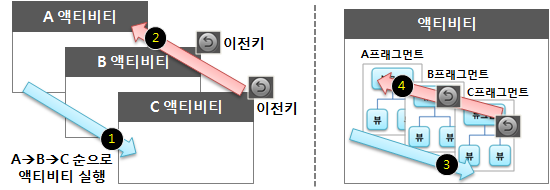
**22.3.5 프래그먼트 백스택**

 프래그먼트는 액티비티와 같이 백스택BackStack을 지원한다. 무슨 의미인지 다음의 그림을 통해 이해해보자.

그림 22-28 액티비티 백스택과 프래그먼트 백스택



①     만일 A, B, C액티비티를 순서대로 실행하면 A->B->C 순으로 스택을 형성한다. 이를 태스크의 액티비티 스택이라고 하는데, 태스크에 대해서는 12장에서 설명되었다.

②     A->B->C 순으로 액티비티 스택을 형성한 태스크는 이전키 등을 눌러 현재 액티비티를 종료하면 C->B->A액티비티 순으로 복귀한다. 이를 액티비티 백스택이라 하며, 태스크의 기본 개념이기도 하다.

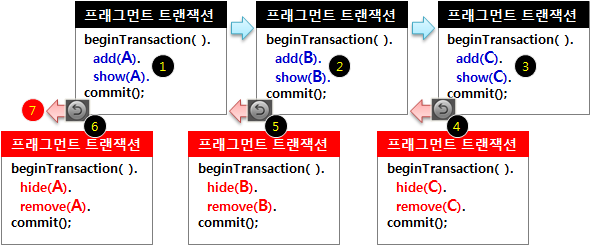
③     프래그먼트에도 백스택 기능을 가지고 있다. 만일 A, B, C프래그먼트를 순서대로 실행하면 A->B->C 순으로 스택을 형성한다.

④     이 상태에서 액티비티와 같이 이전키를 누르면 C->B->A프래그먼트 순으로 복귀한다.

**백스택의 원리**

저자가 [ REF \_Ref398751551 \r \h 그림 22-28] ③의 설명에서 프래그먼트를 ‘실행’이라고 표현했다. 하지만 프래그먼트는 실행이라는 기능이 없기 때문에 좀 애매하게 여겨질 수 있다. 그렇다면 프래그먼트의 실행이란 어떤 의미일까? 프래그먼트는 추가(add), 제거(remove), 교체(replace), 떼어내기(detach), 붙이기(attach), 숨기기(hide), 보이기(show) 등의 기능이 있고, 이는 프래그먼트 트랜잭션 단위로 처리된다. 저자는 바로 이 트랜잭션 단위로 처리되는 것을 실행이라 한 것이다. 다시 말하면 프래그먼트는 하나의 프래그먼트 트랜잭션 단위로 실행된다는 의미다. 그렇다면 A->B->C 순으로 트랜잭션이 실행되고, 이전키를 통해 C->B->A 순으로 복귀되는 과정을 [ REF \_Ref398752346 \r \h 그림 22-29]으로 정리해보자. 그림에서는 백스택 원리에 대해서도 같이 설명한다.

그림 22-29 프래그먼트 백스택 단위와 원리



①     먼저 A 프래그먼트를 화면에 보여주기 위해 beginTransaction 함수로 트랜잭션 하나를 생성한다. 생성된 프래그먼트 트랜잭션에 A프래그먼트를 추가, 보이기를 설정한다. 참고로 A프래그먼트를 추가하면 기본으로 화면에 보이기 때문에 보이기 위한 함수 show(A)는 필요 없다. 다만 프래그먼트 원리를 위해 추가한 것이다.

②     ①과 동일한 방법으로 B프래그먼트를 보여준다.

③     ①과 동일한 방법으로 C프래그먼트를 보여준다.

