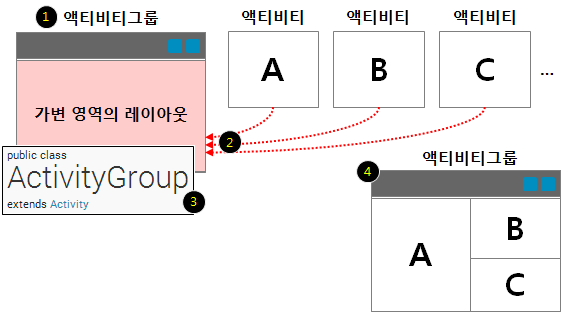
**22.2 액티비티그룹**

액티비티그룹은 액티비티면서 내부적으로 또다른 액티비티를 포함할 수 있다. 이는 뷰그룹을 떠올리면 이해하기 쉬운데, 뷰그룹이 하위에 자식 뷰들을 포함하고 배치할 수 있는 것과 같다. 다음 그림을 통해 액티비티그룹에 대해서 좀더 살펴보자.

그림 22-7 액티비티그룹



①     액티비티그룹은 자신이 가진 레이아웃 중 가변적으로 변하는 영역에

②     특정 액티비티를 포함할 수 있다. 포함되는 액티비티는 일종의 자식 액티비티라 볼 수 있다.

③     액티비티그룹 역시 액티비티다. 따라서 일반 액티비티의 역할을 모두 가질 뿐만 아니라 다른 액티비티를 포함할 수 있도록 확장되었다. 또한 액티비티그룹은 자식 액티비티로 액티비티그룹을 가질 수도 있다. 이는 액티비티그룹이 액티비티를 상속받았기 때문에 가능한 것이다. 뷰그룹이 자식뷰로 뷰그룹을 가질 수 있는 것과 같다.

④     액티비티그룹은 자신이 가진 레이아웃의 어떤부분이든 자식 액티비티를 배치할 수 있다.

이렇게 액티비티그룹을 이용하면 하나의 액티비티 내에서 다수의 레이아웃을 전환해야 하는 경우 편리하다. 즉 전환할 레이아웃을 독립적인 액티비티 단위로 처리하기 때문에 재사용성을 높이고 소스의 복잡도를 크게 낮춘다.

**22.2.1 액티비티그룹을 사용해보자.**

설명에 앞서 액티비티그룹은 안드로이드 API 13부터 디플리케이티드 되었다. 그 말은 안드로이드에서 더 이상 액티비티그룹을 지원하지 않고 사용을 권장하지 않는다는 의미다. 왜 그렇게 되었는지는 액티비티그룹을 직접 활용해본 뒤 설명토록 한다. 그리고 디플리케이티드 된 기술이므로 상세한 내용은 생략하고, 전반적인 구조를 파악할 수 있는 것에만 초점을 둔다.

사실 액티비티그룹은 많은 개발자에게 좀 생소할 수 있다. 그러나 액티비티그룹에서 파생된 탭액티비티TabActivity를 언급하면 모르는 개발자들이 없을 것이다. 그만큼 액티비티그룹은 탭 형태의 화면 구조인 탭액티비티를 통해 제한적으로 사용되었다는 의미다. [ REF \_Ref396239822 \r \h 그림 22-8]을 통해 탭액티비티가 활용된 앱을 살펴보자.

그림 22-8 탭앱티비티



①     안드로이드 기본 앱인 환경 설정 앱이다. 화면 상단에는 탭 버튼이 수평으로 나열되어 있고 해당버튼을 누르면 아래에 그 버튼과 관련된 화면이 나타난다.

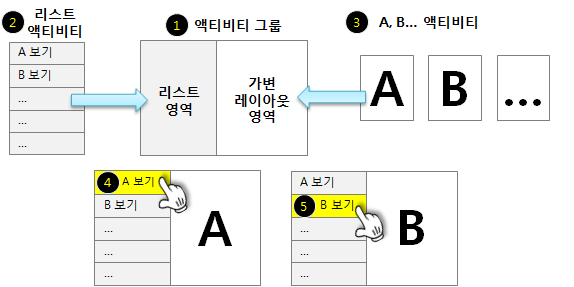
②     또 다른 기본 앱을 살펴보면 전화 앱이 있다. 역시 상단에 수평으로 나열된 버튼이 있고 해당버튼을 누르면 아래에 그 버튼과 관련된 화면이 나타난다.

③     환경 설정과 전화 같은 화면 구조의 앱은 액티비티그룹을 통해 구현되었고, 화면 상단에 탭 버튼영역과 화면 하단에 탭 버튼을 눌렀을 때 보여지는 가변 영역으로 구성된다. 그리고 이러한 화면 구조의 액티비티그룹은 가장 많이 활용되기 때문에 안드로이드는 개발자들이 쉽게 구현할 수 있도록 파생된 탭액티비티를 제공한다.

④     탭액티비티 클래스를 보면 액티비티그룹을 상속받고 있는 것을 확인할 수 있다.

탭액티비티를 통해 액티비티그룹이 뭔지 약간의 감은 잡았을 거라 생각한다. 탭액티비티 역시 액티비티그룹과 함께 안드로이드 API 13부터 디플리케이티드 되었다. 그러므로 탭액티비티를 배우는데 아까운 시간을 낭비하지 않고, 바로 액티비티그룹 예제를 맛보고 넘어가자. 다음은 액티비티그룹에 사용될 레이아웃 구조다.

그림 22-9 액티비티그룹 예제 레이아웃 구조



①     액티비티그룹의 레이아웃은 좌측 리스트 영역과 우측 가변적으로 변할 레이아웃 영역으로 구성한다. 즉 수평배치 LinearLayout으로 두 가지 영역의 자식뷰를 배치할 수 있는 형태다.

