수 있습니다. 아래에서 TextView의 layout\_height가 fill\_parent일 때, LinearLayout의 높이(layout\_height)에 따라 TextView가 차지하는 영역이 달라지는 것을 볼 수 있습니다.

|  |  |
| --- | --- |
| [http://cfile1.uf.tistory.com/image/1205F70D4AE211B122C99D](http://cfile1.uf.tistory.com/original/1205F70D4AE211B122C99D)  <LinearLayout android:layout\_height = "fill\_parent" ... | [http://cfile27.uf.tistory.com/image/1305F70D4AE211B123F37F](http://cfile27.uf.tistory.com/original/1305F70D4AE211B123F37F)  <LinearLayout android:layout\_height = "wrap\_content" ... |

TextView는 텍스트를 표시하기 위해 text 속성을 통해 화면에 표시할 텍스트를 지정합니다. 이곳에 화면에 표시할 텍스트를 직접 넣어줄 수도 있고, 리소스에 저장된 문자열의 주소를 넣어 리소스에 저장된 값을 표시하도록 할 수도 있습니다. 위의 코드에서는 이름이 hello인 문자열의 값을 참조하는 모습을 볼 수 있습니다.

소스 코드에서 리소스 파일을 참조할 수 있는 것처럼, 리소스 파일에서도 다른 리소스 파일의 값을 참조하도록 할 수 있습니다. 리소스 내에서 다른 리소스의 값을 참고하기 위해, **@[리소스 종류]/[리소스 이름]** 형식의 주소를 사용합니다.

|  |
| --- |
| Exercises : Java Code를 이용한 액티비티 화면 구성 |

안드로이드 액티비티의 화면 구성 방법은 크게 두 가지가 있습니다. 하나는 앞에서 알아보았던 것처럼 레이아웃을 이용하여 화면을 구성하는 방법이고, 다른 하나는 이번 글에서 살펴볼 자바 코드를 통해 화면을 구성하는 방법입니다.일반적으로 정적인 레이아웃을 구성할 때는 레이아웃 파일을 이용하고, 동적인 레이아웃을 구성할 경우 자바 코드를 통해 레이아웃을 구성하게 되며 이 두 가지 방법을 모두 사용하여 레이아웃을 구성하는 것 또한 가능합니다. (기본 레이아웃을 구성한 후, 사용자의 동작에 따라 View들을 추가해주는 등의 레이아웃)

이번 글에서는 자바 코드만을 이용하여 저번 강좌의 Hello, Android와 동일한 화면을 구성하는 방법에 대해 알아보도록 하겠습니다. 아래는 저번 강좌에서 사용했던 레이아웃 파일입니다.

01.<?xml version="1.0" encoding="utf-8"?>

02.<LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"

03.android:orientation="vertical"

04.android:layout\_width="fill\_parent"

05.android:layout\_height="fill\_parent"

06.>

07.<TextView

08.android:layout\_width="fill\_parent"

09.android:layout\_height="wrap\_content"

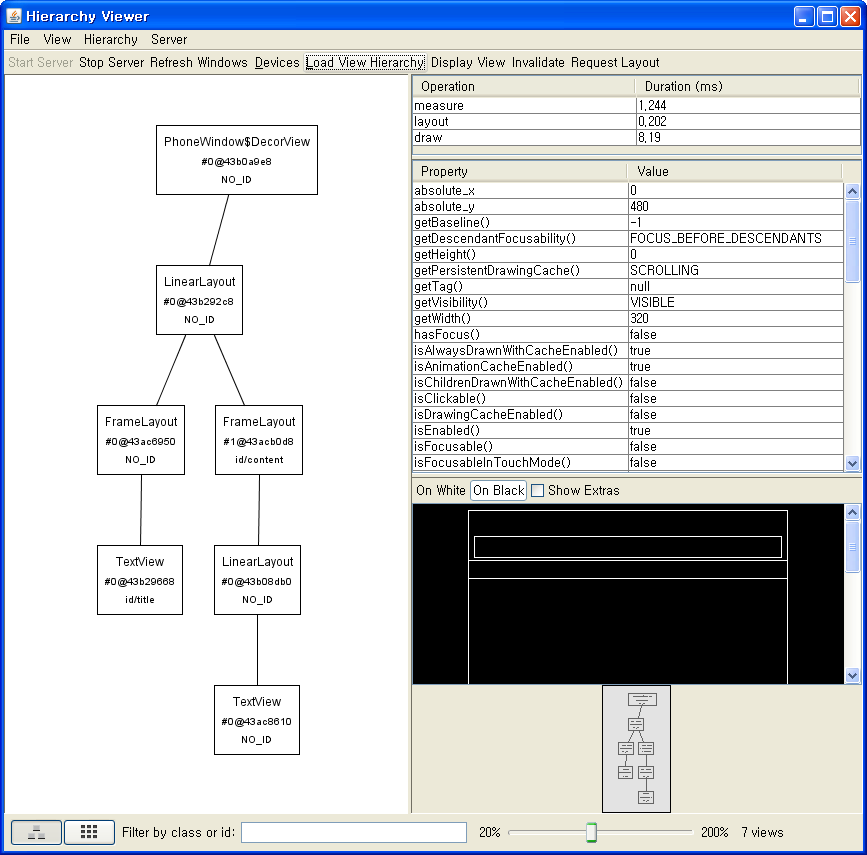
10.android:text="@string/hello"