④     이제 C 프래그먼트가 보이는 상태에서 이전키를 누르면 바로 이전 B프래그먼트가 보이게 된다. 그 원리를 살펴보면, 프래그먼트매니저는 실행된 프래그먼트 트랜잭션을 모두 저장하고 있다가, 이전키가 눌러지면 제일 마지막에 실행된 프래그먼트 트랜잭션 내용을 반대로 처리한다. 마지막 트랜잭션은 ③과 같이 C프래그먼트를 add, show하고 있다. 따라서 이전키를 누르면 C프래그먼트를 hide, remove한다. 정말 간단한 원리가 아닌가! 참고로 add, attach, show의 반대는 각각 remove, detach, hide다.

⑤     계속 이전키를 누르면 ④와 동일한 방법대로 B는 사라지고 A프래그먼트로 복귀한다.

⑥     계속 이전키를 누르면 ④와 동일한 방법대로 A는 사라지고 이전 프래그먼트로 복귀한다.

⑦     백스택에 더 이상 프래그먼트 트랜잭션이 없는 상태에서 이전키를 누르면, 액티비티가 종료된다.

**백스택 적용**

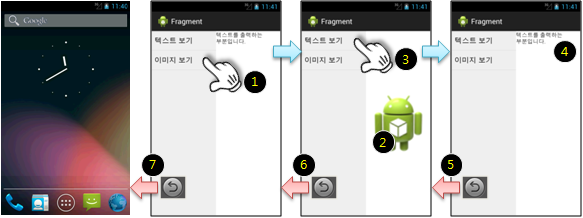
[ REF \_Ref398754435 \r \h 그림 22-29] ④의 설명에서 프래그먼트매니저는 백스택 기능을 위해 처리된 프래그먼트 트랜잭션을 모두 저장해둔다 했다. 하지만 이는 잘못된 설명이다. 사실 트랜잭션은 개발자가 선택적으로 프래그먼트매니저에 저장한다. 상세한 내용은 예제를 통해 이해하자. 액티비티 예제 소스를 수정한다.

예제 22-31 프래그먼트 백스택 적용

|  |
| --- |
| **src/MainActivity.java**    **public** **class** MainActivity **extends** Activity {        @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);            ...            listMenuFragment.setOnListItemClickListener(  **new** ListMenuFragment.OnListItemClickListener()          {              @Override  **public** **void** onItemClick( **int** itemType )              {                  TextViewerFragment textViewerFragment = (TextViewerFragment)                      getFragmentManager().findFragmentByTag( "TEXT\_VIEWER" );                  ImageViewerFragment imageViewerFragment = (ImageViewerFragment)                      getFragmentManager().findFragmentByTag( "IMAGE\_VIEWER" );    **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_TEXT\_VIEWER*)                  {                      getFragmentManager()                          .beginTransaction()                          .hide( imageViewerFragment )                          .show( textViewerFragment )                          // 백스택을 위해 해당 트랜잭션을 저장한다.  **.addToBackStack( "TEXT\_VIEWER\_BACKSTACK" )**                      .commit();                  }  **else** **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_IMAGE\_VIEWER* )                  {                      getFragmentManager()                      .beginTransaction()                          .hide( textViewerFragment )                          .show( imageViewerFragment )  // 백스택을 위해 해당 트랜잭션을 저장한다.  **.addToBackStack( "IMAGE\_VIEWER\_BACKSTACK" )**                      .commit();                  }              }          });      }  } |

[예제 22-31]을 살펴보면, 택스트뷰어, 이미지뷰어 프래그먼트를 보여주는 트랜잭션에 각각에 addToBackStack 함수를 추가했다. 바로 이 함수가 프래그먼트매니저에 해당 트랜잭션을 저장하도록 한다. 저장된 프래그먼트 트랜잭션은 이전키를 누를 때 프래그먼트매니저가 이전으로 복귀시켜준다. 만일 addToBackStack 함수를 추가하지 않으면 백스택은 동작하지 않는다. 참고로 addToBackStack의 인자는 백스택에 추가된 트랜잭션을 구분할 문자열을 넣는다. 이에 대한 상세한 설명은 예제를 실행하고 결과를 확인한 다음 설명한다.

