**주의 : 소스 내용 중 "0nClick", "0nStart" 함수명 첫글자가 숫자 0인것은 오타가 아닙니다.**

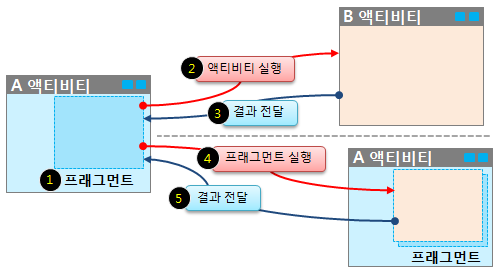
           다움 게시판은 o n C l i c k 라는 글을 입력할 수 없기 때문에 어쩔 수 없이

           영문 소문자 o를 숫자 0으로 대체하였습니다. ^^;

**22.3.8 프래그먼트에서 액티비티 혹은 다른 프래그먼트 실행과 데이터 전달**

액티비티에서 또 다른 액티비티를 실행할 수 있었듯 프래그먼트 역시 다른 액티비티 실행이 가능하다. 이뿐만 아니라 프래그먼트에서 또 다른 프래그먼트를 실행할 수도 있고, 프래그먼트와 실행되는 액티비티 혹은 프래그먼트 간 데이터도 주고 받을 수 있다. 다음 그림을 살펴보자.

그림 22-48 프래그먼트에서 액티비티 혹은 다른 프래그먼트 실행과 데이터 전달



①     만일 액티비티 내에 프래그먼트가 있을 때

②     해당 프래그먼트는 다른 액티비티를 실행할 수 있다. 이때 실행하는 방법은 액티비티가 다른 액티비티를 실행하는 방법과 같다. 그리고 액티비티를 실행할 때 인텐트를 통해 데이터도 전달할 수 있다.

③     실행된 액티비티가 종료될 때 그 결과 데이터를 프래그먼트가 받을 수 있다. 여기서 프래그먼트가 데이터를 받는 방법 또한 액티비티와 같다.

④     프래그먼트는 다른 프래그먼트를 실행할 수도 있다. 이때 실행하는 방법은 앞서 배웠던 프래그먼트매니저를 통해 추가(add)하는 방식이다. 그리고 실행할 때 데이터를 번들에 담아 전달한다.

⑤     실행된 프래그먼트가 종료하는 방법은 앞서 배웠던 백스택을 이용한다. 즉 프래그먼트매니저의 popBackStack함수를 이용한다는 의미다. 따라서 이 것이 가능하려면 ④에서 프래그먼트를 추가하는 트랜젝션에addToBackStack 함수가 추가되어야 한다. 그리고 프래그먼트가 종료될 때 결과 데이터를 프래그먼트가 받은 것은 안드로이드에서 제공하지 않는다. 따라서 직접 구현해야하는데, 그 방법은 액티비티가 결과 데이터를 받는 방법과 동일한 구조로 구현하는 것이 좋다. 상세한 내용은 예제를 통해 설명한다.

[그림 22-48]에 설명된 내용을 직접 예제로 구현해보자. 예제는 텍스트와 이미지뷰어 프래그먼트에서 다른 텍스트와 이미지를 변경하는 기능을 추가한다. 이때 변경되는 텍스트와 이미지 경로를 선택하기 위한 프래그먼트 혹은 액티비티를 실행하게 된다. 구현할 예제 구성은 [그림 22-49]와 같다.

그림 22-49 텍스트 혹은 이미지 경로를 얻어와 표시하는 예제



①     **먼저 프래그먼트에서 액티비티를 실행하고 데이터를 주고받는 과정의 예제다.** 텍스트뷰어 프래그먼트 레이아웃에는 다른 텍스트 선택하기 버튼과 그 아래 텍스트뷰 하나를 추가되었다.

②     텍스트뷰에는 불러올 텍스트 경로를 표시한다. 만일 다른 텍스트 경로를 얻어오면 텍스트뷰에 그 경로를 표시한다.

③     다른 텍스트 선택하기 버튼의 기능은 텍스트 경로를 선택하는 액티비티를 실행하는 것이다. 해당 버튼을 누른다.

④     텍스트 선택하기 액티비티가 실행되고, 화면에 텍스트 경로를 요청이라는 문자열이 표시된다. 이 문자열은 프래그먼트에서 텍스트 경로 요청이라는 정보를 전달받아 표시된다.