11./>

12.</LinearLayout>

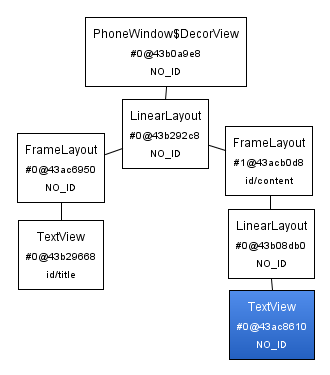
Hello, Android의 레이아웃은 LinearLayout을 바탕으로 하고 있으며, 그 안에 텍스트를 표시해주는 뷰인 TextView가 들어가 있습니다. TextView를 감싸고 있는 LinearLayout의 너비(layout\_width) 및 높이(layout\_height)는 각각 fill\_parent, 즉 화면을 꽉 채우게 설정되어 있습니다. fill\_parent 속성은 해당 뷰를 포함하는 부모 객체가 가진 너비 혹은 높이만큼의 크기를 가진다는 것인데, 여기에서 LinearLayout은 그 자체가 최상위 객체, 루트(root) 객체 이므로 액티비티의 화면을 꽉 채우게 됩니다.

한번, 화면에 표시되는 레이아웃이 어떤 구조를 가지고 있는지 알아보도록 하겠습니다. [SDK 설치 경로]/tools 폴더의 Hierarchy Viewer를 실행시켜 보면, 액티비티의 레이아웃이 어떤 구조를 가지고 있는지 더 잘 알 수 있습니다. HelloAndroid 어플리케이션이 실행된 상태에서 Hierarchy Viewer를 실행시킨 후, Load View Hierarchy 버튼을 누르면 아래와 같이 현재 화면의 View에 대한 정보들이 표시됩니다.

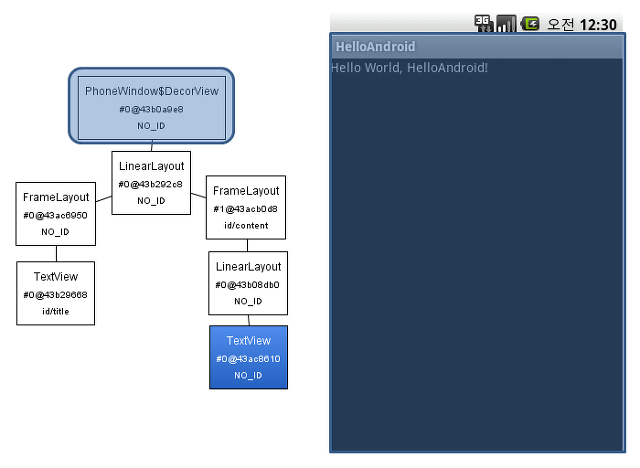
[](http://cfile29.uf.tistory.com/original/204FEE174AFAF13C2B71A8)

Hierarchy Viewer의 모습

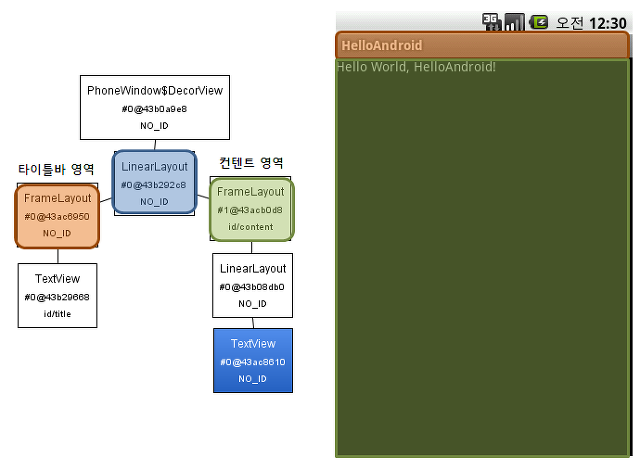
아래는 Hello, Android의 뷰 구조입니다.