②     액티비티그룹 레이아웃의 좌측 영역에는 리스트뷰가 포함된 레이아웃의 액티비티를 배치한다.

③     액티비티그룹 레이아웃의 우측 영역에는 여러 액티비티들이 가변적으로 배치된다.

④     완성된 액티비티그룹의 레이아웃에서 좌측 액티비티의 A라는 리스트 아이템을 선택하면 A 액티비티를 우측에 보여주고

⑤     B라는 리스트 아이템을 선택하면 B 액티비티로 변경되어 보여준다. 즉 탭액티비티에서 탭 버튼들이 좌측 리스트 형식으로 변경되었다고 보면 된다.

이제 새로운 프로젝트를 생성하고 다음과 같이 AndroidManifest.xml을 구성한다.

예제 22-9 액티비티그룹 예제 앱의 AndroidManifest.xml

|  |
| --- |
| **AndroidManifest.xml**    <?xml version=*"1.0"* encoding=*"utf-8"*?>  <manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*      package=*"com.superdroid.fragmentactivitygroup"*      android:versionCode=*"1"*      android:versionName=*"1.0"* >        <uses-sdk          android:minSdkVersion=*"16"*          android:targetSdkVersion=*"16"* />        <application          android:allowBackup=*"true"*          android:icon=*"@drawable/ic\_launcher"*          android:label=*"@string/app\_name"*          android:theme=*"@style/AppTheme"* >            <!-- 액티비티그룹에 사용될 액티비티 -->  **<activity android:name=*".MainActivityGroup"***              android:label=*"@string/app\_name"* >              <intent-filter>                  <action android:name=*"android.intent.action.MAIN"* />                    <category android:name=*"android.intent.category.LAUNCHER"* />              </intent-filter>          </activity>            <!-- 화면 좌측에 표시될 리스트 액티비티 -->  **<activity android:name=*".ListMenuActivity"*/>**            <!-- 화면 우측에 텍스트를 보여주는 A 액티비티 -->  **<activity android:name=*".TextViewerActivity"*/>**          <!-- 화면 우측에 이미지를 보여주는 B 액티비티 -->  **<activity android:name=*".ImageViewerActivity"*/>**        </application>    </manifest> |

[예제 22-9]을 살펴보면 총 네 가지 액티비티를 등록하였다. 먼저 첫 번째는 자식 액티비티를 포함하게 될 액티비티그룹이고, 두 번째는 좌측 리스트 메뉴 액티비티가 된다. 그리고 세 번째와 네 번째는 리스트 아이템의 선택에 따라 우측에 가변적으로 보여주게 될 액티비티다.

다음은 네 가지 액티비티에서 사용될 레이아웃을 구현해보자. 먼저 [ REF \_Ref396329446 \r \h 그림 22-9] ①의 액티비티그룹에서 사용될 레이아웃을 구현한다.

예제 22-10 액티비티그룹에 사용될 레이아웃

|  |
| --- |
| **res/layout/activity\_group\_layout.xml**    <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*      android:orientation=*"horizontal"*>        <!-- 좌측 리스트 영역 -->  **<FrameLayout android:id=*"@+id/list\_menu\_layout"***          android:layout\_width=*"0dp"*          android:layout\_height=*"match\_parent"*          android:layout\_weight=*"1"*          android:background=*"#EEE"* />        <!-- 우측 가변 레이아웃 영역 -->  **<FrameLayout android:id=*"@+id/viewer\_layout"***          android:layout\_width=*"0dp"*          android:layout\_height=*"match\_parent"*          android:layout\_weight=*"1"*/>    </LinearLayout> |

[예제 22-10]을 살펴보면 수평 배치 LinearLayout에 두 가지 FrameLayout 자식 뷰그룹을 배치하고 있다. 이 각각의 영역에는 자식 액티비티가 배치될 것이다.

다음은 [ REF \_Ref396329446 \r \h 그림 22-9] ②의 액티비티그룹 좌측 영역에 들어갈 리스트 메뉴 액티비티 레이아웃을 구현한다.

예제 22-11 화면 좌측에 리스트 메뉴 액티비티 레이아웃

|  |
| --- |
| **res/layout/activity\_list\_menu\_layout.xml**    <ListView  xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:id=*"@+id/listview"*      android:background=*"#EEE"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*  **android:entries=*"@array/IDSA\_LIST\_ITEM"***/> |

[예제 22-10]을 살펴보면 간단히 리스트뷰 하나만 추가되었다. 그리고 리스트뷰에 entries 속성을 사용하고 있는데, 이 속성값으로 문자열 배열 리소스를 설정하면 자동으로 문자열 배열 내용이 리스트의 각 아이템들로 채워진다. 따라서 자바 소스를 통해 어댑터를 구현하지 않아도 리스트뷰 내부적으로 ArrayAdapter를 생성하고 설정되어 편리하다. 그렇다면 다음은 entries 속성 값으로 설정된 문자열 배열 리소스를 구현해보자.

예제 22-12 리스트뷰의 아이템으로 표시될 문자열 배열 리소스

|  |
| --- |
| **res/values/string-arrays.xml**    <?xml version=*"1.0"* encoding=*"utf-8"*?>  <resources>    **<string-array name=*"IDSA\_LIST\_ITEM"*>**          <item>텍스트 보기</item>          <item>이미지 보기</item>          <item>계산기</item>      </string-array>    </resources> |

[예제 22-12]을 살펴보면 문자열 배열 리소스에는 세 가지 항목을 추가했다. 각 항목은 액티비티 레이아웃 좌측에 보여질 액티비티명이다. 차후 리스트의 항목을 선택하면 텍스트를 보여주는 액티비티, 이미지를 보여주는 액티비티, 계산기를 보여주는 액티비티가 액티비티그룹 우측 영역에서 실행되게 할 것이다. 그렇다면 리스트의 첫 번째 아이템에 해당하는 텍스트를 보여줄 액티비티 레이아웃을 구현해보자. 이는 [ REF \_Ref396329446 \r \h 그림 22-9] ③ 영역에 들어갈 액티비티 중 하나다.