⑤     간단히 Uri 전달 버튼을 누르면 액티비티가 종료되고 특정 경로 정보를 프래그먼트로 전달한다.

⑥     프래그먼트로 복귀되고 전달받은 경로 정보를 텍스트뷰에 표시한다.

⑦     **다음은 프래그먼트에서 다른 프래그먼트를 실행하고 데이터를 주고받는 과정의 예제다.** 메뉴 리스트 프래그먼트에서 이미지 보기 아이템을 선택한다.

⑧     이미지뷰어 프래그먼트가 보인다. 이미지뷰어 프래그먼트 레이아웃에는 텍스트뷰어 프래그먼트와 같이 버튼과 텍스트뷰가 추가되었다.

⑨     텍스트뷰에는 불러올 이미지 경로를 표시한다. 만일 다른 이미지 경로를 얻어오면 텍스트뷰에 그 경로를 표시한다.

⑩     다른 이미지 선택하기 버튼의 기능은 이미지 경로를 선택하는 프래그먼트를 실행하는 것이다. 해당 버튼을 누른다.

⑪     이미지 선택하기 프래그먼트가 이미지뷰어 프래그먼트 위에 실행되고, 화면에 이미지 경로를 요청이라는 문자열이 표시된다. 이 문자열은 이전 프래그먼트에서 이미지 경로 요청이라는 정보를 전달받아 표시된다.

⑫     간단히 Uri 전달 버튼을 누르면 프래그먼트가 종료되고 특정 경로 정보를 이전 프래그먼트로 전달한다.

⑬     프래그먼트로 복귀되고 전달받은 경로 정보를 텍스트뷰에 표시한다.

[그림 22-49]은 비록 이미지나 텍스트를 변경하진 않지만 프래그먼트에서 액티비티 혹은 프래그먼트를 실행하고 서로 데이터를 주고받는 좋은 예가 된다.

텍스트뷰어 프래그먼트에서 액티비티를 실행하고 데이터를 주고받는 예제를 구현해보자. 먼저 실행될 액티비티와 레이아웃을 추가한다.

예제 22-44 경로 선택 액티비티 레이아웃 XML

|  |
| --- |
| **res/layout/select\_uri\_layout.xml**    <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:orientation=*"vertical"*      android:background=*"#FFF"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*>        <!-- 요청한 경로 타입(image/text)을 표시하기 위한 텍스트뷰 -->  **<TextView android:id=*"@+id/request\_uri\_title"***  android:layout\_width=*"match\_parent"*  android:layout\_height=*"wrap\_content"*/>       <!-- 요청한 경로를 전달하기 위한 버튼 -->  **<Button android:id=*"@+id/response\_uri\_btn"***  android:layout\_width=*"match\_parent"*  android:layout\_height=*"wrap\_content"*  android:text=*"Uri 전달"*  android:0nClick=*"0nClick"*/>    </LinearLayout> |

예제 22-45 경로 선택 액티비티

|  |
| --- |
| **src/SelectUriActivity.java**    **public** **class** SelectUriActivity **extends** Activity {        String mUriType = "";      String mResponseUri = "";        @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);            setContentView( R.layout.*select\_uri\_layout* );            // ① 프래그먼트로 부터 전달받은 Uri 타입을 인텐트 엑스트라로 부터 추출한다.  // ====================================================================  **Intent intent = getIntent();**  **mUriType = intent.getStringExtra( "UriType" );**          // ====================================================================            // ② 전달받은 Uri 타입에 따라 텍스트뷰에 보여질 내용을 달리한다.          //    텍스트뷰에는 프래그먼트의 요청 타입을 출력하게 된다.          //    그리고 미리 프래그먼트로 전달할 Uri를 설정해둔다.          // ====================================================================          TextView tv = (TextView)findViewById( R.id.*request\_uri\_title* );    **if( "image".equals( mUriType ) == true )**  **{**  **mResponseUri = "/images/1.jpg";**  **tv.setText( "이미지 경로를 요청" );**  **}**  **else if( "text".equals( mUriType ) == true )**  **{**  **mResponseUri = "/text/1.txt";**  **tv.setText( "텍스트 경로를 요청" );**  **}**          // ====================================================================      }    **public** **void** 0nClick( View v )      {          // ③ Uri 전달 버튼을 누르면 액티비티를 실행한 프래그먼트로 결과를 전달한다.          //    결과 값은 인텐트를 통해 전달한다. 그리고 프래그먼트로 액티비티의 결과값을          //    전달하는 방법은 액티비티로 결과값을 전달하는 방법과 동일하다.          // ====================================================================  **Intent responseIntent = new Intent();**  **responseIntent.putExtra( "ResponseUri", mResponseUri );**  **setResult( *RESULT\_OK*, responseIntent );**  **finish();**          // ====================================================================      }  } |

①     해당 액티비티는 요청 타입에 따라 이미지 혹은 텍스트의 경로를 전달할 수 있다. 요청 타입은 이 액티비티를 실행하는 곳에서 인텐트의 엑스트라를 통해 전달 받는다.