[](http://cfile25.uf.tistory.com/original/2040DF194AFAF17A35A053)

우리가 생각했던 구조는 LinearLayout이 TextView를 감싸고 있는 구조였는데, 생각보다 복잡해 보이는군요. 이는 화면에 표시되는 모든 View의 구조가 다 표시되었기 때문입니다. 별로 어렵지 않으니 하나하나씩 보도록 합시다.

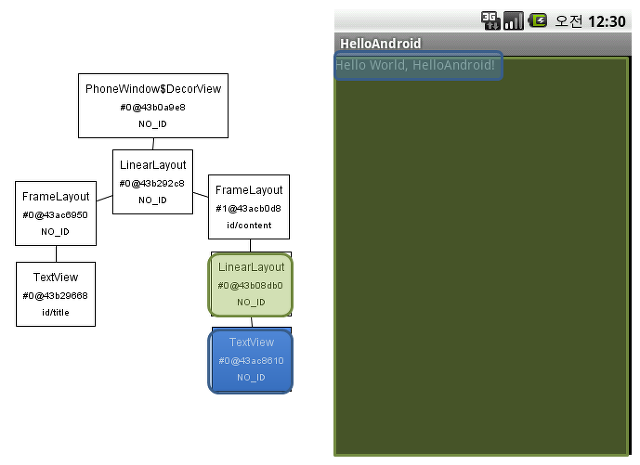
[](http://cfile10.uf.tistory.com/original/15796D1F4AFAF1F5019D82)

우선, 가장 상위에 위치한 개체, 루트 개체인 PhoneWindow$DecorView는 상태 표시줄을 제외한 실질적으로 액티비티 화면이 표시되는 모든 영역을 뜻합니다. 이는 크게 타이틀바 (액티비티 이름이 표시되는 영역)과 실제 액티비티의 내용들이 표시되는 영역으로 나뉩니다.

[](http://cfile5.uf.tistory.com/original/17796D1F4AFAF1F602BE6C)

루트 객체의 바로 아래에는 타이틀바와 컨텐트 영역을 포함하는 LinearLayout이 있습니다.

각 영역은 FrameLayout을 바탕으로 구성됩니다. FrameLayout은 레이아웃이 포함하는 뷰들을 왼쪽 가장 윗쪽을 기준으로 차곡차곡 쌓아주는 형식으로 배치합니다. 이러한 각 영역 안에 실질적으로 타이틀바에 표시되는 텍스트 및 액티비티 화면에 표시될 레이아웃 객체들이 들어가게 됩니다.

[](http://cfile6.uf.tistory.com/original/19796D1F4AFAF1F703B868)

id가 content인 FrameLayout, 즉 컨텐트 영역의 하위에 우리가 작성했던 LinearLayout 및 TextView가 있는 것을 볼 수 있습니다. 이처럼, 간단해 보이는 화면도 사실은 이렇게 여러 구조를 가지고 있습니다.

fill\_parent 속성이 부모 객체의 크기만큼의 크기를 갖는 것이니, 여기에서는 LinearLayout의 부모 객체, FrameLayout의 크기만큼의 크기를 갖고 이 FrameLayout은 화면에서 가로 세로로 타이틀바 및 상태 표시줄을 제외한 모든 화면을 차지하도록 되어있으므로 이 FrameLayout 하위의 객체들은 최대 이 레이아웃의 크기만큼을 차지할 수 있게 됩니다.

이제 자바 실제 코드를 통해 레이아웃을 구성해보도록 하겠습니다. 우선, 아래의 코드에서부터 시작합니다.

1.public class HelloAndroid extends Activity {

2./\*\* Called when the activity is first created. \*/

3.@Override

4.public void onCreate(Bundle savedInstanceState) {

5.super.onCreate(savedInstanceState);

6.