예제 22-13 화면 우측에 텍스트를 보여줄 A 액티비티 레이아웃

|  |
| --- |
| **res/layout/activity\_text\_viewer\_layout.xml**    <TextView xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:id=*"@+id/text\_viewer"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*  **android:text=*"텍스트를 출력하는 부분입니다."***/> |

[예제 22-13]을 살펴보면 간단히 문자열을 출력하는 텍스트뷰 하나만 추가되었다. 다음은 리스트의 두 번째 아이템에 해당하는 텍스트를 보여줄 액티비티 레이아웃을 구현해보자.

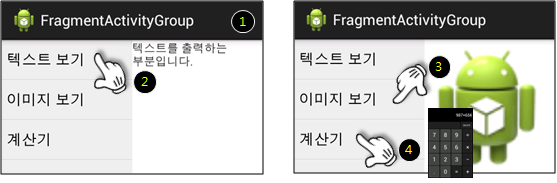
예제 22-14 화면 우측에 이미지를 보여줄 B 액티비티 레이아웃

|  |
| --- |
| **res/layout/activity\_image\_viewer\_layout.xml**    <ImageView xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:id=*"@+id/image\_viewer"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*  **android:src=*"@drawable/ic\_launcher"***/> |

[예제 22-14] 역시 간단히 이미지를 보여주는 이미지뷰 하나만 추가되었다. 이미지뷰에 출력될 리소스는 패키지 내부에 기본으로 존재하는 앱 아이콘 이미지를 사용한다.

여기까지 구현된 레이아웃이 어떻게 보여지고, 어떻게 동작하게 될지 잠시 살펴보자.

그림 22-10 각 액티비티가 액티비티그룹에 배치된 모습



①     액티비티그룹이 실행된 화면이다. 좌측 영역에는 리스트 메뉴 액티비티가 구동되고, 우측에는 리스트 아이템에 해당하는 액티비티가 구동된다.

②     좌측 액티비티에서 텍스트 보기 리스트 아이템을 클릭하면, 우측에는 텍스트를 보여주는 액티비티가 실행된다.

③     그리고 이미지 보기 리스트 아이템을 클릭하면, 우측에는 이미지를 보여주는 액티비티가 실행된다.

④     그리고 계산기 아이템을 클릭하면, 우측에는 안드로이드 기본 앱 중 계산기 앱의 액티비티가 실행된다. 액티비티그룹은 내부적으로 독립적인 액티비티 단위를 추가할 수 있기 때문에 외부 앱의 액티비티도 실행해본 것이다.

지금까지 액티비티그룹과 각각의 자식 액티비티들의 레이아웃을 구성했다. 다음은 액티비티 자체를 구현한다.

구현했던 레이아웃들의 역순으로 액티비티 소스를 구현하자. 먼저 텍스트를 보여주는 액티비티다.

예제 22-15 화면 우측에 텍스트를 보여줄 A 액티비티

|  |
| --- |
| **src/TextViewerActivity.java**    **public** **class** TextViewerActivity **extends** Activity {        @Override  **protected** **void** onCreate(Bundle savedInstanceState)      {  **super**.onCreate(savedInstanceState);  **setContentView(R.layout.*activity\_text\_viewer\_layout*);**      }  } |

[예제 22-15] 예제를 최대한 단순화시켜 이해도를 높이기 위해 불필요한 소스는 추가하지 않았다. 간단히 액티비티 콘텐트 영역에 텍스트를 보여주는 레이아웃을 설정한 것 이외는 없다.

다음은 이미지를 보여주는 액티비티를 구현한다.

예제 22-16 화면 우측에 이미지를 보여줄 B 액티비티

|  |
| --- |
| **src/ImageViewerActivity.java**    **public** **class** ImageViewerActivity **extends** Activity {        @Override  **protected** **void** onCreate(Bundle savedInstanceState)      {  **super**.onCreate(savedInstanceState);  **setContentView(R.layout.*activity\_image\_viewer\_layout*);**      }  } |

[예제 22-16] 예제 역시도 간단히 이미지를 보여주는 레이아웃 설정 외에는 없다.

다음은 리스트 메뉴 액티비티를 구현한다.

예제 22-17 화면 좌측에 리스트 메뉴 액티비티

|  |
| --- |
| **src/ListMenuActivity.java**    **public** **class** ListMenuActivity **extends** Activity **implements** OnItemClickListener {        ListView                    mListView = **null**;      OnListItemClickListener     mListItemClickListener = **null**;        @Override  **protected** **void** onCreate(Bundle savedInstanceState)      {  **super**.onCreate(savedInstanceState);          setContentView(R.layout.*activity\_list\_menu\_layout*);            mListView = (ListView)findViewById( R.id.*listview* );            // ① 리스트 클릭 리스너를 설정한다.  **mListView.setOnItemClickListener( this );**      }        // ② 해당 액티비티를 사용하는 액티비티그룹에서 리스트 아이템 클릭시 그 이벤트를      //    수신받기 위한 리스너 인터페이스를 정의한다.      //    이 리스너를 통해 해당 액티비티는 독립적으로 동작될 수 있다.  **public interface OnListItemClickListener**  **{**  **void onItemClick( Intent viewerIntent, String viewerId );**  **}**        // ③ 해당 액티비티에서 발생되는 리스트 아이템 클릭 이벤트 수신 받기 위한      //    리스너를 등록함수다.  **public void setOnListItemClickListener(**  **OnListItemClickListener ln )**      {          mListItemClickListener = ln;      }          @Override  **public** **void** onItemClick( AdapterView<?> parent,                                  View view,  **int** position,  **long** id )      {          // ④ 클릭한 아이템에 해당하는 액티비티를 실행할 수 있도록 인텐트를 설정한다.          // ====================================================================          Intent startViewerActivityIntent = **new** Intent();          String viewerActivityId = **null**;    **switch** ( position )          {  **case** 0:                  startViewerActivityIntent.setClass( **this**,                                                      TextViewerActivity.**class** );                  viewerActivityId = "TextViewerActivity";  **break**;    **case** 1:                  startViewerActivityIntent.setClass( **this**,                                                      ImageViewerActivity.**class** );                  viewerActivityId = "ImageViewerActivity";  **break**;    **case** 2:                  startViewerActivityIntent.setAction( Intent.*ACTION\_MAIN* );                  startViewerActivityIntent.addCategory(                                                Intent.*CATEGORY\_APP\_CALCULATOR* );                  viewerActivityId = "Calculator";  **break**;          }          // ====================================================================            // ⑤ 등록된 아이템 클릭 리스너의 핸들러 함수를 실행한다.          // ====================================================================  **if**( mListItemClickListener != **null** )          {  **mListItemClickListener.onItemClick(**  **startViewerActivityIntent, viewerActivityId );**          }          // ====================================================================      }  } |

