|  |  |
| --- | --- |
| |  | | --- | | **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. ^^; | |

**22.3.7 프래그먼트 복원**

안드로이드는 단말의 환경이 변경되거나 시스템의 메모리 부족으로 LMK가 동작하면 앱이 강제로 종료거나 재 실행될

수 있다. 이런 상황은 11장 액티비티 생명주기 11.2.4절에서 상세히 설명되었다. 어쨌든 문제는 앱이 강제 종료되고

재실행되면 종료전의 프래그먼트 정보가 모두 사라져 버린다. 따라서 액티비티와 같이 프래그먼트도 재실행될 때

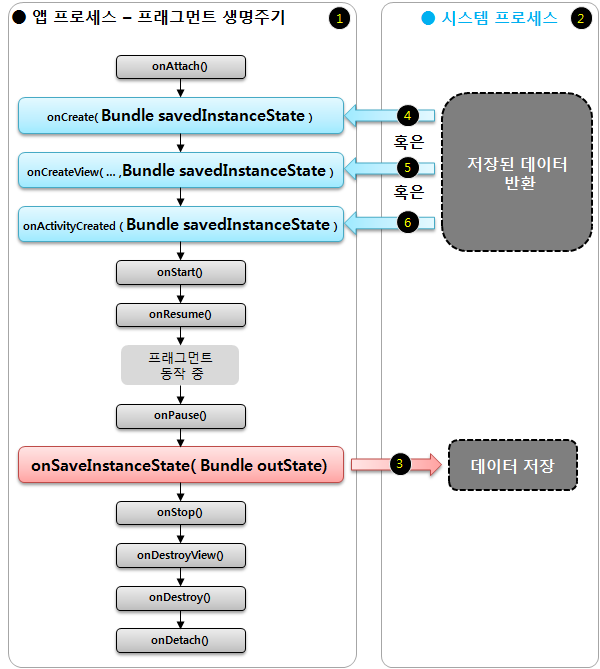
복원을 고려해야 한다.

프래그먼트의 복원은 액티비티의 복원 방법과 크게 다르지 않다. 즉 액티비티 복원의 경우 액티비티 생명주기 함수를

이용하는데 프래그먼트 역시 프래그먼트 생명주기 함수를 이용한다는 의미다. 따라서 프래그먼트 복원 방법을

이해하기 위해 프래그먼트 생명주기를 다시 한번 살펴보자.

그림 22-40 프래그먼트 복원과 관련된 생명주기 함수



①     만일 단말기의 환경 변화나 메모리 부족이 발생되면 앱 프로세스는 강제 종료될 수 있다. 프로세스가

      종료된다는 것은 해당 앱에서 사용된 모든 데이터가 사라짐을 의미한다. 따라서 앱은 이런 상황을 대비해서

      사용중인 데이터를 특정 시점에 저장하고 앱이 재실행될 때 복원해주어야 한다. 그렇다면 사용중인 데이터를

      어디에 저장해야 할까? 물론 파일로 저장해두는 방법도 있겠지만 더 간단한 방법이 있다. 바로 다른

      프로세스의 메모리에 임시 보관하는 방법이다.

②     안드로이드 시스템은 강제 종료되는 앱의 복원을 위해 해당 앱의 데이터를 임시 보관하고 특정 시점에 보관된

       데이터를 돌려준다. 이런 기능은 액티비티의 생명주기 함수와 프래그먼트 생명주기 함수를 통해 제공한다.

       액티비티의 복원은 11장에서 설명되었기 때문에 프래그먼트 생명주기를 이용한 복원 방법만 살펴본다.

③     프래그먼트 생명주기 함수중 [안내]태그제한으로등록되지않습니다-onPause 다음에는 늘 onSaveInstanceState가 호출된다. 바로 이 함수에서

       프래그먼트가 사용하고 있는 데이터를 잠시 시스템 프로세스로 저장해둘 수 있다. 간단히

       onSaveInstanceState 함수의 매개변수로 전달되는 번들에 저장하고 싶은 데이터를 담아두기만 하면

       내부적으로 시스템 프로세스로 번들이 전달된다.