7.}

8.}

우리가 구성할 화면은 LinearLayout 내부에 TextView가 들어가 있는 구조이므로, 우선 LinearLayout 객체를 생성해 주도록 합시다.

1.LinearLayout root = new LinearLayout(this);

그 다음은 LinearLayout이 어떻게 화면에 표시될 것인지를 지정해 주어야 합니다. 이러한 속성을 지정해 주기 위해 LinearLayout 클래스 내의 내부 클래스인 LayoutParams 객체를 사용하여 LinearLayout의 어떻게 화면에 표시될지 그 속성을 담고 있는 객체를 생성합니다.

1.LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(

2.ViewGroup.LayoutParams.FILL\_PARENT,

3.ViewGroup.LayoutParams.FILL\_PARENT,

4.0.0F);

**[API]**  
**public LinearLayout.LayoutParams (int width, int height, float weight)**  
  
주어진 너비, 높이 및 weight를 갖는 새 레이아웃 속성을 생성합니다.

LinearLayout이 컨텐트 영역에 꽉 차게 나오도록 width 및 height에 모두 fill\_parent 속성을 지정해줍니다. weight는 화면에 여러 뷰들이 표시될 때 해당 View가 화면에서 차지하는 비율로, 여기에서는 사용하지 않으므로 0을 넣어줍니다.

1.root.setLayoutParams(params);

2.root.setOrientation(LinearLayout.VERTICAL);

다음으로, LinearLayout의 레이아웃 속성으로 우리가 만든 앞서 만들어준 속성을 사용하기 위해 setLayoutParams() 메소드의 인자로 우리가 만들어준 레이아웃 속성(객체)을 넣어줍니다. 그리고, LinearLayout에서 View를 표시할 방향을 지정하기 위해 setOrientation() 메소드를 사용합니다. 여기에서는 Vertical, 레이아웃 내의 View들을 수직으로(위->아래) 표시하도록 하였습니다.

이제는 TextView를 LinearLayout에 추가할 차례입니다. TextView 객체를 생성한 후, setText()를 통해 TextView에 표시될 문자열을 지정합니다. 여기에서는 리소스에 저장되어있는 문자열을 불러왔습니다. 그 다음, LinearLayout과 마찬가지로 TextView의 레이아웃 속성을 지정합니다.

1.TextView tv = new TextView(this);

2.tv.setText(R.string.hello);

3.LinearLayout.LayoutParams tvParams = new LinearLayout.LayoutParams(

4.ViewGroup.LayoutParams.WRAP\_CONTENT,

5.ViewGroup.LayoutParams.WRAP\_CONTENT,

6.0.0F);

7.tv.setLayoutParams(tvParams);

TextView 객체의 설정이 완료되었으므로, 이 TextView를 LinearLayout에 추가시켜줍니다. 그 후, LinearLayout을 액티비티의 화면으로 표시하도록 설정합니다.

1.root.addView(tv);

2.setContentView(root);

완성된 코드는 아래와 같습니다.

01.package com.androidhuman.HelloAndroid;

02.

03.import android.app.Activity;

04.import android.os.Bundle;

05.import android.view.ViewGroup;

06.import android.widget.LinearLayout;

07.import android.widget.TextView;

08.

09.public class HelloAndroid extends Activity {

10./\*\* Called when the activity is first created. \*/

11.@Override

12.public void onCreate(Bundle savedInstanceState) {

13.super.onCreate(savedInstanceState);

14.LinearLayout root = new LinearLayout(this);

15.LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(

16.ViewGroup.LayoutParams.FILL\_PARENT,

17.ViewGroup.LayoutParams.FILL\_PARENT,

18.0.0F);

19.

20.root.setLayoutParams(params);

21.root.setOrientation(LinearLayout.VERTICAL);

22.

23.TextView tv = new TextView(this);

24.tv.setText(R.string.hello);

25.LinearLayout.LayoutParams tvParams = new LinearLayout.LayoutParams(

26.ViewGroup.LayoutParams.WRAP\_CONTENT,

27.ViewGroup.LayoutParams.WRAP\_CONTENT,

28.0.0F);

29.tv.setLayoutParams(tvParams);

30.

31.root.addView(tv);

32.

33.setContentView(root);

34.}

35.}

이 코드를 실행시키면 우리가 저번에 레이아웃 파일을 통해 구성했던 화면과 동일한 화면이 표시되게 됩니다.