그림 22-30 프래그먼트 백스택 예제 실행



①     이미지 보기 버튼을 누른다.

②     화면 우측에는 이미지뷰어 프래그먼트가 보여진다. 여기서 이미지뷰어 프래그먼트를 보이도록 하는 프래그먼트 트랜잭션이 addToBackStack 함수를 통해 저장된 것을 기억해두자.

③     텍스트 보기 버튼을 누른다.

④     화면 우측에는 텍스트뷰어 프래그먼트가 보여진다. 여기서 텍스트뷰어 프래그먼트를 보이도록 하는 프래그먼트 트랜잭션이 addToBackStack 함수를 통해 저장된 것을 기억해두자.

⑤     이제 이전키를 누르면 ④의 트랜잭션 내용이 역순으로 실행되고, 화면 우측에는 이전 이미지뷰어 프래그먼트가 보이게 된다.

⑥     계속 이전키를 누르면 ③의 트랜잭션 내용이 역순으로 실행되고, 화면 우측에는 이전 이미지뷰어 프래그먼트가 보이게 된다.

⑦     이제 백스택에는 더 이상 쌓인 프래그먼트 트랜잭션이 없다. 이 상태에서 이전키를 누르면 액티비티가 종료되고 홈으로 복귀된다.

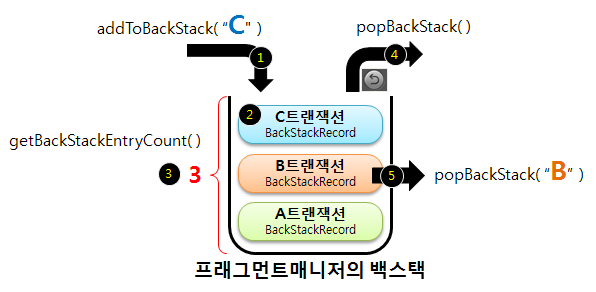
[참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D7%26fileid%3D1%26regdt%3D20140911212131&url=http%3A%2F%2Fcfile272.uf.daum.net%2Fattach%2F251CCA4454198D5D0ABEE7'))22-7. 프래그먼트 백스택.zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D7%26fileid%3D1%26regdt%3D20140911212131&url=http%3A%2F%2Fcfile272.uf.daum.net%2Fattach%2F251CCA4454198D5D0ABEE7'))

**백스택을 제어하는 함수**

[ REF \_Ref399422902 \r \h 그림 22-31]를 통해 프래그먼트매니저 백스택에 관련된 함수들을 한눈에 살펴보자.

그림 22-31 프래그먼트 백스택 함수들



①     addToBackStack 함수로 백스택에 트랜잭션을 추가한다. 그리고 해당 함수의 인자로 추가되는 트랜잭션 이름을 넘기는데, 이 것이 왜 필요한지는 ⑤에서 설명한다. 참고로 해당 함수는 프래그먼트 트랜잭션의 멤버함수이므로 beginTransaction과 commit 사이에서 사용되어야 한다.

②     추가된 트랜잭션은 프래그먼트매니저 내부 스택에 저장되는데, 해당 스택을 백스택이라 하고, 스택에 추가된 하나의 트랜잭션은 백스택레코드BackStackRecord라 한다.

③     백스택에 추가된 백스택레코드의 개수는 프래그먼트매니저의 getBackStackEntryCount 함수로 참조할 수 있다.

④     이전키를 누르면 백스택 탑에 있는 백스택레코드 하나를 꺼내서 해당 트랜잭션에 담긴 명령어를 반대로 수행한다. 따라서 화면은 이전으로 넘어가는 효과를 볼 수 있다. 사실 이전키를 누르면 프래그먼트매너저의 popBackStack 함수가 호출되는데 바로 이 함수가 백스택에서 백스택레코드 하나를 꺼내는 함수다. 따라서 직접 popBackStack을 호출하면 이전키를 누른 동일한 효과를 볼 수 있다.