①     리스트 메뉴의 역할은 리스트의 특정 아이템을 선택했을 때 액티비티그룹 영역 좌측에 선택된 아이템에 해당하는 액티비티를 실행하는 것이다. 따라서 아이템을 클릭했을 때 이벤트를 수신 받아 처리하는 것이 핵심이겠다. 이를 위해 리스트뷰에 아이템 클릭 리스너를 설정한다. OnItemClickListener의 구현은 해당 클래스를 상속받은 액티비티 내에서 구현한다.

②     그런데 리스트 메뉴 액티비티에서 선택된 아이템의 액티비티를 실행하는 것은 좋은 방법이 아니다. 그 이유는 액티비티의 독립성 때문인데, 액티비티 실행시킬 곳이 액티비티그룹의 우측 영역이라 자식 액티비티가 부모 액티비티그룹을 제어하는 것은 독립성에 위배된다. 즉 부모 액티비티그룹과 자식 액티비티가 서로 연관관계가 생겨 분리될 수 없어져 버린다. 그렇다면 부모 액티비티그룹이 우측 영역에 액티비티를 실행해야 한다. 그런데 액티비티를 실행하는 시점을 액티비티그룹은 알 수 없다. 그 시점은 리스트의 아이템이 선택되었을 땐데, 메뉴 리스트 액티비티만 OnItemClickListener를 통해 받을 수 있기 때문이다. 이를 위해 리스트 메뉴 액티비티는 내부적으로 OnListItemClickListener라는 아이템 선택 리스너 인터페이스를 정의한다. 그리고 액티비티그룹이든 그 누구든 해당 인터페이스를 구현한 객체를 리스트 메뉴 액티비티에 전달하면, 리스트 메뉴 액티비티는 리스트 아이템이 선택되었을 때 OnListItemClickListener의 핸들러 함수 onItemClick을 실행해준다. 이로써 리스트 메뉴 액티비티는 독립적인 클래스가 되었다. 이는 뷰가 터치 혹은 키보드 이벤트 리스너를 제공하여 독립성을 높이는 이유와 같은 원리다. 리스너 인터페이스는 클래스의 독립성을 높이는데 가장 대표적인 방법이다. 늘 활용되는 방법이니 꼭 기억해두자.

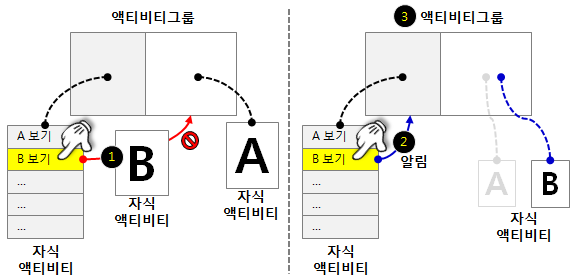
③     setOnListItemClickListener 함수는 리스너 객체를 설정하는 함수다. 해당 함수를 통해 액티비티그룹 등에서 리스너 객체를 등록할 수 있다.

④     리스트뷰의 아이템이 클릭되면 onItemClick 핸들러 함수가 호출되는데, 이때 선택된 아이템의 액티비티를 실행할 수 있는 인텐트를 준비해둔다.

⑤     마지막으로 등록된 OnListItemClickListener 객체가 있다면 해당 객체의 onItemClick 핸들러 함수를 호출하고, 인자로 인텐트와 액티비티를 구분할 id를 전달한다. 인자값 중 액티비티를 구분하는 id는 액티비티그룹에서 설명한다.

[예제 22-17]의 소스 구조를 다음의 그림으로 정리하고 넘어가자.

그림 22-11 자식 액티비티 관리는 액티비티그룹의 역할



①     리스트 아이템을 선택하면 아이템에 해당하는 액티비티를 액티비티그룹 우측에 실행한다. 만일 우측에 A 액티비티가 존재한다면 B 액티비티로 대체될 것이다. 하지만 자식 액티비티가 액티비티그룹의 우측 영역을 제어하는 것은 올바른 구조가 아니다. 각 액티비티는 독립적으로 동작해야 어떤 액티비티그룹에도 유연하게 끼워넣을 수 있기 때문이다.

②     이를 위해 리스트 메뉴 액티비티는 아이템을 선택했을 때 선택된 아이템 정보를 상위 액티비티 그룹에게 알려주고

③     액티비티그룹은 그 정보를 받아 좌측 영역의 액티비티를 실행한다. 즉 액티비티그룹은 자식 액티비티들을 관리 및 제어하고, 자식 액티비티들은 자신만의 고유한 기능을 독립적으로 수행하는 구조가 되어야 한다. 이렇게 자식 액티비티의 독립성을 위해 리스트 메뉴 액티비티는 아이템 선택 리스너 인터페이스를 제공하고, 액티비티그룹은 자식 리스트 메뉴 액티비티에 리스너를 등록하여 직접 자식 액티비티들을 제어한다. 액티비티그룹에서 리스너를 등록하는 과정은 [예제 22-18]에서 살펴본다.

