Fragment 사용법

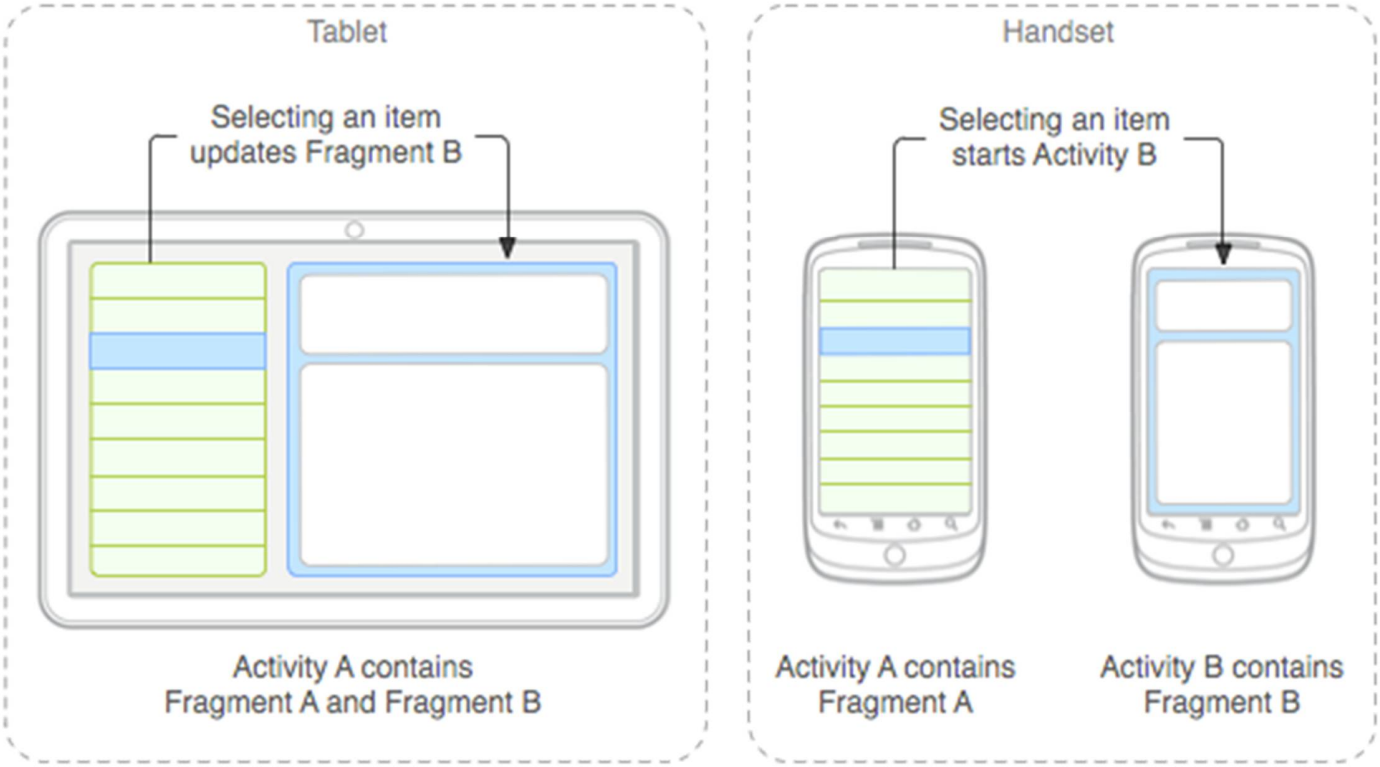
목차

1. Fragment 의 등장 배경

안드로이드 3.0(허니컴)이 공개되면서 프래그먼트(Fragment)가 추가 되었습니다. 프래그먼트는 태블릿과 같은 큰 화면을 가지는 단말에서 애플리케이션이 화면을 더 효율적으로 활용할 수 있도록 도와줍니다.

Fragment 는 한 화면을 여러 공간으로 쪼개어 사용하거나 액티비티 이동 없이 화면 내부구성을 다른 레이아웃으로 교체 시킬 때에는 Fragment 를 사용하고 있습니다.