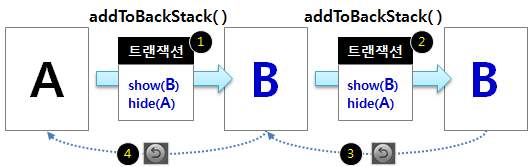
⑤     popBackStack 중에는 문자열을 인자를 가진 오버로딩 함수가 하나 더 있는데, 백스택에 존재하는 특정 백스텍레코드를 선택적으로 꺼내는 함수다. 해당 인자로는 꺼낼 백스택레코드명을 넘기면 된다. 참고로 백스택레코드명은 ①에서 백스택을 추가할 때 사용된 addToBackStack 함수의 인자로 설정할 수 있었다. 만일 특정 레코드를 꺼낼 필요가 없다면 굳이 백스택레코드명을 설정할 필요가 없다. 즉 ① addToBackStack 함수 인자를 null로 설정해도 된다는 의미다.

특정 백스택레코드를 꺼내는 함수는 최대한 신중히 사용해야 한다. 그 이유는 백스택에 레코드들이 ABC 순으로 쌓이고 그 역순 C->B->A로 꺼내는 것이 자연스럽기 때문이다. 만일 A->B->C 백스택 상태에서 특정 레코드 B를 먼저 꺼냈다고 생각해보자. B->C->A 순으로 화면이 복귀될 것이고 실행된 순과 다르기 때문에 어색해진다. 그래서 특별한 경우가 아니라면 대부분 거의 사용되지 않고, ④와 같이 백스택의 탑 레코드부터 하나씩 꺼내는 것이 일반적이다.

**백스택을 사용할 때 주의할 점**

그런데 백스택을 사용할 때 주의해야할 점이 있다. 앞서 [ REF \_Ref398754435 \r \h 그림 22-29]에서 백스택의 원리는 처리된 프래그먼트 트랜잭션의 명령을 반대로 수행하는 것이라 했다. 이점을 정확히 이해하지 못하면 앱을 개발하면서 간혹 백스택이 문제를 일으키는 경우가 있다. 어떤 문제를 말하는지 다음의 그림을 살펴보자.

그림 22-32 백스택을 사용했을 때 예상되는 화면의 변화



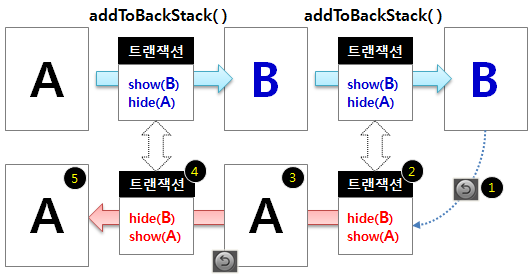
①     A프래그먼트가 화면에 보이는 상태에서 B프래그먼트 보여준다. 이를 위해 B프래그먼트를 보이고, A프래그먼트를 감춰야한다. 이 두 가지 명령은 하나의 트랜잭션에 추가하고 실행한다. 그리고 해당 트랜잭션에는 addToBackStack 함수를 사용하여 백스택에도 추가해둔다. 그 결과 B프래그먼트가 화면에 나타났다.

②     여기서 현재 화면에 보이는 B프래그먼트를 다시 보이도록 한다고 가정해보자. 물론 이러한 경우는 흔치 않겠지만 간혹 실수로 벌어지는 일이다. 어쨌든 ①과 같이 트랜잭션을 구성하고 실행한다. 그 결과 B프래그먼트가 계속 보인다.

③     이제 이전키를 누르면 이전 화면이 B프래그먼트가 보였기 때문에 계속 B가 보일 것이라 예상할 것이다.