여기까지 자바 코드를 통해 액티비티의 화면을 구성하는 것에 대해 알아보았습니다. 사실, 레이아웃을 구성할 때 자바 코드를 통해 구성하는 것보다는 레이아웃 파일을 이용하여 구성하는 것이 일반적으로 더 효율적입니다. 하지만, 동적으로 화면을 구성하는 경우 (사용자의 입력에 반응하여 View에 요소들이 추가되어야 한다는 둥...) 자바 코드를 이용하여 레이아웃을 구성해야 합니다.

이러한 특성 때문에 보통 기본 바탕이 되는 레이아웃은 레이아웃 파일을 통해, 그 이후는 자바 코드를 통해 레이아웃을 구성하게 됩니다. 따라서 레이아웃 파일을 이용하는 방법과 자바 코드를 이용하여 레이아웃을 구성하는 방법 모두 꼭 알고 있어야 합니다.

|  |
| --- |
| Exercises : 새로운 액티비티 생성하기 |

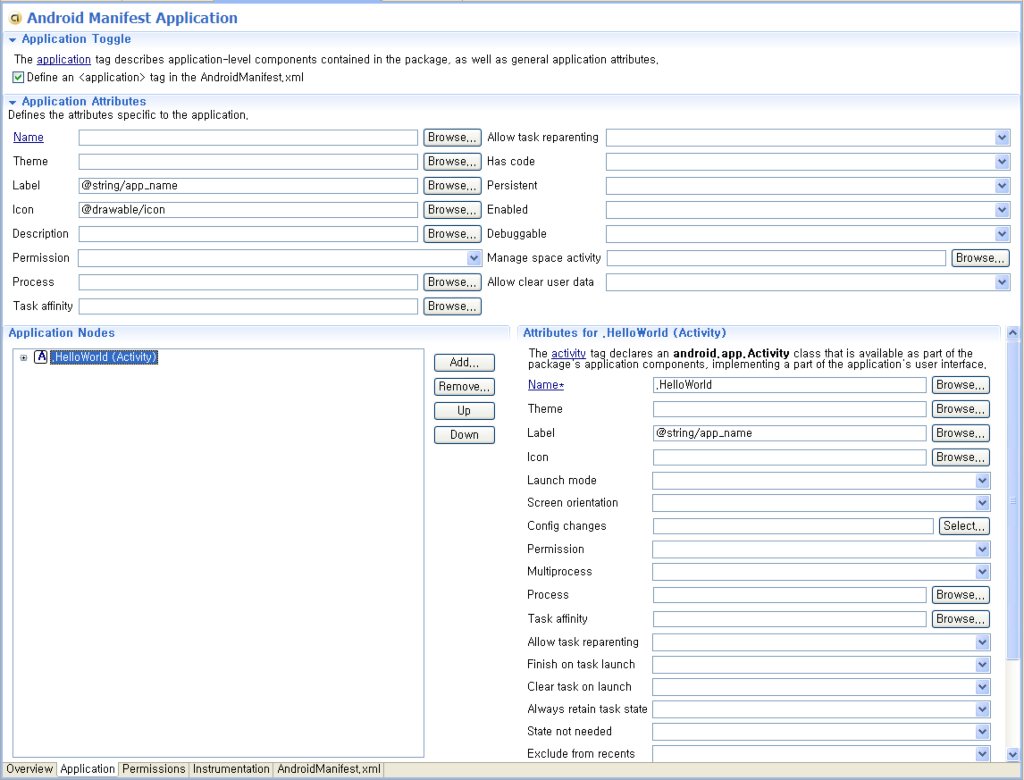
액티비티를 추가해주기 위해서는 아래의 두 가지 작업을 수행해주어야 합니다.