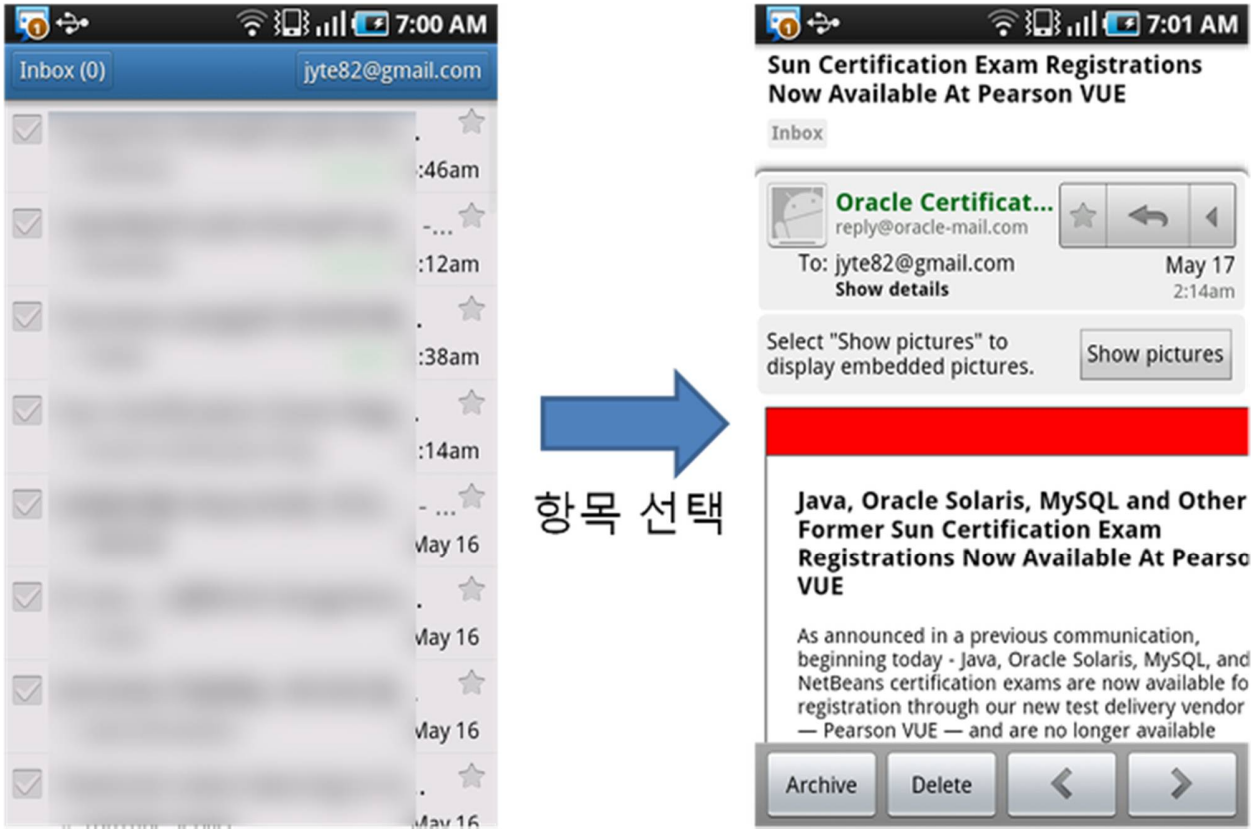
Fragment 는 View 의 특징과 Activity 의 특징을 모두 가지는 뷰 그룹이다.