④     그리고 다시 이전키를 누르면 A프래그먼트가 보일 것이다. 하지만 실제로 이런 경우를 확인하면 예상과 다르게 동작한다. 다시 ③부터 어떻게 동작하는지 [ REF \_Ref399513681 \r \h 그림 22-33]을 살펴보자.

그림 22-33 백스택을 사용했을 때 예상과 다른 화면의 변화 분석



①     백스택이 A->B->B 상태에서 이전키를 누른다.

②     여기서 제일 마지막에 추가된 백스택레코드의 명령을 살펴보면 B를 show하고 A를 hide했다. 따라서 이전으로 돌아가기 위해 실행된 명령을 반대로 수행할 것이다. 즉 B를 hide하고 A를 show한다. 그 결과 화면을 어떻게 될까?

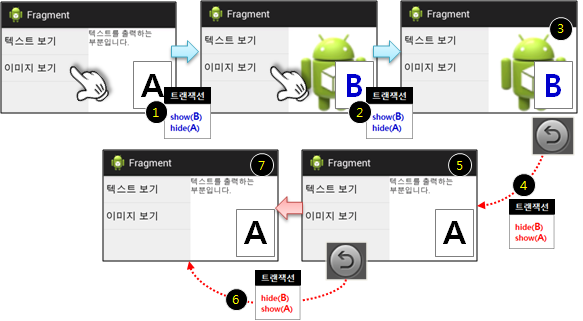
③     당연히 예상한 B프래그먼트는 보이지 않고 A프래그먼트가 보인다.

④     다시 이전키를 누르면 어떻게 될까? 이전 백스택레코드 명령을 살펴보면 B를 show하고 A를 hide이기 때문에 반대로 B는 hide하고 A는 Show하게 된다.

⑤     따라서 계속 A프래그먼트가 보일 것이다.

정말 [ REF \_Ref399513681 \r \h 그림 22-33]과 같이 동작하는지 예제 앱을 실행하여 결과를 확인해보자.

그림 22-34 프래그먼트 백스택 예제 실행



①     예제앱을 실행하면 액티비티 우측은 텍스트뷰어가 보인다. 이것을 A프래그먼트라 생각하자. 여기서 이미지 보기 버튼을 누른다.

②     이제 화면 우측은 이미지뷰어가 보인다. 이것을 B프래그먼트라 생각하자. 다시 이미지 보기 버튼을 누른다.

③     화면에는 이미 B가 보이고 있기 때문에 계속 B가 유지된다. 어쨌든 백스택은 A->B->B 상태다.

④     이전키를 누른다. 가장 최근 프래그먼트 트랜잭션 명령이 반대로 수행된다.

⑤     화면은 B프래그먼트인 이미지뷰어가 유지되지 않고 A가 보인다.

⑥     다시 이전키를 누르면 이전 프래그먼트 트랜잭션 명령이 반대로 수행된다.

⑦     결국 A프래그먼트인 텍스트뷰어가 계속 보이게 된다.

[ REF \_Ref399514696 \r \h 그림 22-34]의 결과에서도 알 수 있드시, 화면의 변화는 예상과 결과가 다르다. 따라서 개발자가 백스택은 처리된 프래그먼트 트랜잭션 명령어가 반대로 처리된다는 것을 모르면 문제의 원인을 분석하기 힘들다. 심지어 안드로이드의 버그라 판단할 수 있다.

어쨌는 백스택을 사용할 때, 동일한 트랜잭션을 연속해서 실행하는 경우는 피해야 한다. 따라서 예제가 올바르게 동작하도록 수정하자.