②     요청한 경로 타입에 따라 전달할 경로를 설정하고, 텍스트뷰에 요청한 경로 타입을 표시한다. 표시될 텍스트뷰는 [그림 22-49] ④와 같다.

③     Uri 전달 버튼을 누르면 전달할 경로를 인텐트 엑스트라에 설정한다. 설정된 인텐트는 setResult 함수를 통해 프래그먼트로 전달할 준비를 마친다. 마지막으로 finish 함수로 해당 액티비티를 종료하면 설정된 인텐트는 전달된다.

다음은 구현된 경로 선택 액티비티를 AndroidManifest.xml에 등록한다.

예제 22-46 AndroidManifest.xml에 경로 선택 액티비티 추가

|  |
| --- |
| **AndroidManifest.xml**        ...      <application          android:allowBackup=*"true"*          android:icon=*"@drawable/ic\_launcher"*          android:label=*"@string/app\_name"*          android:theme=*"@style/AppTheme"* >            ...  **<activity android:name=*".SelectUriActivity"*/>**         ... |

이제 경로 선택 액티비티를 사용할 준비가 끝났다. 다음은 텍스트뷰어 프래그먼트에서 구현된 경로 선택 액티비티를 실행하도록 구현한다. 먼저 텍스트뷰어 프래그먼트 레이아웃을 수정하자.

예제 22-47 텍스트뷰어 프래그먼트 레이아웃 XML

|  |
| --- |
| **res/layout/fragment\_text\_viewer.xml**    <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:orientation=*"vertical"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*>        ...      <!-- 텍스트 경로를 요청하는 버튼이다. -->  **<Button android:id=*"@+id/request\_text\_uri\_btn"***          android:layout\_width=*"match\_parent"*          android:layout\_height=*"wrap\_content"*          android:text=*"다른 텍스트 선택하기"*/>       <!-- 요청한 경로를 전달 받아 그 경로를 보여주기 위한 텍스트뷰다.-->  **<TextView android:id=*"@+id/show\_response\_text\_uri"***          android:layout\_width=*"match\_parent"*          android:layout\_height=*"wrap\_content"*          android:text=*"미전달"*/>  </LinearLayout> |

다음은 텍스트뷰어 프래그먼트를 수정한다.