마지막으로 액티비티그룹이 자식 액티비티를 관리하는 것을 다음 소스로 살펴보자.

예제 22-18 액티비티그룹

|  |
| --- |
| **src/MainActivityGroup.java**    // ① 액티비티그룹을 상속받음으로써 자식 액티비티를 관리할 수 있게 된다.  **public class MainActivityGroup extends ~~ActivityGroup~~ {**        FrameLayout mListMenuLayout     = **null**;      FrameLayout mViewerLayout   = **null**;        // ② 액티비티그룹의 자식 액티비티를 관리하는 로컬 액티비티매니저를 참조한다.  **~~LocalActivityManager~~** mLocalActivityMgr = getLocalActivityManager();        @Override  // ③ 액티비티그룹도 액티비티를 상속받았기 때문에 액티비티 생명주기 함수가  //    모두 존재한다. 따라서 액티비티그룹도 onCreate 생명주기 함수가 최초 호출된다.  **protected** **void** onCreate(Bundle savedInstanceState)      {  **super**.onCreate(savedInstanceState);          setContentView(R.layout.*activity\_group\_layout*);            // ④ 액티비티가 들어갈 뷰 참조          // ====================================================================          // 리스트 메뉴 영역의 레이아웃 참조          mListMenuLayout = (FrameLayout)findViewById( R.id.*list\_menu\_layout* );          // 뷰어 영역의 레이아웃 참조          mViewerLayout = (FrameLayout)findViewById( R.id.*viewer\_layout* );          // ====================================================================            // ⑤ 화면 좌측에 리스트 메뉴 액티비티 실행          // ====================================================================          // 메뉴 리스트 액티비티를 실행한다.          Intent menuActivityIntent = **new** Intent( **this**, ListMenuActivity.**class** );          Window menuWindow =  **mLocalActivityMgr.startActivity( "ListMenuActivity",**  **menuActivityIntent );**            // 실행된 메뉴 액티비티의 레이아웃을 참조하여 액티비티그룹 좌측영역에 추가한다.          View   menuView   = menuWindow.getDecorView();          mListMenuLayout.addView( menuView );          // ====================================================================            // ⑥ 리스트 메뉴 아이템이 클릭되었을 때 호출되는 리스너를 등록한다.          // ====================================================================  **ListMenuActivity listActivity =**  **(ListMenuActivity)mLocalActivityMgr**                                                  **.getActivity( "ListMenuActivity" );**  **listActivity.setOnListItemClickListener(**  **new ListMenuActivity.OnListItemClickListener()**          {              @Override  **public** **void** onItemClick( Intent viewerIntent, String viewerId )              {                  // 클릭된 아이템의 해당하는 액티비티를 넘겨받은 인텐트로 실행한다.                  Window viewerWindow = **mLocalActivityMgr.startActivity(**  **viewerId, viewerIntent );**                    // 실행된 액티비티 레이아웃을 참조하여 액티비티그룹 우측영역에 추가한다.                  View    viewerView  = viewerWindow.getDecorView();                  mViewerLayout.removeAllViews();                  mViewerLayout.addView( viewerView );              }          });          // ====================================================================      }  } |

①     액티비티그룹을 상속 받음으로써 자식 액티비티를 관리할 수 있게 된다. 그런데 이클립스에서 ~~ActivityGroup~~에 취소선을 표시하고 있다. 그 이유는 액티비티그룹 자체가 디플리케이티드 되었기 때문이다.

②     액티비티그룹에는 자식 액티비티를 관리하기 위해 로컬액티비티매니저LocalActivityManager가 존재한다. 관리란 자식 액티비티를 추가하거나, 제거, 추가된 액티비티를 참조 등의 용도로 사용된다. 로컬액티비티매니저 객체는 getLocalActivityManager 함수를 통해 참조할 수 있다. 참고로 액티비티그룹이 디플리케이티드 되었기 때문에 로컬액티비티매니저도 함께 디플리케이티드 되었다. 따라서 ~~LocalActivityManager~~에 취소선이 그어졌다.

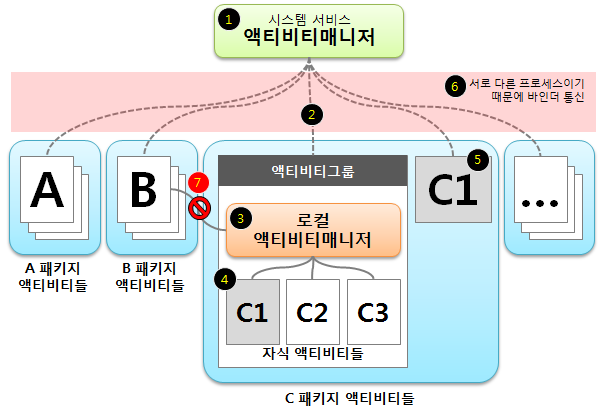
③     액티비티그룹도 액티비티를 상속받았기 때문에 액티비티다. 따라서 액티비티그룹 역시 생명주기의 최초 함수인 onCreate가 호출된다.

④     액티비티그룹 레이아웃에서 자식 액티비티가 실행될 좌/우측 영역의 뷰그룹을 참조해둔다. 이후 이 뷰그룹을 이용하여 자식 액티비티를 끼워넣게 된다.