*- 메니페스트 파일(AndroidManifest.xml)에 액티비티 등록하기*

*- 액티비티 소스 코드 만들기*

**1. 메니페스트 파일에 액티비티 등록하기**

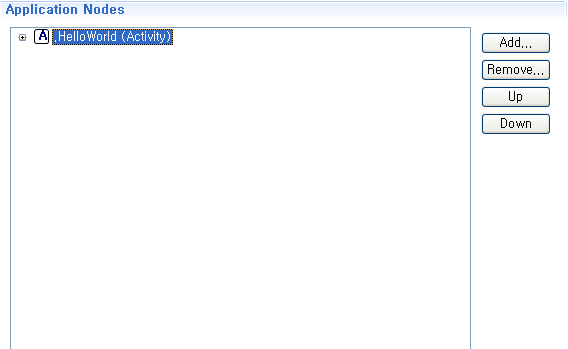
안드로이드에서 액티비티를 호출하기 위해서는 먼저 메니페스트 파일에 액티비티가 등록되어 있어야 합니다. 메니페스트 파일에 액티비티를 등록해주기 위해, 먼저 메니페스트 파일(AndroidManifest.xml) 파일을 연 후, Application 탭을 클릭합니다.

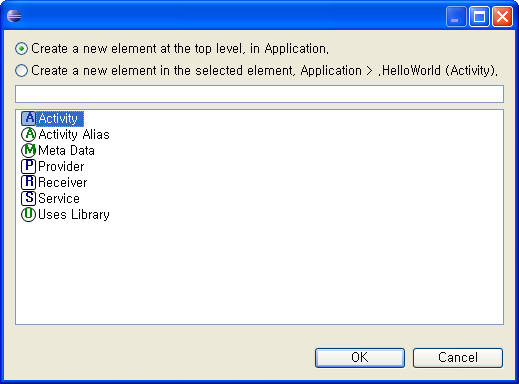
[](http://cfs10.tistory.com/original/9/tistory/2009/02/22/17/11/49a108bc09cdc)

이클립스 ADT로 열어본AndroidManifest.xml

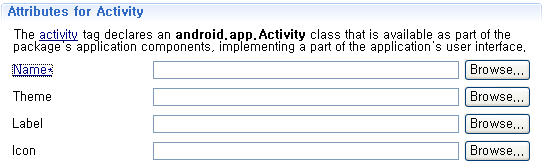
여러가지 속성을 설정할 수 있는 화면들과 함께 어플리케이션의 여러 구성요소들을 등록할 수 있는 Application Nodes 화면이 화면 좌측 하단에 있습니다. 이곳에서 액티비티 뿐만 아니라 다른 액티비티 구성요소 (서비스, 컨텐트 프로바이더 등)도 추가해줄 수 있습니다.

액티비티를 추가하기 위해 Application Nodes 부분의 Add... 버튼을 클릭합니다.

[](http://cfs14.tistory.com/original/1/tistory/2009/02/22/17/10/49a1087e2c040)

[](http://cfs14.tistory.com/original/31/tistory/2009/02/22/17/10/49a1087e4f2fe)

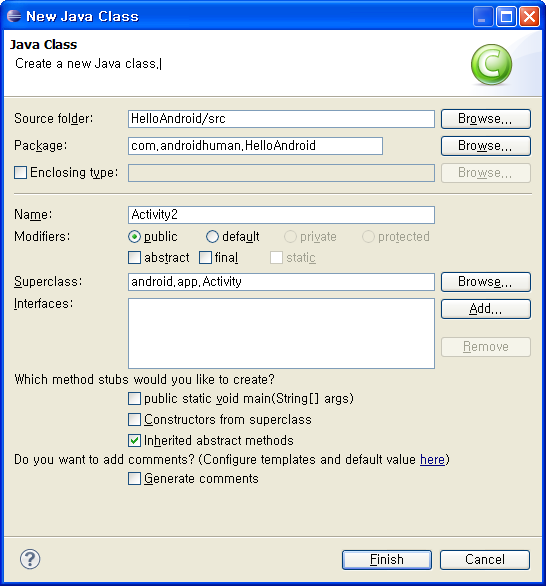
Add.. 화면을 클릭하면 다음과 같은 대화상자가 나타나는데, 이곳에서 "Activity"를 클릭합니다.  
"OK" 버튼을 클릭한 후, 추가된 Activity를 선택한 후 오른쪽의 Attributes 화면을 보면 다음과 같은 화면이 보일 것입니다.

[](http://cfs14.tistory.com/original/35/tistory/2009/02/22/17/10/49a1087e69b8b)

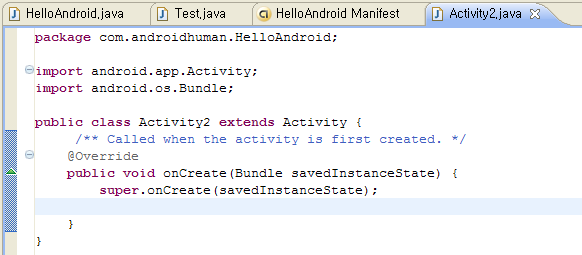
**2. 액티비티 소스 코드 만들기**

메니페스트 파일에 액티비티를 선언하였으니, 이 액티비티의 실질적인 동작을 구현할 자바 소스 코드를 만들어야 합니다.