예제 22-32 연속으로 중복된 프래그먼트 트랜잭션이 수행되지 않도록 수정

|  |
| --- |
| **src/MainActivity.java**    **public** **class** MainActivity **extends** Activity {        @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);          ...  listMenuFragment.setOnListItemClickListener(  **new** ListMenuFragment.OnListItemClickListener()          {              @Override  **public** **void** onItemClick( **int** itemType )              {                  TextViewerFragment textViewerFragment = (TextViewerFragment)                      getFragmentManager().findFragmentByTag( "TEXT\_VIEWER" );                  ImageViewerFragment imageViewerFragment = (ImageViewerFragment)                      getFragmentManager().findFragmentByTag( "IMAGE\_VIEWER" );      **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_TEXT\_VIEWER* &&  **textViewerFragment.isVisible() == false** )                  {                      getFragmentManager()                      .beginTransaction()                          .hide( imageViewerFragment )                          .show( textViewerFragment )                          .addToBackStack( "TEXT\_VIEWER\_BACKSTACK" )                      .commit();                  }  **else** **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_IMAGE\_VIEWER* &&  **imageViewerFragment.isVisible() == false** )                  {                      getFragmentManager()                      .beginTransaction()                          .hide( textViewerFragment )                          .show( imageViewerFragment )                          .addToBackStack( "IMAGE\_VIEWER\_BACKSTACK" )                      .commit();                  }              }          });      }      ... |

[예제 22-32]의 수정된 내용을 살펴보자. 메뉴 리스트 아이템이 클릭되면, 해당 프래그먼트를 보여준다. 여기서 만일 보여주고자 하는 프래그먼트가 이미 보이고 있다면 처리하지 않도록 수정했다. 참고로 프래그먼트의 isVisible 함수는 해당 프래그먼트가 보이는 상태인지 확인할 수 있다. 예제를 실행하고 [ REF \_Ref399514696 \r \h 그림 22-34]의 과정을 재현해보면, 문제가 해결된 것을 확인할 수 있다.

[참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D7%26fileid%3D6%26regdt%3D20140911212131&url=http%3A%2F%2Fcfile289.uf.daum.net%2Fattach%2F2505BA4254255FD33AD515'))22-8. 중복된 프래그먼트 트랜잭션이 실행되지 않도록 수정.zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D7%26fileid%3D6%26regdt%3D20140911212131&url=http%3A%2F%2Fcfile289.uf.daum.net%2Fattach%2F2505BA4254255FD33AD515'))

마지막으로 프래그먼트의 상태를 확인하는 함수는 다음과 같다.

▶   **final** **public** **boolean** isAdded()

해당 프래그먼트가 추가된 상태인지 확인

▶   **final** **public** **boolean** isRemoving()

해당 프래그먼트가 제거된 상태인지 확인

▶   **final** **public** **boolean** isDetached()

해당 프래그먼트의 레이아웃이 떼어진 상태인지 확인

▶   **final** **public** **boolean** isVisible()

해당 프래그먼트가 보이는 상태인지 확인

▶   **final** **public** **boolean** isHidden()

해당 프래그먼트가 숨겨진 상태인지 확인

**백스택의 변화를 감지하는 리스너**

addToBackStack, popBackStack 등의 함수로 트랜잭션을 추가하거나 꺼내면 백스택의 변화가 발생된다. 이때 프래그먼트매니저는 그 변화를 감지하고 필요한 처리를 할 수 있도록 리스너를 제공한다. 직접 예제를 통해 기능을 살펴보자.

예제 22-33 프래그먼트 백스택 변화 감지 리스너

|  |
| --- |
| **src/MainActivity.java**    ① 백스택 변화를 감지하는 리스너 인터페이스를 상속받는다.  **public** **class** MainActivity **extends** Activity  **implements OnBackStackChangedListener**      {    ② 백스택 변화를 감지하는 리스너의 핸들러 함수를 구현한다.  @Override  **public void onBackStackChanged()**      {          Toast.*makeText*( getApplicationContext(),                          "Changed BackStack \n" +                          "BackStack Entry Count : " +                            getFragmentManager().**getBackStackEntryCount()**,                          Toast.*LENGTH\_LONG* ).show();      }          @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);          ...            listMenuFragment.setOnListItemClickListener(  **new** ListMenuFragment.OnListItemClickListener()          {             ...          });    // ③ 백스택 변화를 감지하는 리스너를 등록한다.  **getFragmentManager().addOnBackStackChangedListener(this);**  }        @Override  **protected** **void** onDestroy()      {          // ④ 등록된 백스택 리스너를 해제한다.  **getFragmentManager().removeOnBackStackChangedListener(this);**  **super**.onDestroy();      }  } |