예제 22-48 텍스트뷰어 프래그먼트

|  |
| --- |
| **src/TextViewerFragment.java**    **public** **class** TextViewerFragment **extends** Fragment **implements** 0nClickListener {    **private static final int *REQUEST\_CODE\_REQUEST\_TEXT\_URI* = 1;**      ...      @Override  **public** View onCreateView( LayoutInflater inflater,                                   ViewGroup container,                                   Bundle savedInstanceState )      {          View view =  inflater.inflate(  R.layout.*fragment\_text\_viewer*, container, **false**);          ...          // ① 텍스트 URI를 요청하는 버튼에 클릭 리스너를 설정한다.          // ====================================================================          Button requestUriBtn = (Button)view.findViewById(                                                     R.id.*request\_text\_uri\_btn* );          requestUriBtn.set0nClickListener( **this** );          // ====================================================================    **return** view;      }        @Override  **public** **void** 0nClick( View v )      {  **switch** (v.getId())          {  ...  **case** R.id.*request\_text\_uri\_btn*:              {                  // ② 액티비티를 실행하기 위한 인텐트를 생성한다.                  //    인텐트를 생성할 때 인자로 실행될 액티비티의 클래스를 설정한다.  **Intent intent = new Intent( getActivity(),**  **SelectUriActivity.class );**                    // ③ 실행될 액티비티에 전달할 값을 인텐트 엑스트라에 설정한다.                  //    전달할 값은 요청할 Uri의 타입이다.                  //    실행되는 프래그먼트를 해당 타입을 보고 이미지 혹은 택스트의 Uri를                  //    반환하게 된다.  **intent.putExtra( "UriType", "text" );**                    // ④ 프래그먼트에서 액티비티를 실행하는 방법은 액티비티에서                  //    다른 액티비티를 실행하는 방법과 같다.                  //    따라서 startActivity 함수로 액티비티를 실행할 수 있다.                  //    그런데 실행되는 액티비티로 부터 결과를 받아야 하기 때문에                  //    startActivityForResult 함수를 사용한다.  **startActivityForResult( intent,**  ***REQUEST\_CODE\_REQUEST\_TEXT\_URI* );**  **break**;              }          }      }        // ⑤ 실행된 액티비티 결과를 받기 위해 onActivityResult 함수를 재정의 한다.      //    프래그먼트에서 실행한 액티비티의 결과를 받는 방법 역시 액티비티 간에 결과를      //    받는 방법과 같다. 따라서 액티비티와 같이 onActivityResult 함수만 재정의      //    하면 된다.      @Override  **public** **void** onActivityResult( **int** requestCode, **int** resultCode, Intent data )      {  **if( requestCode == *REQUEST\_CODE\_REQUEST\_TEXT\_URI* &&**  **resultCode == Activity.*RESULT\_OK* )**          {              // ⑥ 실행된 액티비티의 Uri 결과를 받아 텍스트뷰에 출력한다.              // ================================================================  **String responseUri = data.getStringExtra( "ResponseUri" );**              TextView tv = (TextView)getView().findViewById(                                                   R.id.*show\_response\_text\_uri* );              tv.setText( responseUri );              // ================================================================          }  **super**.onActivityResult( requestCode, resultCode, data );      }      ... |

①     다른 텍스트 선택하기 버튼에 클릭 리스너를 설정한다. 참고로 해당 버튼은 [그림 22-49] ③과 같다.

②     다른 텍스트 선택하기 버튼이 클릭되면 경로 선택 액티비티를 실행한다. 이를 위해 경로 선택 액티비티 실행에 필요한 인텐트를 생성 및 설정한다. 참고로 인텐트 생성자의 첫번째 인자에는 액티비티 컨텍스트가 필요한데 프래그먼트에서 getActivity 함수를 이용하면 참조할 수 있다. getActivity를 통해 참조되는 액티비티는 프래그먼트가 소속된 부모 액티비티다.

③     경로 선택 액티비티가 실행할 때는 어떤 타입의 경로를 요청하는 것인지 알려주어야 한다. 이를 위해 인텐트의 엑스트라에 경로 타입을 설정한다. 참고로 경로 타입에 사용되는 키와 값은 경로 선택 액티비티에서 정의하는 것을 사용해야 한다. [예제 22-45]의 경로 선택 액티비티는 경로 타입의 키값으로 UriType 문자열을 사용하고,타입 값은 텍스트의 경우 text 그리고 이미지의 경우 image 문자열을 사용한다.

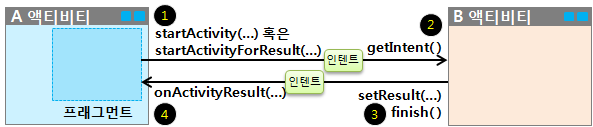
④     이제 경로 선택 액티비티를 실행하기 위한 준비가 끝났다. 마지막으로 액티비티를 실행하면 되는데 프래그먼트도 액티비티와 동일한 방법으로 액티비티를 실행할 수 있다. 즉 startActivity 함수 혹은 startActivityForResult함수를 이용하면 된다는 의미다. 프래그먼트 클래스에도 해당 함수를 지원한다.

⑤     onActivityResultResult 재정의 함수를 통해 경로 선택 액티비티의 결과를 전달받을 수 있다. 이 함수는startActivityForResult 함수로 액티비티를 실행해야만 호출된다.

⑥     선택 액티비티가 전달한 인텐트 엑스트라에서 선택된 경로를 추출하여 텍스트뷰에 출력한다.

지금까지 구현한 프래그먼트에서 액티비티를 실행하고 결과를 받는 과정은 액티비티와 모두 동일하다. 따라서 생소할 것이 없을 것이다. 다시 한번 다음의 그림으로 정리해보자.

그림 22-50 프래그먼트에서 액티비티를 실행하고 결과를 받는 과정



①     프래그먼트에서 B액티비티를 실행할 때는 startActivity 혹은 startActivityForResult 함수를 사용한다. 이때 실행될 B액티비티로 전달할 데이터가 있다면 인텐트에 담아서 보내면 된다.