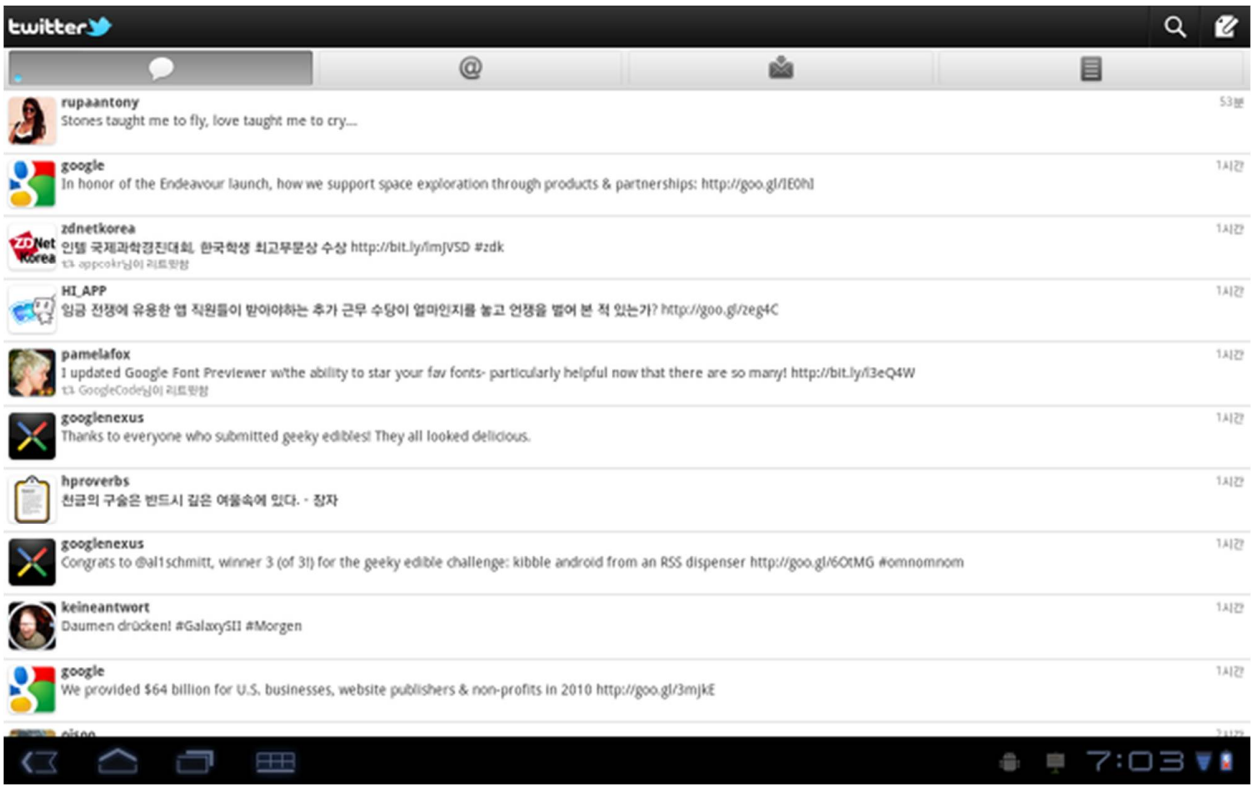
기존에는 애플리케이션 화면을 구성하는 큰 틀이 액티비티(Activity) 하나였고, 이 안을 여러 뷰로 구성하여 정보를 표시하고, 상호작용을 수행했습니다.

그런데, 뷰만을 사용해서 다양한 내용을 보여주기는 매우 어려웠습니다. 특히나 전체적인 UI 틀은 고정되어 있으면서 특정 부분만 변화하며 다른 내용을 표시하도록 하려면 매우 복잡한 구성이 필요했죠. 또한, 서로 다른 역할을 하는 코드들이 같은 곳에 있게 되어서 가독성도 떨어지고 유지보수에도 악영향을 미쳤습니다.

이 때문에 대부분 애플리케이션에서는 뷰 처리의 어려움도 피하고, 코드도 분리하게 위해 액티비티 전환을 사용했습니다. 아래의 GMail 앱을 통해 전형적인 액티비티 구성 방식을 볼 수 있습니다.