①     프래그먼트 백스택 변화 감지 리스너 인터페이스는 OnBackStackChangedListener다. 따라서 리스너를 사용하려면 해당 인터페이스를 구현하고 프래그먼트매니저에 설정하면 된다. 따라서 먼저 인터페이스 구현하기 위해 액티비티가 OnBackStackChangedListener를 상속 받는 방법을 사용하도록 한다.

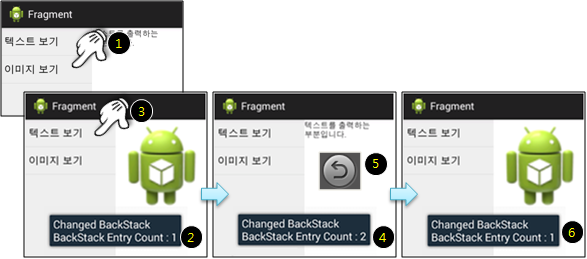
②     OnBackStackChangedListener의 핸들러 함수는 onBackStackChanged다. 액티비티 내에 해당 함수를 구현한다. 그리고 해당 함수가 호출될 때 화면으로 직접 확인해보기 위해 토스트 팝업을 구동한다. 토스트 팝업의 내용은 현재 백스택에 쌓인 백스택레코드 개수를 출력한다.

③     리스너를 등록하는 함수는 addOnBackStackChangedListener다. 만일 중복해서 여러 번 리스너를 등록하면 리스너의 핸들러 함수가 등록한 수만큼 호출될 수 있으므로 주의하자.

④     등록된 리스너를 해제하는 함수는 removeOnBackStackChangedListener다.

예제를 실행하여 결과를 확인해보자.

그림 22-35 프래그먼트 백스택 변화 감지 리스너 예제 실행



①     예제가 실행되면 이미지 보기 버튼을 누른다.

②     화면 우측에 이미지뷰어 프래그먼트가 보이고, 백스택에 프래그먼트 트랜잭션이 추가되었다. 따라서 백스택 변화가 발생되어 토스트 팝업이 나타났다. 백스택에는 백스택레코드 하나가 추가되었고 토스트 팝업 내용에 그 개수가 출력되었다.

③     텍스트 보기 버튼을 누른다.

④     화면 우측에 텍스트뷰어 프래그먼트가 보이고, 백스택에 프래그먼트 트랜잭션이 추가되었다. 따라서 백스택 변화가 발생되어 토스트 팝업이 나타났다. 백스택에는 백스택레코드가 총 두 개가 추가되었고 토스트 팝업 내용에 그 개수가 출력되었다.

⑤     이전키를 누른다.

⑥     백스택에서 백스택레코드를 꺼내 이전 화면으로 넘어갔다. 따라서 백스택 변화가 발생되어 토스트 팝업이 나타났다. 백스택에서 백스택레코드 하나가 제거되었고 토스트 팝업 내용에 남은 개수가 출력되었다.

[참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D7%26fileid%3D5%26regdt%3D20140911212131&url=http%3A%2F%2Fcfile268.uf.daum.net%2Fattach%2F241CE94154255FB82CABFE'))22-9. 콜스택 변화 감지 리스너.zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D7%26fileid%3D5%26regdt%3D20140911212131&url=http%3A%2F%2Fcfile268.uf.daum.net%2Fattach%2F241CE94154255FB82CABFE'))

다음 강좌가 이어집니다.