②     실행된 B액티비티는 getIntent 함수를 통해 프래그먼트가 전달한 인텐트를 받을 수 있다.

③     실행된 B액티비티가 모든 역할을 수행하고 종료할 때 자신을 실행한 프래그먼트로 결과값을 전달할 수 있다.이때도 인텐트를 통해 데이터를 전달하는데 setResult 함수로 전달할 인텐트를 설정하면 된다. 마지막으로finish 함수를 호출하면 B액티비티를 종료되고, 결과 인텐트도 전달된다.

④     B액티비티가 전달한 인텐트는 프래그먼트의 onActivityResult 재정의 함수로 전달된다.

이제 [그림 22-49] ①부터 ⑥까지의 과정이 구현되었다. 다음은 이미지뷰어 프래그먼트에서 경로선택 프래그먼트를 실행하고 데이터를 주고받는 예제를 구현해보자. 먼저 실행될 프래그먼트를 구현한다. 참고로 경로 선택 프래그먼에서 사용되는 레이아웃은 구현하지 않고 [예제 22-45]의 액티비티 레이아웃을 활용하도록 한다.

예제 22-49 경로 선택 프래그먼트

|  |
| --- |
| **src/SelectUriFragment.java**    **public** **class** SelectUriFragment **extends** Fragment **implements** 0nClickListener {        String mUriType = "";      String mResponseUri = "";    **public** **static** SelectUriFragment newInstance()      {          SelectUriFragment f = **new** SelectUriFragment();  **return** f;      }        @Override  **public** **void** onCreate( Bundle savedInstanceState )      {          // ① 프래그먼트로 부터 전달받은 Uri 타입을 추출한다.          // ====================================================================  **Bundle argBundle = getArguments();**  **mUriType = argBundle.getString( "UriType" );**          // ====================================================================    **super**.onCreate( savedInstanceState );      }        @Override  **public** View onCreateView( LayoutInflater inflater,                                ViewGroup container,                                Bundle savedInstanceState )      {          View view =  inflater.inflate( R.layout.*select\_uri\_layout*,                                         container,  **false**);            // ② 전달받은 Uri 타입에 따라 텍스트뷰에 보여질 내용을 달리한다.          //    텍스트뷰에는 프래그먼트의 요청 타입을 출력하게 된다.          //    그리고 미리 프래그먼트로 전달할 Uri를 설정해둔다.          // ====================================================================          TextView tv = (TextView)view.findViewById( R.id.*request\_uri\_title* );    **if**( "image".equals( mUriType ) == **true** )          {              mResponseUri = "/images/1.jpg";              tv.setText( "이미지 경로를 요청" );          }  **else** **if**( "text".equals( mUriType ) == **true** )          {              mResponseUri = "/text/1.txt";              tv.setText( "텍스트 경로를 요청" );          }          // ====================================================================            // ③ Uri 전달 버튼에 클릭 리스너를 설정한다.          // ====================================================================          Button responseBtn = (Button)view.findViewById( R.id.*response\_uri\_btn* );          responseBtn.set0nClickListener( **this** );          // ====================================================================    **return** view;      }    **public** **void** 0nClick( View v )      {  // ④ 현재 프래그먼트를 종료한다.          //    프래그먼트를 종료하기 위해 프래그먼트가 포함된 부모 액티비티의          //    onBackPressed 함수를 호출한다. 그 이유는 해당 함수 내부에 백스택을          //    처리하는 popBackStack 함수가 포함되어 있기 때문이다.  **getActivity().onBackPressed();**            // ⑤ Uri 전달 버튼을 누르면 프래그먼트로 결과를 전달한다.          //    결과 전달 방법은 액티비티의 결과 전달 방법과 동일하게 맞춘다.          //    프래그먼트가 액티비티의 결과를 받는 코드와 동일하게 구조를 일치시키면          //    재사용성을 높힐 수 있기 때문이다.          // ====================================================================          Intent responseIntent = **new** Intent();          responseIntent.putExtra( "ResponseUri", mResponseUri );  **getTargetFragment().onActivityResult( getTargetRequestCode(),**  **Activity.*RESULT\_OK*, responseIntent );**          // ====================================================================    }  } |

①     해당 프래그먼트는 요청 타입에 따라 이미지 혹은 텍스트의 경로를 전달할 수 있다. 요청 타입은 이 프래그먼트를 실행하는 곳에서 번들 객체에 담아 전달한다. 전달한 번들은 getArguments 함수를 통해 참조할 수 있다.