⑤     다음은 액티비티그룹 좌측 영역에 리스트 메뉴 액티비티를 실행한다. 액티비티를 실행하는 방법은 일반적인 인텐트를 통한 방법과 크게 다르지 않다. 먼저 인텐트 하나를 생성하고 실행될 리스트 메뉴 액티비티 정보를 설정한다. 그 다음 인텐트를 startActivity 함수로 실행한다. 여기서 일반적인 방법과 차이가 있다면 로컬액티비티매니저의 startActivity 함수를 이용한다는 점이다. 액티비티그룹은 자식 액티비티를 로컬액티비티매니저를 통해 관리하기 때문이다. 이 부분은 [ REF \_Ref396505345 \r \h 그림 22-12]에서 좀더 설명하겠다. 그리고 startActivity에 첫 번째 인자로 문자열을 전달하는데, 이 값은 로컬액티비티매니저가 관리하는 액티비티를 찾을 때 id로 이용된다. startActivity를 통해 액티비티를 실행하면 윈도우Window를 반환하는데, 이 윈도우에는 실행된 액티비티의 레이아웃이 포함되어 있다. 따라서 액티비티의 레이아웃을 얻어와 액티비티그룹 레이아웃 좌측 영역에 추가하면 된다. 이 부분은 [ REF \_Ref396505427 \r \h 그림 22-13]에서 좀더 설명하겠다. 참고로 윈도우의 getDecorView 함수를 호출하면 윈도우에 포함된 가장 상위 뷰를 얻어올 수 있다. 그리고 윈도우가 뭔지 정확히 알고 싶다면 책의 21장을 참조하기 바란다. 여기까지 액티비티그룹 좌측 영역에 리스트 메뉴 액티비티가 추가되었다.

⑥     이제 마지막으로 액티비티그룹 우측 영역에 자식 액티비티를 실행하면 된다. 하지만 우측 영역의 경우, 좌측 영역의 리스트 메뉴 액티비티에서 특정 아이템을 선택했을 때 실행된다. 따라서 액티비티그룹은 리스트 메뉴 액티비티에서 아이템 선택했을 때 이벤트를 수신받아 처리해야 한다. 이를 위해 바로 [예제 22-17]의 리스트 메뉴 액티비티에서 리스너를 제공했던 것이다. 그렇다면 리스너를 등록해보자. 먼저 리스너 등록을 위해 리스트 메뉴 액티비티 객체가 필요하다. 이를 위해 로컬액티비티매니저를 통해 앞서 실행된 해당 액티비티를 참조한다. 액티비티 참조는 로컬액티비티매니저의 getActivity 함수를 이용한다. 그 다음 리스트 메뉴 액티비티 객체의 setOnListItemClickListener 함수를 통해 리스트 아이템 클릭 리스너를 등록한다. 등록한 리스너의 구현 내용을 살펴보면 액티비티그룹 우측 영역에 클릭된 아이템의 액티비티를 실행하고 있다. 참고로 실행할 액티비티는 리스너의 onItemClick 핸들러 함수의 인자로 넘어오는 인텐트와 액티비티 id를 이용한다.

그림 22-12 액티비티매니저와 로컬액티비티매니저



①     안드로이드에서 어떤 앱을 불문하고 실행된 모든 액티비티는 시스템 서비스인 액티비티매니저가 관리한다. 여기서 관리란 모든 액티비티의 생명주기와 모든 액티비티의 태스크등을 말한다. 참고로 태스크에 대해서는 책의 12장에서 설명되었다(저자가 집필한 "이것이 안드로이드다"를 이하 ‘책’이라고 함)

②     사실 액티비티그룹 역시 액티비티매니저 입장에서는 일반적인 액티비티일 뿐이다. 따라서 액티비티그룹도 다른 액티비티와 같이 동등하게 관리된다.

③     액티비티그룹에는 로컬액티비티매니저가 존재하는데, 액티비티그룹의 자식 액티비티들을 관리한다.

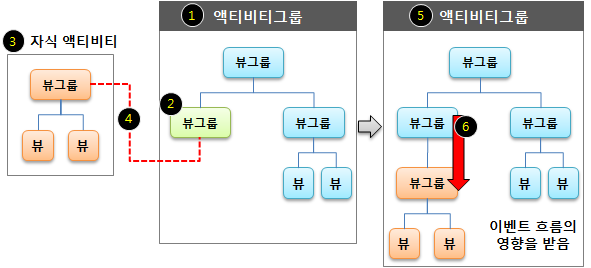
④     따라서 C1, C2, C3 자식 액티비티들은 시스템의 액티비티매니저가 전혀 관여하지 않고 독립적으로 로컬액티비티가 관리하게 된다.

⑤     하지만 액티비티그룹의 자식 액티비티가 아닌, 일반 액티비티로 실행된 C1 액티비티는 시스템의 액티비티매니저를 통해 관리된다.

⑥     여기서 중요한 점 한가지가 있다. 액티비티매니저는 시스템 프로세스에서 동작하고 액티비티매니저가 관리하는 모든 액티비티들은 각각의 앱 프로세스에서 동작한다. 따라서 액비티매니저가 관리하는 앱의 액티비티를 제어하기 위해서는 프로세스 통신(IPC/RPC)이 필요하고, 안드로이드에서 프로세스 통신은 바인더를 이용한다. 참고로 바인더에 대해서는 책의 18장에서 상세히 설명되었다. 그리고 액티비티그룹의 로컬액티비티매니저는 자신의 패키지 내에 액티비티를 관리하기 때문에 바인더 통신이 필요 없이 직접 접근하여 관리한다.

⑦     그 말은 로컬액티비티매니저의 경우 다른 프로세스의 액티비티를 실행하고 관리할 수 없다는 의미다. 즉 액티비티그룹은 다른 앱의 액티비티를 실행하고 포함할 수 없다. 정말 포함될 수 없는지는 예제 실행 결과인 [ REF \_Ref396509238 \r \h 그림 22-14] ④, ⑤과정에서 확인할 수 있다.