④     시스템 프로세스에 저장된 번들은 강제 종료된 앱이 재실행될 때 프래그먼트 생명주기 함수인 onCreate

       인자로 돌려준다. 따라서 전달된 번들을 이용하여 복원하면 된다.

⑤     onCreate에서 전달된 번들은 onCreateView 생명주기 함수에서도 전달되고

⑥     onActivityCreated 생명주기 함수에서도 전달된다. 따라서 개발자가 원하는 시점에 복원과정을 수행하면

       된다.

[그림 22-40]의 복원 과정에서 ③의 onSaveInstanceState 함수가 호출되는 것을 예제로 직접 확인해보자. 텍스트뷰어

프래그먼트를 다음과 같이 수정한다.

예제 22-39 텍스트뷰어 프래그먼트에 복원 관련 생명주기 함수 추가

|  |
| --- |
| **src/TextViewerFragment.java**    **public** **class** TextViewerFragment **extends** Fragment {    ...        @Override  **public** **void** onCreate( **Bundle savedInstanceState** ) {... }        @Override  **public** View onCreateView( LayoutInflater inflater,                                   ViewGroup container,  **Bundle savedInstanceState** ) {... }        @Override  **public** **void** onActivityCreated( **Bundle savedInstanceState** ) {... }        @Override  **public void onSaveInstanceState( Bundle outState )**      {          Log.*e*("superdroid", "Fragment onSaveInstanceState()");    **super**.onSaveInstanceState( outState );      }    ...  } |

기존 onCreate, onCreateView, onActivityCreated 생명주기 함수의 매개변수를 보면, 모두 번들이 전달되는 것을

볼 수 있다. 이 번들은 시스템 프로세스를 통해 전달되며 복원에 활용된다. 그렇다면 이 값을 시스템 프로세스에

저장하는 onSaveInstanceState 함수를 살펴보자. 해당 함수의 매개변수로 번들이 전달된다. 바로 이 함수에서

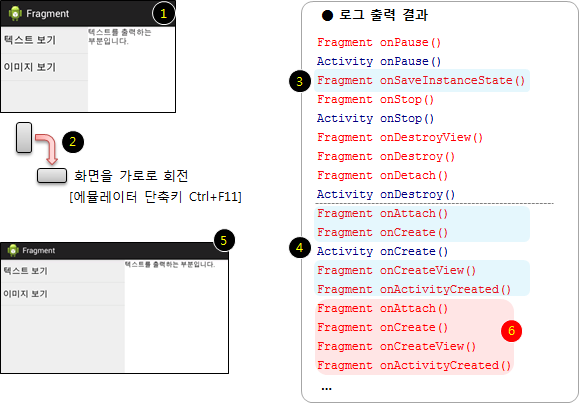
전달받은 번들에 복원시 필요한 데이터를 추가하면 된다. onSaveInstanceState 함수가 호출되는 것을 확인하기

위해 로그를 남기고 실행하여 결과를 확인해보자.

[참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D9%26fileid%3D9%26regdt%3D20141006162821&url=http%3A%2F%2Fcfile295.uf.daum.net%2Fattach%2F2268B443543CC8600F36B0')) 22-14. 프래그먼트 복원 관련 생명주기 함수.zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D9%26fileid%3D9%26regdt%3D20141006162821&url=http%3A%2F%2Fcfile295.uf.daum.net%2Fattach%2F2268B443543CC8600F36B0'))

그림 22-41 복원 관련 생명주기 함수



①     최초 화면 우측에는 텍스트뷰어 프래그먼트가 보인다.

②     다음은 프래그먼트 복원을 위한 onSaveInstanceState 함수가 호출되는지 확인이 필요하다.

       이를 위해 단말의 방향을 세로에서 가로로 회전시킨다. 이는 단말 환경에 변화를 주는 방법인데,

       액티비티는 강제로 종료하고 재실행되므로 복원과정을 확인할 수 있다.

③     로그를 살펴보면, 텍스트뷰어 프래그먼트 생명주기는 [안내]태그제한으로등록되지않습니다-onPause 함수가 호출된후 이어

       onSaveInstanceState가 호출된다. 따라서 이 함수에서 프래그먼트가 사용하는 데이터를 저장하면 된다.