②     요청한 경로 타입에 따라 전달할 경로를 설정하고, 텍스트뷰에 요청한 경로 타입을 표시한다. 표시될 텍스트뷰는 [그림 22-49] ⑪과 같다.

③     Uri 전달 버튼에 클릭 리스너를 설정한다.

④     Uri 전달 버튼을 누르면 현재 프래그먼트를 종료한다. 종료 방법은 이전키를 누르는 효과를 활용한다. 소스를 통해 이전키를 처리하는 방법은 프래그먼트가 속한 액티비티의 onBackPressed 함수를 호출하면 된다. 해당 함수가 호출되면 프래그먼트매니저의 popBackStack 함수가 호출되고 이전 프래그먼트로 전환된다.

⑤     그런데 아직 이전 프래그먼트로 전달할 경로를 설정하지 않았다. 이를 위해 인텐트를 하나 생성하고 엑스트라에 설정한다. 그런데 반드시 인텐트로 전달할 데이터를 만들 필요는 없다. 다만 액티비티의 결과 전달 방식과 맞추기 위해 인텐트를 사용할 뿐이다. 이게 무슨 말일까? 사실 안드로이드에 프래그먼트의 실행 결과를 실행한 프래그먼트로 전달하는 기능은 제공되지 않는다. 그렇다고 방법이 없는 것은 아니다. 실행된 프래그먼트는 실행한 프래그먼트 객체를 참조할 수 있는데 그 객체의 특정 함수를 호출하여 결과를 전달하면 된다. 좀더 쉽게 말하자면 실행한 프래그먼트가 자신 클래스에 특정 함수를 구현하고 실행된 프래그먼트에게 ‘결과는 이 함수를 호출해서 전달해줘!’ 라고 약속하는 것이다. 그럼 실행된 프래그먼트는 전달할 결과 값을 실행한 프래그먼트 함수를 호출함으로써 전달된다. 물론 전달 값은 해당 함수의 인자를 이용한다. 그런데 서로 약속된 결과 함수 타입을 맘대로 정하는 것은 일관성이 없을 수 있다. 따라서 기존 액티비티 결과가 전달되는 onActivityResult 함수 타입을 그대로 활용하여 구조의 일관성과 재사용성을 동시에 높힐 수 있다.

getTargetFragment 함수는 자신을 실행한 프래그먼트 객체를 참조할 수 있다. 따라서 해당 함수를 통해onActivityResult 함수를 직접 호출하고 인자로 결과 인텐트를 전달한다. 그런데 onActivityResult의 인자로 자신을 실행한 프래그먼트의 요청코드 값이 필요하다. 그 값은 getTargetRequestCode 함수를 통해 참조할 수 있다.

어쨌든 이 부분은 해당 프래그먼트를 실행하는 이미지뷰어 프래그먼트 구현부를 보면 좀더 이해가 쉽다. 따라서 구현부인 [예제 22-51]에서 좀더 살펴보자.

이제 경로 선택 프래그먼트를 사용할 준비가 끝났다. 다음은 이미지뷰어 프래그먼트에서 구현된 경로 선택 프래그먼트를 실행하도록 구현한다. 먼저 이미지뷰어 프래그먼트 레이아웃을 수정하자.

예제 22-50 이미지뷰어 프래그먼트 레이아웃 XML

|  |
| --- |
| **res/layout/fragment\_image\_viewer.xml**    <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:orientation=*"vertical"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*>    <!-- 이미지 경로를 요청하는 버튼이다. -->  **<Button android:id=*"@+id/request\_image\_uri\_btn"***  android:layout\_width=*"match\_parent"*          android:layout\_height=*"wrap\_content"*          android:text=*"다른 이미지 선택하기"*/>         <!-- 요청한 경로를 전달 받아 그 경로를 보여주기 위한 텍스트뷰다.-->  **<TextView android:id=*"@+id/show\_response\_image\_uri\_tv"***          android:layout\_width=*"match\_parent"*          android:layout\_height=*"wrap\_content"*          android:text=*"미전달"*/>    <ImageView              android:layout\_width=*"match\_parent"*              android:layout\_height=*"match\_parent"*              android:src=*"@drawable/ic\_launcher"*/>    </LinearLayout> |

다음은 이미지뷰어 프래그먼트를 수정한다.