파란색으로 되어있는 Name\* 을 클릭하여 새 클래스를 만들어줍시다.



위와 같은 대화상자가 뜨게 될 것입니다. 다른 것은 특별히 건드릴 필요가 없고, 클래스의 이름 (일반적으로 액티비티의 이름과 동일하게 설정합니다) 을 입력합니다. 이름을 입력했으면, Finish 버튼을 누릅시다.



Finish 버튼을 누르면, 다음과 같이 새로운 코드 파일이 생성이 됩니다. 처음 프로젝트를 만들 때와 동일하게 onCreate() 메소드까지 추가되어있습니다. :) 단, 액티티의 화면을 표시해주는 메소드인 setContentView()메소드가 추가되어있지 않으므로, 이 상태로 이 액티비티를 호출하면 화면에는 아무것도 표시되지 않게 됩니다.  
  
 여기까지 하셨다면, 호출 대상을 명확히 지정하는 명시적 인텐트(Explicit Intent)를 이용하여 이 액티비티를 호출할 수 있습니다. 하지만, 인텐트 필터를 정의하지 않았기에 암시적 인텐트(Implicit Intent)를 통해 호출될 수는 없습니다.

|  |
| --- |
| Exercises : 인텐트를 이용하여 액티비티 호출하기 |

강좌를 따라오기 전에, 준비해야 할 것들은 다음과 같습니다.

- 새로운 프로젝트  
- 두 개의 소스 코드 및 레이아웃 코드 (액티비티를 두 개 만들어야하니, 당연히 두 개 필요하겠죠??)

이번 강좌에서 예제로 만드는 어플리케이션의 구성은 다음과 같습니다.

\* 첫 번째 액티비티 (IntentExample)에서 버튼을 누르면 두 번째 액티비티가 호출됨  
\* 두 번째 액티비티 (Activity2)에서 종료 버튼을 누르면 두 번째 액티비티가 종료됨

우선, 첫 번째 액티비티의 레이아웃 코드와 소스 코드를 보겠습니다.  
  
[레이아웃]  
01.<Linearlayout xmlns:android="<http://schemas.android.com/apk/res/android>"

02.android:orientation="vertical"

03.android:layout\_width="fill\_parent"

04.android:layout\_height="fill\_parent">

05.<Textview android:layout\_width="fill\_parent"

06.android:layout\_height="wrap\_content"

07.android:text="첫 번째 액티비티 화면입니다."/>

08.<Button android:layout\_width="fill\_parent"

09.android:layout\_height="wrap\_content"

10.android:text="두 번째 액티비티 호출"

11.android:id="@+id/launchActivity"></Button>

12.</Linearlayout>

[소스]

01.package com.androidhuman.IntentExample;

02.import android.app.Activity;

03.import android.content.Intent; // 인텐트 사용 위해 import합니다.

04.import android.os.Bundle;

05.import android.view.View;

06.import android.widget.Button;

07.

08.public class IntentExample extends Activity {

09./\*\* Called when the activity is first created. \*/

10.@Override

11.public void onCreate(Bundle savedInstanceState) {

12.super.onCreate(savedInstanceState);

13.setContentView(R.layout.main);

14.Button launch = (Button)findViewById(R.id.launchActivity);

15.launch.setOnClickListener(new Button.OnClickListener(){

16.public void onClick(View v) {

17.// TODO Auto-generated method stub

18.Intent intent = new Intent(IntentExample.this, Activity2.class); // 두번째 액티비티를 실행하기 위한 인텐트

19.startActivity(intent); // 두번째 액티비티를 실행합니다.