       다음은 프래그먼트 종료 생명주기 함수들이 순서대로 호출된다.

④     이후 자동으로 다시 액티비티가 재실행되고 텍스트뷰어 프래그먼트가 생성 및 실행된다. 로그를 살펴보면,

       onAttach, onCreate, onCreateView, onActivityCreated 등의 생명주기 함수가 순서대로 실행되는 것을 볼

       수 있다.

⑤     마지막으로 화면은 가로 방향에 맞춰 다시 그려졌다.

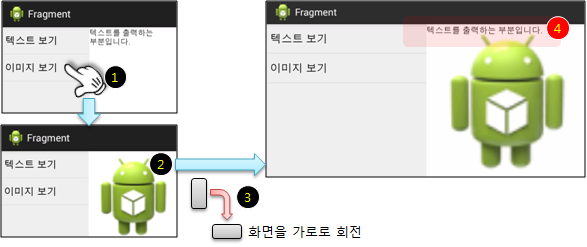
⑥     그런데 로그에 이상한 내용이 출력되었다. 텍스트뷰어 프래그먼트가 생성 및 실행되는 과정의 생명주기

       함수들이 한번더 출력된 것이다. 즉 로그의 내용으로 보면 텍스트뷰어 프래그먼트가 두번 생성 및

       실행되었다고 볼 수 있다.

       정말 그러한지 예제를 통해 직접 화면으로 확인해보자. [그림 22-42]의 과정으로 예제를 다시 실행한다.

그림 22-42 상태를 유지하는 프래그먼트매니저



①     예제앱이 실행되면 이미지 보기 버튼을 누른다.

②   화면 우측에 이미지뷰어 프래그먼트가 보인다.

③   화면을 세로에서 가로로 회전한다.

④   화면 회전 후 이미지뷰어 프래그먼트 위로 텍스트뷰어 프래그먼트까지 보인다. 그 이유는 예제 앱에서 화면

     회전 후 액티비티가 재시작될 때 onCreate 생명주기 함수에서 텍스트뷰어 프래그먼트를 추가하고 있기

     때문이다. 시스템은 단말의 환경이 변하면 액티비티를 재실행하여 상태를 유지한다. 이와 같은 이유로

     액티비티 내 실행중인 프래그먼트 역시 상태를 유지하기 위해 실행중이었던 프래그먼트를

     프래그먼트매니저에 의해 재실행한다. 따라서 당연히 재실행될 때 프래그먼트를 추가하면 문제가 된다.

[그림 22-42]와 같은 문제로 예제의 액티비티는 다음과 같이 수정되어야 한다.

예제 22-40 프래그먼트매니저의 상태 유지 고려

|  |
| --- |
| **src/ MainActivity.java**    **public** **class** MainActivity **extends** Activity **implements** OnBackStackChangedListener {      ...        @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);          ...            // savedInstanceState 번들이 널이라는 것은 해당 앱이 강제 종료 후          // 재시작된 것이 아니라는 의미다. 따라서 이 경우 화면 좌측의 텍스트뷰어          // 프래그먼트를 추가해준다.          // 참고로 만일 savedInstanceState 번들이 널이 아니라면 강제 종료 후          // 재시작하는 과정이므로 텍스트뷰어 프래그먼트를 추가하면 안된다.          // 그 이유는 프래그먼트매니저가 자동으로 해당 프래그먼트를 생성 후 추가해주기          // 때문이다.  **if( savedInstanceState == null )**          {              getFragmentManager()              .beginTransaction()                  .add( R.id.*viewer\_fragment\_container*,                        mTextViewerFragment, "TEXT\_VIEWER")              .commit();          }  ... |

수정 내용을 살펴보자. 액티비티의 onCreate 생명주기 함수가 호출될 때 매개변수로 전달되는 번들이 널이라면

최초 실행된 액티비티고 널이 아니면 재실행된 액티비티를 의미한다.