예제 22-51 이미지뷰어 프래그먼트

|  |
| --- |
| **src/ImageViewerFragment.java**    **public** **class** ImageViewerFragment **extends** Fragment **implements** 0nClickListener{    **private** **static** **final** **int** *REQUEST\_CODE\_REQUEST\_IMAGE\_URI* = 1;        @Override  **public** View onCreateView( LayoutInflater inflater, ViewGroup container,                                Bundle savedInstanceState )      {          View view = inflater.inflate( R.layout.*fragment\_image\_viewer*,                                   container,  **false**);            // ① 이미지 URI를 요청하는 버튼에 클릭 리스너를 설정한다.          // ====================================================================  Button requestUriBtn = (Button)view.findViewById(                                                    R.id.*request\_image\_uri\_btn* );          requestUriBtn.set0nClickListener( **this** );          // ====================================================================    **return** view;      }        @Override  **public** **void** 0nClick( View v )      {          // ② 실행할 프래그먼트를 객체를 생성한다.          SelectUriFragment selectUriFragment = SelectUriFragment.*newInstance*();            // ③ 실행될 프래그먼트에 전달할 인자 값을 설정한다.          //    전달할 값은 요청할 Uri의 타입이다.          //    실행되는 프래그먼트를 해당 타입을 보고 이미지 혹은 택스트의 Uri를          //    반환하게 된다.          // ====================================================================  **Bundle argsBundle = new Bundle();**  **argsBundle.putString( "UriType", "image" );**  **selectUriFragment.setArguments( argsBundle );**          // ====================================================================            // ④ 실행될 프래그먼트에 실행하는 프래그먼트 객체와 요청 코드를 설정한다.          //    이후 실행되는 프래그먼트는 이 설정으로 실행되는 프래그먼트는 자신을 실행한          //    프래그먼트를 참조하거나 실행한 프래그먼트의 요청코드를 참조할 수 있다.          // ====================================================================  **selectUriFragment.setTargetFragment( this,**  ***REQUEST\_CODE\_REQUEST\_IMAGE\_URI* );**          // ====================================================================            // ⑤ 화면 우측에 새로운 프래그먼트를 실행한다.          //    그런데 중요한 것은 addToBackStack 함수를 트랜잭션에 포함하는 것이다.          //    백스택에 트랜잭션을 추가해야만 실행되는 프래그먼트가 종료되고          //    그 결과를 실행한 프래그먼트가 받을 수 있기 때문이다.          // ====================================================================          getFragmentManager()            .beginTransaction()  **.add( getId(), selectUriFragment )**  **.addToBackStack( null )**            .commit();          // ====================================================================      }        // ⑥ 실행된 프래그먼트의 결과를 받기 위한 함수를 정의한다.      //    이 함수는 실행된 프래그먼트가 종료 시점에 호출해주게 된다. 따라서 이 함수명을      //    변경해도 실행된 프래그먼트가 변경된 함수를 호출하면 되니 꼭 onActivityResult      //    함수명을 사용하지 않아도 된다.      //    하지만 프래그먼트가 액티비티의 결과를 받는 코드와 동일하게 구조를 일치시키면      //    재사용성을 높힐 수 있다.      @Override  **public void onActivityResult( int requestCode, int resultCode,**  **Intent data )**      {  **if**( requestCode == *REQUEST\_CODE\_REQUEST\_IMAGE\_URI* &&              resultCode == Activity.*RESULT\_OK* )          {              // ⑦ 실행된 프래그먼트 Uri 결과를 받아 텍스트뷰에 출력한다.              // ================================================================              String responseUri = data.getStringExtra( "ResponseUri" );              TextView tv = (TextView)getView().findViewById(                                               R.id.*show\_response\_image\_uri\_tv* );              tv.setText( responseUri );              // ================================================================          }    **super**.onActivityResult( requestCode, resultCode, data );      }  **...** |

①     다른 이미지 선택하기 버튼에 클릭 리스너를 설정한다. 참고로 해당 버튼은 [그림 22-49] ⑩과 같다.

②     다른 이미지 선택하기 버튼이 클릭되면 경로 선택 프래그먼트를 실행한다. 이를 위해 경로 선택 프래그먼트를 생성한다.