그림 22-13 자식 액티비티의 레이아웃을 가지는 액티비티그룹



①     액티비티그룹도 일반 액티비티와 같이 레이아웃을 가진다.

②     또한 특정 뷰그룹 영역에 자식 액티비티를 실행할 수 있다. 하지만 엄밀히 보면 뷰그룹 영역에 액티비티 컴포넌트를 추가하는게 아니다.

③     실행된 자식 액티비티에는 자신의 레이아웃을 가지고,

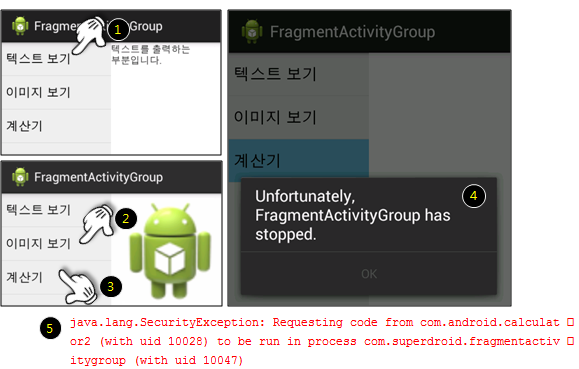
④     자식 액티비티의 레이아웃은 부모 액티비티그룹의 특정 뷰그룹 내에 추가되는 것이다. 참고로 이 것이 바로 [예제 22-18] ⑤의 과정이기도 하다.

⑤     따라서 액티비티그룹은 결국 자식 액티비티의 레이아웃들을 자신의 레이아웃 영역에 끼워 넣은 형태다. 어떻게 보면 액티비티그룹은 여러 개의 자식 액티비티들을 합친 하나의 액티비티와 같다. 시스템의 액티비티매니저 측면에서도 액티비티그룹은 하나의 액티비티일 뿐이다.

⑥     이렇게 합쳐진 자식 액티비티의 레이아웃은 상위 액티비티그룹의 뷰그룹에 영향을 받는다. 예를 들어 상위 뷰그룹에서 터치이벤트를 가로채는 onInterceptTouchEvent 함수를 사용했다면 하위 자식 액티비티의 레이아웃으로 터치이벤트가 전달되지 않는 것이다. 참고로 터치이벤트에 대해서는 책의 8장에서 상세히 설명되었다.

이제 모든 구현은 끝났다. 실행하고 그 결과를 살펴보자.

그림 22-14 액티비티매니저와 로컬액티비티매니저



①     리스트 메뉴 액티비티에서 영역에서 ‘텍스트 보기’ 아이템을 클릭하면 우측 영역에 텍스트 뷰어 액티비티가 실행된다.

②     이어 ‘이미지 보기’ 아이템을 클릭하면 우측 영역에 이미지 뷰어 액티비티로 바뀐다.

③     이어 ‘계산기’ 아이템을 클릭하여 우측 영역에 외부 앱의 액티비티인 계산기를 실행한다.

④     하지만 실행되지 않고 앱이 에러로 강제 종료되었다는 팝업을 보게 된다.

⑤     로그캣을 살펴보면 Exception 정보를 확인할 수 있다. 그 의미를 분석해보면 계산기는 다른 프로세스의 액티비티기 때문에 실행될 수 없다는 것이다. 정확한 이유는 [ REF \_Ref396505345 \r \h 그림 22-12] ⑦에서 설명되었다.

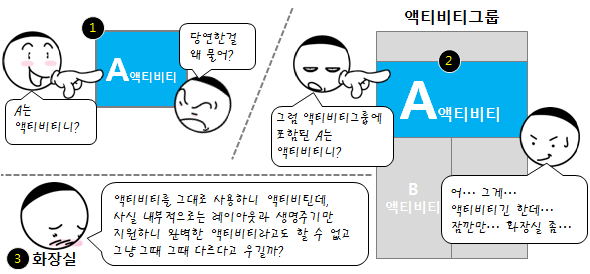
|  |
| --- |
| **그렇다면 절대 다른 앱의 액티비티는 액티비티그룹에서 실행될 수 없나?**    절대 불가능한 것은 아니다. 다른 앱을 나의 앱 프로세스에서 동작시키면 가능하다. 이를 위해서는 다른 앱과 나의 앱의 패키지를 동일한 키로 서명하고, AndroidManifest.xml 내 <manifest> 요소의 sharedUserId 속성 값을 두 앱 모두 같은 문자열을 적어주면 된다. 하지만 어떻게 남이 만든 앱에 그러한 코드를 추가하거나 서명할 수 있겠는가! 물론 두 가지 앱 모두 같은 회사나 개인이 만든 앱이라면 가능하겠다. |

**22.2.2 액티비티그룹의 장점과 문제점**

액티비티그룹의 가장 큰 장점은 아무리 레이아웃 구조가 복잡하더라도 액티비티그룹 내 소스가 분리된 독립적인 실행 단위인 액티비티 컴포넌트를 활용하기 때문에 관리가 용이하다는 점이다. 여기서 특히 액티비티그룹이 독립적인 액티비티 컴포넌트를 활용한다는 것은 매우 매력적인 구조다. 즉 액티비티 스스로도 실행될 수 있는 단위고, 액티비티그룹에 포함되어 실행될 수도 있기 때문에, 이미 액티비티 구현 방법을 익힌 개발자들은 추가적으로 액티비티그룹 사용법만 배우면 쉽게 응용할 수 있다. 그리고 안드로이드 구조적인 측면에서도 뷰그룹이 여러 개의 자식 뷰들을 포함하고, 뷰그룹도 뷰를 상속받기 때문에 또 다른 뷰그룹에 포함될 수 있는 것과 비교했을 때 액티비티그룹과 액티비티의 관계는 매우 자연스러워 보인다. 안드로이드도 처음에는 바로 이 구조가 매우 훌륭한 아이디어라고 생각했다. 하지만 다양한 사양의 안드로이드 단말이 출시되고 안드로이드 버전이 올라갈수록 조금씩 구조적 문제점이 드러나기 시작한다. 지금부터 액티비티그룹의 문제점을 살펴보자.