따라서 번들 객체 savedInstanceState가 널인 경우만 최초 화면 우측에 보여져야할 텍스트뷰어 프래그먼트를

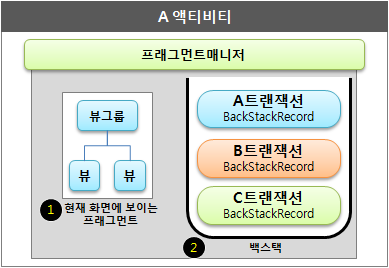
추가한다. 이제 실행해 결과를 확인해보면 [그림 22-42]의 문제는 재현되지 않는다.

[참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D9%26fileid%3D10%26regdt%3D20141006162821&url=http%3A%2F%2Fcfile252.uf.daum.net%2Fattach%2F2374CF3E543CC88036F17E')) 22-15. 프래그먼트매니저의 상태 유지 고려.zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D9%26fileid%3D10%26regdt%3D20141006162821&url=http%3A%2F%2Fcfile252.uf.daum.net%2Fattach%2F2374CF3E543CC88036F17E'))

그렇다면 액티비티가 재실행되더라도 프래그먼트매니저가 상태를 유지하는 것은 무엇인지 살펴보자.

그림 22-43 프래그먼트매니저의 상태 유지



①     [예제 22-40]의 결과에서 보았듯 현재 화면에 보이는 프래그먼트는 액티비티가 재실행하더라도

      프래그먼트매니저에 의해 다시 생성되고 화면에 보여지게 된다.

②    프래그먼트매니저의 백스택 역시 액티비티가 재실행되더라도 상태를 유지한다.

      따라서 재실행된 후 이전 키를 누르면 백스택에 존재했던 트랜잭션이 실행될 수 있다.

그런데 프래그먼트매니저가 상태를 유지한다면 프래그먼트 객체 자체를 복원할 필요가 없지 않는가!

이는 많은 개발자들이 오해하기 쉬운 부분이다. 좀더 쉽게 다시 설명하자면, 액티비티가 강제 종료되면

프래그먼트 객체는 모두 소멸된다.

단지 종료전 화면에 어떤 프래그먼트가 있었고 백스택는 어떤 백스택레코드가 있었는지만 유지하는 것이다.

따라서 프래그먼트매니저는 액티비티 재실행시 그 상태정보를 이용하여 화면에 나타나야할 프래그먼트만 다시

생성하고 추가하는 것이다. 그리고 이전키를 누를 때도 상태 정보를 이용하여 이전으로 복귀될 프래그먼트를 다시

생성하고 추가한다. 어쨌든 액티비티가 재실행되면 프래그먼트매니저가 관리하던 프래그먼트들은 모두 재생성된다.

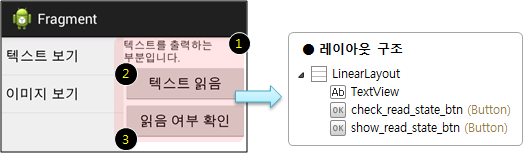
그리고 재생성되면 기존의 프래그먼트 객체의 데이터는 모두 초기화되니 복원해주어야 한다.

좀더 이해도를 높이기 위해 프래그먼트 객체가 복원해야할 데이터는 무엇이고 어떻게 복원하는지

예제를 통해 실습해보자.

먼저 프래그먼트 복원 예제에 사용될 텍스트뷰어 프래그먼트 레이아웃을 [그림 22-43]과 같이 구성한다.

그림 22-44 프래그먼트 객체 복원 예제의 텍스트뷰어 프래그먼트 레이아웃 구성



①     텍스트뷰어 프래그먼트 레이아웃에 버튼 두 개를 추가한다.

②     첫 번째 텍스트 읽음 버튼을 누르면 텍스트뷰어 프래그먼트 클래스 내부에 멤버변수로 읽음 여부를 저장해둘 것이다.

③     두 번째 읽음 여부 확인 버튼은 프래그먼트 클래스 내부에 저장된 텍스트 읽음 상태 멤버변수의 값을

       화면에 표시할 것이다. 여기서 멤버변수가 바로 프래그먼트 내부에서 사용되는 데이터에 해당하며,

       프래그먼트가 재실행될 때 복원 대상이다.

[그림 22-43] 텍스트뷰어 프래그먼트 레이아웃 구성을 적용하기 위해 다음과 같이 수정한다.