20.}

21.});

22.}

23.}

다른 것들은 대부분 앞에서 다루었던 것들이고, 생소한 것은 단연 인텐트(Intent)일 것입니다. 인텐트에 관한 것을 설명하자면 엄청 길어지지만, 항상 그래왔던 것처럼(!) 하나하나씩 다루어보도록 하겠습니다.   
이번 예제에서 사용한 인텐트의 생성자는 "다른 액티비티를 호출할 때" 사용하는 생성자입니다. 참고로 인텐트를 사용하여 다른 네이티브 액티비티(전화번호 입력기, 주소록, 브라우저등..)도 호출할 수 있습니다. 하지만 일단 그런 것은 나중에 다루도록 하고, 이번 강좌에서는 사용자 액티비티를 호출하는 것에 대해 다루도록 하겠습니다.  
그럼, 생성자의 원형을 분석해보도록 합시다.

**public Intent(Context packageContext, Class<?>cls)  
\* packageContext :**  
한마디로 말하면 호출하는 액티비티를 뜻합니다. 여기에서는 IntentExample이 되겠습니다.(IntentExample.this)  
**\*cls :**호출할 클래스를 뜻합니다. 여기에서는 Activity2 액티비티를 호출해야하므로, Activity2의 클래스인 Activity2.class를 입력합니다.

이렇게 인텐트 설정이 끝났으면, 인텐트를 실행해야겠지요? 사실, 인텐트 자체를 생성함으로써 인텐트가 실행되는 것이 아니라, 인텐트는 하나의 "실행하는 방법"만을 담고 있고, 실제로 그 일을 수행하는 것은 startActivity()메소드가 담당합니다. 인텐트를 실행하는 메소드도 이것 말고도 여러 가지가 있지만, 일단은 요거 하나만 다뤄보도록 하겠습니다.

**public void startActivity(Intent *intent*)**  
  
파라미터로 받은 *intent* 인텐트를 실행합니다.

자, 그럼 첫 번째 액티비티에 대한 코딩은 끝났으니 두 번째 액티비티에 대한 소스와 레이아웃을 설계해보도록 하죠.  
  
[레이아웃]  
01.<Linearlayout xmlns:android="<http://schemas.android.com/apk/res/android>"

02.android:orientation="vertical"

03.android:layout\_width="wrap\_content"

04.android:layout\_height="wrap\_content"

05.android:id="@+id/LinearLayout01">

06.

07.<Textview android:layout\_width="fill\_parent"

08.android:layout\_height="wrap\_content"

09.android:text="두번째 액티비티 화면입니다. 아래 버튼을 누르시면 액티비가 종료됩니다."

10.android:id="@+id/Activity\_2\_text"></Textview>

11.

12.<Button android:layout\_width="fill\_parent"

13.android:layout\_height="wrap\_content"

14.android:text="액티비티 종료하기"

15.android:id="@+id/terminateActivity"></Button>

16.</Linearlayout>

[소스]

01.package com.androidhuman.IntentExample;

02.import android.app.Activity;

03.import android.os.Bundle;

04.import android.view.View;

05.import android.widget.Button;

06.public class  Activity2 extends Activity {

07./\*\* Called when the activity is first created. \*/

08.@Override

09.public void onCreate(Bundle savedInstanceState) {

10.super.onCreate(savedInstanceState);

11.setContentView(R.layout.layout2);

12.Button terminate = (Button)findViewById(R.id.terminateActivity);

13.terminate.setOnClickListener(new Button.OnClickListener(){

14.public void onClick(View v) {

15.// TODO Auto-generated method stub

16.finish(); // 액티비티를 종료합니다.

17.}

18.

19.});

20.

21.}

22.}

두 번째 액티비티는 사실 더 별 거 없습니다. 버튼에 리스너 다는 건 이제 아무것도 아니고, 인텐트도 없구요. 대신에, 액티비티를 종료하는 메소드인 finish()메소드를 볼 수 있습니다. 사용자가 액티비티를 종료하고 싶을 땐, 이 finish()메소드를 사용하면 됩니다.

**public void finish()**  
액티비티를 종료합니다.

여기까지 잘 따라오셨다면, 다음과 같은 화면을 보실 수 있을 것입니다.   
첫 번째 액티비티에서 버튼을 누르면 두 번째 액티비티가 실행되고, 두 번째 액티비티의 버튼을 누르면 두 번째 액티비티는 종료됩니다.

|  |  |
| --- | --- |
| [http://cfile24.uf.tistory.com/image/115CF20949AAEBE715FE37](http://cfile24.uf.tistory.com/original/115CF20949AAEBE715FE37) | [http://cfile2.uf.tistory.com/image/115C910949AAEBE70022DD](http://cfile2.uf.tistory.com/original/115C910949AAEBE70022DD) |