③     다음은 실행될 프래그먼트에 전달할 경로 타입 정보를 설정한다. 이 정보는 생성된 프래그먼트에 미리 설정해두는 방법을 사용한다. 즉 번들 객체에 정보를 담아 경로 선택 프래그먼트의 setArguments 함수를 통해 설정해두는 것이다. 그리고 설정된 번들은 경로 선택 프래그먼트가 실행될 때 getArguments 함수를 통해 참조한다.

④     경로 선택 프래그먼트에 미리 설정해야할 정보 한 가지가 더 있다. 바로 자신을 실행한 프래그먼트 객체와 요청 코드다. 이 값은 경로 선택 프래그먼트에서 결과를 전달하기 위해 필요하다. 따라서 결과를 전달할 필요가 없다면 꼭 설정할 필요는 없다. 이 값을 설정하기 위해 setTargetFragment 함수를 사용한다. 이 과정은 [예제 22-49]⑤에서 활용되었다.

⑤     이제 경로 선택 프래그먼트를 실행한다. 실행한다고 해서 특별한 것이 아니다. 이미 배운 프래그먼트매니저를 이용해 add 함수로 프래그먼트를 추가하면 된다. 참고로 add 함수의 첫 번째 인자로 프래그먼트 컨테이너 ID를 필요로 한다. 이 값은 getId 함수를 통해 참조할 수 있다.

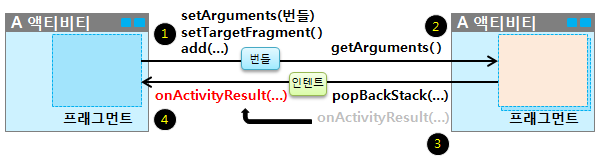
프래그먼트를 추가하는 트랜잭션에는 꼭 addToBackStack 함수를 포함시켜야 한다. 해당 트랜잭션을 백스택에 추가하지 않으면 경로 선택 프래그먼트가 종료되고 해당 프래그먼트가 복귀될 수 없기 때문이다.

⑥     다음은 경로 선택 프래그먼트가 종료될 때 그 결과값을 전달 받기 위한 함수를 구현한다. 해당 함수는 경로 선택 프래그먼트가 종료될 때 호출해준다. 참고로 [예제 22-49] ⑤에서 호출해주는 것을 확인할 수 있다. 그런데 이 함수명을 액티비티의 결과를 전달받는 onActivityResult 함수를 사용하고 있다. 이는 앞서 설명했듯 액티비티와 프래그먼트의 결과 처리 구조를 통일시키기 위함이다.

⑦     마지막으로 실행된 프래그먼트 경로를 받아 텍스트뷰에 출력한다.

지금까지 구현한 프래그먼트에서 프래그먼트를 실행하고 결과를 받는 과정을 다시 한번 다음의 그림으로 정리해보자.

그림 22-51 프래그먼트에서 액티비티를 실행하고 결과를 받는 과정



①     프래그먼트에서 프래그먼트를 실행할 때는 프래그먼트매니저의 add 함수를 사용한다. 사실 실행의 개념이 아니라 프래그먼트를 추가하는 것이다. 이때 추가되는 프래그먼트에 전달하고 싶은 정보는 setArguments 함수를 이용한다. 그리고 setTargetFragment 함수를 통해 실행하는 프래그먼트 객체를 설정한다.

②     실행된 프래그먼트는 getArgumnet 함수를 통해 자신을 실행한 프래그먼트가 전달한 번들을 참조할 수 있다.

③     프래그먼트가 종료할 때는 백스택을 이용한다. popBackStack 함수를 호출하면 현재 프래그먼트는 제거되고 이전 프래그먼트로 이동하는 것이다. 그런데 popBackStack 함수를 직접 호출하는 것보다 액티비티의onBackPressed를 사용하는 것이 직관적이다. 그리고 프래그먼트의 결과는 자신을 실행한 프래그먼트의onActivityResult 함수 호출을 통해 전달한다.

④     이전 프래그먼트로 전환되고 결과는 onActivityResult 함수를 통해 전달 받았다. 다시 한번 말하지만onActivityResult 함수는 개발자가 정하기 나름이다. 만일 A라는 이름으로 구현하고 실행된 프래그먼트가 결과를 전달할 때 A라는 함수를 호출해주면 된다. 하지만 액티비티의 결과를 받는 구조와 동일하게 onActivityResult함수 형태를 따르는 것이 일관성 있고 유연하다.

모든 구현을 끝났다. 예제를 실행하면 [그림 22-49]의 과정과 같이 동작하는 것을 확인할 수 있다.