예제 22-41 텍스트뷰어 프래그먼트 레이아웃

|  |
| --- |
| **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"*>    <TextView  android:layout\_width=*"match\_parent"*  android:layout\_height=*"wrap\_content"*  android:text=*"텍스트를 출력하는 부분입니다."*/>    <Button      android:id=*"@+id/check\_read\_state\_btn"*    android:layout\_width=*"match\_parent"*          android:layout\_height=*"wrap\_content"*          android:text=*"텍스트 읽음"*/>    <Button          android:id=*"@+id/show\_read\_state\_btn"*          android:layout\_width=*"match\_parent"*          android:layout\_height=*"wrap\_content"*          android:text=*"읽음 여부 확인"*/>    </LinearLayout> |

다음은 텍스트뷰어 프래그먼트 소스를 수정한다. 그런데 수정할 내용에 복원 기능을 구현하지 않을 것이다.

이는 복원을 고려하지 않을 때 어떤 문제가 발생되는지 확인해보기 위해서다.

예제 22-42 텍스트뷰어 프래그먼트에 복원을 고려하지 않는 예제

|  |
| --- |
| **src/TextViewerFragment.java**    **public** **class** TextViewerFragment **extends** Fragment **implements** 0**nClickListener** {    // ① 텍스트 읽음 여부를 저장해두는 멤버변수 하나를 등록해둔다.      //    이 값은 프래그먼트 복원 테스트를 위해 추가되었다.  **Boolean  mIsRead = false;**      ...        @Override  **public** View onCreateView( LayoutInflater inflater,                                   ViewGroup container,                                   Bundle savedInstanceState )      {          Log.*e*("superdroid", "Fragment onCreateView()");            // ② 텍스트뷰어 레이아웃에 포함된 두 가지 버튼에 클릭 리스너를 설정          // ====================================================================          View view =  inflater.inflate( R.layout.*fragment\_text\_viewer*,                                         container,  **false**);    Button changeStringBtn =  (Button)view.findViewById( R.id.*check\_read\_state\_btn* );          Button showStringBtn = (Button)view.findViewById( R.id.*show\_read\_state\_btn*);          changeStringBtn.set0nClickListener( **this** );          showStringBtn.set0nClickListener( **this** );          // ====================================================================    **return** view;      }        @Override  **public** **void** 0nClick( View v )      {  **switch** (v.getId())          {  // ③ 텍스트 읽음 버튼을 누르면 프래그먼트 내부 멤버변수 mIsRead에 true              //    값을 저장해둔다.              // ================================================================  **case** R.id.*check\_read\_state\_btn*:              {  **mIsRead = true;**  **break**;              }              // ================================================================                // ④ 읽음 여부 확인 버튼을 누르면 프래그먼트 내부 멤버변수 mIsRead 값을              //    토스트로 보여준다.              // ================================================================  **case** R.id.*show\_read\_state\_btn*:              {                  Toast.*makeText*( getActivity(),                                  "읽음 여부 : " + mIsRead,                                  Toast.*LENGTH\_LONG* ).show();  **break**;              }              // ================================================================          }  }  ... |

①     텍스트 읽음 상태를 저장할 멤버 변수를 선언한다.

②     텍스트 읽음과 읽음 여부 확인 버튼에 클릭 리스너를 등록한다.

③     텍스트 읽음 버튼을 누르면 ①에서 선언한 mIsRead 멤버변수 값을 true로 변경한다.

       즉 이 값이 true라는 것은 텍스트를 읽었다는 의미다.

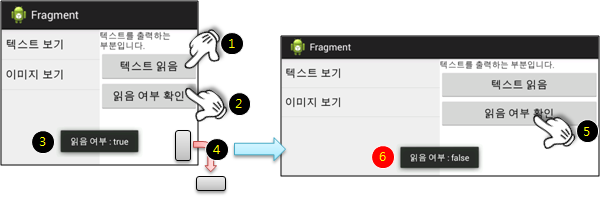
④     읽음 여부 확인 버튼을 누르면 mIsRead 멤버변수 값을 토스트 팝업으로 보여준다.