액티비티그룹은 자식 액티비티를 애매모호하게 만든다. 다음의 그림을 살펴보자.

그림 22-15 액티비티를 애매모호하게 만드는 액티비티그룹



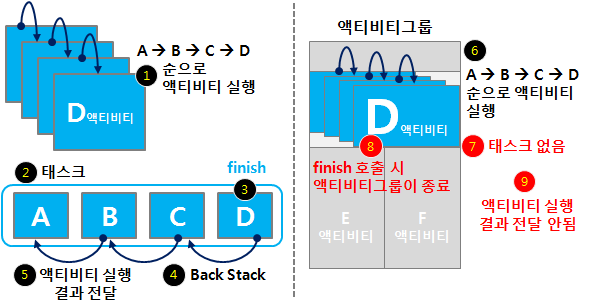
①     액티비티그룹에 포함될 수 있는 액티비티는 일반적인 액티비티와 전혀 다르지 않다. 즉 독립적으로 실행될 수 있는 컴포넌트라는 의미다.

②     그렇다면 액티비티그룹에 포함되었을 때 과연 액티비티라고 볼 수 있을까?

③     사실 액티비티가 액티비티그룹에 포함된 경우 완벽한 액티비티라고 볼 수 없다. 그 이유는 액티비티그룹과 합쳐져 버렸기 때문이다. 따라서 액티비티 스스로의 기능과 속성은 모두 사라지고 액티비티그룹에 귀속된다. 그리고 바로 이점이 액티비티를 애매모호하게 만드는 점이다.

그렇다면 액티비티그룹에 귀속된 자식 액티비티는 왜 액티비티라고 볼 수 없는지 [ REF \_Ref396747899 \r \h 그림 22-16]을통해 좀더 살펴보자.

그림 22-16 액티비티그룹의 문제점



①     일반적으로 액티비티를 A->B->C->D 순으로 실행하면,

②     태스크 하나가 생성되고 내부에는 A->B->C->D 순으로 액티비티 스택이 형성된다. 참고로 태스크에 대해서는 책의 12장에서 상세히 다루고 있다.

③     태스크의 가장 상단의 액티비티가 종료되면,

④     그 아래의 액티비티로 복귀되고 이어 복귀되는 액티비티를 종료해 나가면 D->C->B->A 순으로 액티비티가 복귀되는데 이를 태스크의 액티비티 백스택Back Stack이라고 한다.

⑤     또한 만일 B에서 A 액티비티로 복귀될 때, B의 결과를 A 액티비티로 전달하고 싶다면 B 액티비티 종료 시 setIntent 함수로 전달 데이터를 설정하면 된다. 물론 이를 위해 A가 B 액티비티를 실행할 때 startActivityForResult 함수로 B 액티비티를 실행했어야 한다. 참고로 이렇게 액티비티의 결과를 받는 것은 책의 13장에서 상세히 다루고 있다. 지금까지의 내용은 액티비티가 가진 특징이며, 본연의 기능 중 하나다. 뿐만 아니라 액티비티를 충분히 배우고 이해했던 개발자라면 당연하게 여겨지는 내용이기도 하다.

⑥     하지만 액티비티그룹에서 A->B->C->D 순으로 실행되는 자식액티비티는 다르다.

⑦     먼저 A->B->C->D 순의 액티비티 스택을 가지는 태스크가 없다. 즉 태스크 자체를 관리하지 않는다.

⑧     따라서 D 액티비티에서 finish 함수로 종료하면 C 액티비티로 복귀하지 않고, 아예 액티비티그룹이 종료해버린다. 이는 모든 액티비티가 하나의 액티비티그룹으로 합쳐졌기 때문이다.

⑨     또한 액티비티 스택이 형성되지 않기 때문에 상위 액티비티에서 하위 액티비티로 그 결과 데이터를 전달할 수도 없다. 물론 책의 19장에서 배운 애플리케이션 객체를 이용하여 액티비티간 데이터를 공유할 방법은 있다.

[ REF \_Ref396747899 \r \h 그림 22-16]의 내용을 통해 액티비티가 액티비티그룹에 포함되었을 경우 더 이상 액티비티 컴포넌트의 기능을 수행할 수 없는 것을 배웠다. 이 것 외에도 액티비티의 기능은 버전이 올라갈 수록 확장되고 액티비티그룹에는 반영되지 않았다. 그 이유는 액티비티그룹이 자식 액티비티에 요구하는 기능은 다음의 단 한 가지뿐이기 때문이다.

   ***별도의 소스로 분리된 독립적인 액티비티의 레이아웃과 생명주기를 가진다.***

이것 만을 위해 액티비티를 액티비티그룹이 포함하는 구조는 매우 비효율적이고 안드로이드 시스템 내부적으로도 유지보수하기도 쉽지 않다. 따라서 안드로이드는 API 13부터 액티비티그룹을 디플리케이티드 함으로써 과감히 버렸고, 프래그먼트라는 새로운 기능을 내 놓았다. 그리고 바로 그때가 액티비티 내 레이아웃을 분리하여 관리해야 할 필요성이 높아진 태블릿이 나오던 시기기도 하다. 태블릿은 화면이 커서 그 공간을 잘 분리해서 활용해야 했기 때문이다.

프래그먼트는 액티비티그룹이 자식 액티비티에 요구했던 기능만을 추출하여 구현되었기 때문에 좀더 가볍고 개념상으로도 전혀 애매모호하지 않다. 이제 프래그먼트에 대해서 배워보자. 액티비티그룹이 대체된 개념이므로 그렇게 생소하지 않을 것이다.

다음글에서 계속 됩니다.