[참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D9%26fileid%3D11%26regdt%3D20141006162821&url=http%3A%2F%2Fcfile248.uf.daum.net%2Fattach%2F26328841543CC8A22080C8')) 22-16. 프래그먼트 복원 적용 전.zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D9%26fileid%3D11%26regdt%3D20141006162821&url=http%3A%2F%2Fcfile248.uf.daum.net%2Fattach%2F26328841543CC8A22080C8'))

예제를 실행하고 결과를 확인해보자.

그림 22-45 프래그먼트 객체 복원을 고려하기 전



①     예제 앱이 실행되면 우측 텍스트뷰어 프래그먼트의 텍스트 읽음 버튼을 누른다.

       그 결과 내부적으로 읽음 상태를 멤버변수로 저장할 것이다.

②     이어 읽음 여부 확인 버튼을 누른다. 그 결과 내부적으로 읽음 상태 멤버변수의 값을 토스트 팝업으로

       보여줄 것이다.

③     토스트 팝업이 출력되고 읽음 상태 멤버변수 값이 true로 출력된다.

④     이제 단말의 환경을 변경해보자. 화면을 가로로 회전한다.

⑤     다시 읽음 여부 확인 버튼을 누른다.

⑥     이상하게도 읽음 상태 멤버변수 값이 false로 출력되었다. 그 이유는 텍스트뷰어 프래그먼트 객체 자체가

       소멸되고 다시 생성되었기 때문이다. 따라서 이전 객체에서 변경한 멤버변수들을 모두 초기화된다.

이제 [그림 22-44]를 통해 프래그먼트에서 복원을 고려해야할 대상이 무엇인지 분명이 느꼈을 것이다.

그렇다! 복원의 대상은 프래그먼트 클래스 내부의 멤버 변수들이다. 멤버 변수들은 객체가 생성된 후 값이 변경될

수 있다. 따라서 객체가 소멸되고 다시 생성될 때 변경된 값들을 다시 복원해주어야 한다.

그렇다면 [그림 22-44]의 문제를 해결해보자. 텍스트뷰어 프래그먼트를 수정한다.

예제 22-43 텍스트뷰어 프래그먼트에 복원 고려한 예제

|  |
| --- |
| **src/TextViewerFragment.java**    **public** **class** TextViewerFragment **extends** Fragment **implements** 0nClickListener {    **...**        @Override  **public** **void** onSaveInstanceState( Bundle outState )      {          Log.*e*("superdroid", "Fragment onSaveInstanceState()");            // ① 복원을 위해 멤버 변수를 저장한다.  **outState.putBoolean( "mIsRead", mIsRead );**    **super**.onSaveInstanceState( outState );      }        @Override  **public** **void** onCreate( Bundle savedInstanceState )      {          Log.*e*("superdroid", "Fragment onCreate()");    // ② 저장해둔 멤버변수를 복원한다.          // ====================================================================  **if**( savedInstanceState != **null** )          {  **mIsRead = savedInstanceState.getBoolean( "mIsRead",**  **false );**          }          // ====================================================================    **super**.onCreate( savedInstanceState );      }    ... |

①     먼저 복원에 필요한 값을 저장하는 부분을 구현한다. onSaveInstanceState 함수를 재정의하고,

       해당 함수의 매개변수로 전달되는 번들에 복원에 필요한 멤버변수를 저장한다. 여기서 중요한 것은

       번들의 키값인데, 키 값은 멤버 변수명과 동일하게 맞추는 것이 좋다. 그 이유는 실수를 줄이기 위해서다.

②     다음은 복원하는 과정이다. onCreate 생명주기 함수에 추가한다. 해당 함수의 매개변수로 전달되는

       번들에는 ①에서 저장한 번들을 넘겨받는다.

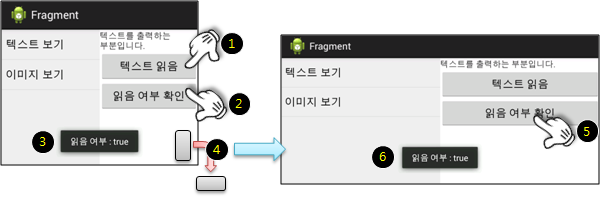
       따라서 번들에서 저장해둔 값을 꺼내 해당 멤버변수에 복원해주면 끝이다.

예제를 실행하고 결과를 확인해보자.

[참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D9%26fileid%3D12%26regdt%3D20141006162821&url=http%3A%2F%2Fcfile277.uf.daum.net%2Fattach%2F23496244543CC8B236809B')) 22-17. 프래그먼트 복원 적용 후.zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D9%26fileid%3D12%26regdt%3D20141006162821&url=http%3A%2F%2Fcfile277.uf.daum.net%2Fattach%2F23496244543CC8B236809B'))

그림 22-46 프래그먼트 객체 복원 고려 후



①부터 ⑥까지 과정을 따라해보자. [그림 22-44]과는 달리 ⑥에서 읽음 상태 값이 유지됨을 볼 수

있다. 즉 정상적으로 복원이 되었다.

마지막으로 단말기 상태가 변경된 상황이 아닌 시스템에 의해 앱이 강제 종료되고 재실행 될때에도 정상적으로

복원되는지 확인해보자.

그림 22-47 시스템에 의한 앱 강제 종료 및 재실행 시 프래그먼트 복원



①     텍스트 읽음 버튼을 누른다.

②     읽음 여부 확인 버튼을 누른다.

③     토스트 팝업을 통해 읽음 상태가 true인 것이 확인되었다.

④     이제부터 시스템에 의한 강제 종료 및 재실행을 재현해보도록 한다. 홈키를 누른다.

⑤     화면은 홈앱으로 전환되었다. 현재 예제 앱은 화면에 보이지 않기 때문에 시스템이 메모리가 부족할 때

       강제로 종료될 수 있는 상태다. 시스템이 강제로 종료하는 상황을 재현하긴 어렵기 때문에 DDMS를 통해

       강제로 종료하자.

⑥     이클립스의 DDMS를 활성화하고, 좌측 상단의 Devices에서 예제 앱 프로세스를 선택한다.

⑦     이어 ‘Stop Process’ 아이콘을 누른다. 예제 앱을 강제로 종료되었다. 이제 앱이 재실행되는 상황을

       재현해본다.

⑧     홈에서 예제 앱을 실행한다.

⑨     예제 앱이 실행되면 읽음 여부 확인 버튼을 누른다.

⑩     읽음 상태 값이 정상적으로 복원되었다. 참고로 ④부터 ⑧까지의 과정은 시스템에 의해 앱이 강제 종료되고

       재실행되는 상황을 재현할 수 있는 가장 효과적인 방법이므로 기억해두자.

|  |
| --- |
| **복원할 수 없는 객체**    [예제 22-43]에서 복원하는 대상으로 Boolean 멤버변수 하나만 테스트하였다. 하지만 앱을 개발하다 보면 클래스 내 멤버변수 개수는 늘어나기 마련이다. 그리고 복원하기 위해서 모든 멤버변수 값은onSaveInstanceState 재정의 함수에서 번들에 담아둬야 한다. 그런데 모든 타임의 멤버변수 값을 번들로 담을 수는 없다. 번들은 [그림 22-40]에서 보았듯 시스템 프로세스로 전달된다. 즉 프로세스 간 통신(IPC)이 일어나는 데이터다. 그리고 프로세스 간 전달되는 데이터는 반드시 직렬화되어야 한다. 물론 번들 자체는 직렬화 객체이므로 문제없다. 다만 번들의 담는 객체가 문제다. 만일 번들에 직렬화되지 않는 객체를 담는다면 다른 프로세스로 전달할 수 없는 데이터가 되고, 프래그먼트 복원도 할 수 없다. 따라서 복원이 필요한 모든 객체는 반드시 직렬화를 고려해야 한다.  사실 번들에는 직렬화되지 않은 객체는 담을 수 조차 없다. 그럼에도 저자가 굳이 설명한 것은 복원 객체의 경우 직렬화되어야 함을 상기시켜주기 위함이다. 앱을 개발하면서 직렬화 객체를 이해하는 것은 필수다. 만일 아직 직렬화에 대한 이해가 부족하다면 꼭 10장 10.1.4절부터 10.2.6절까지 다시 학습하기 바란다. |