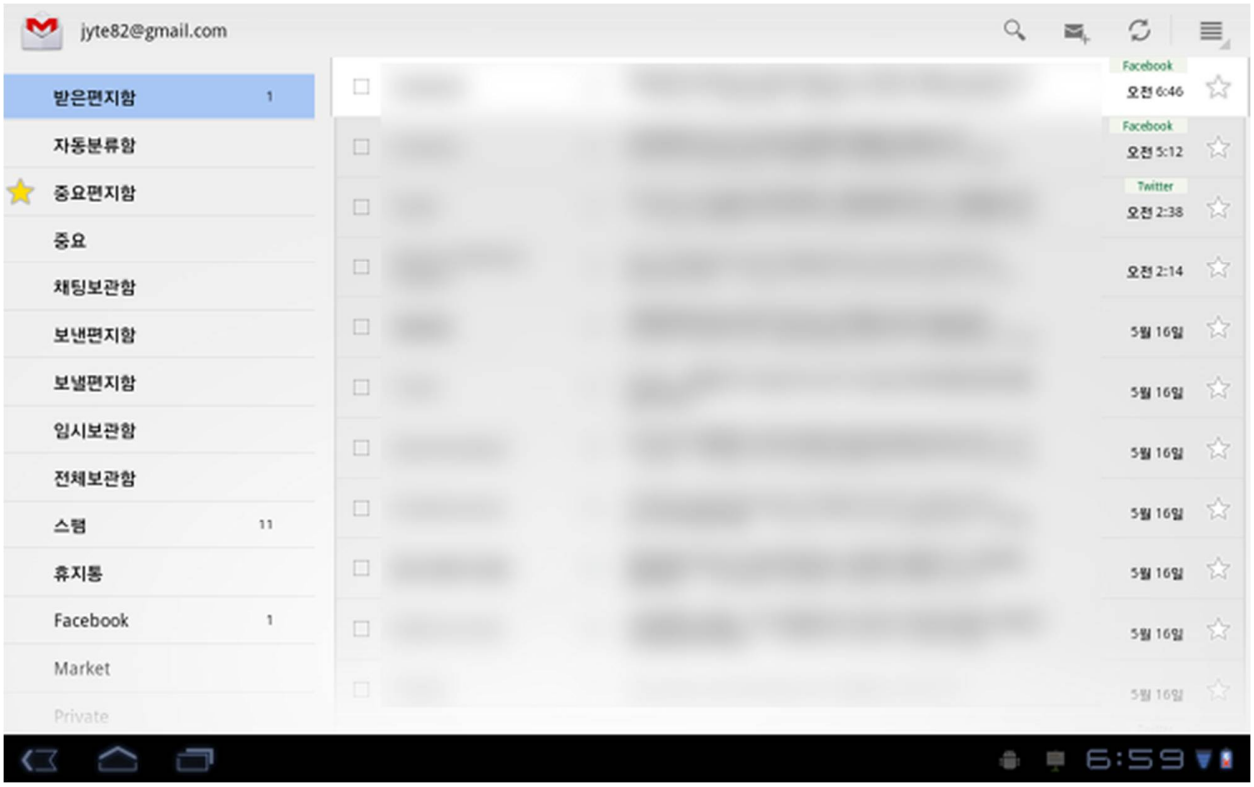


이 방식은 상당히 깔끔했습니다. 애플리케이션의 흐름 구성도 자연스럽게 잡히게 되었구요. 하지만, 화면이 큰 태블릿 단말들이 소개되면서 이 방식의 구성이 화면을 쓸데없이 많이 차지한다는 것을 느끼게 되었습니다. 호환성 문제는 없었지만, 큰 화면을 효율적으로 활용하지 못하는 것이죠.



트위터 공식앱을 실행한 모습. 태블릿에 최적화되지 않아 공간을 낭비하고 있습니다.

즉, 이제는 화면을 어떻게 하면 효율적으로 사용할 수 있는가? 를 고민할 차례가 되었습니다. 그럼, 화면을 효율적으로 활용한다는 것은 과연 어떤 것을 의미하는 것일까요? 여러 가지 방법이 있겠지만, 가장 대표적인 것은 한 화면에 가급적 다양한 정보를 표시하는 것입니다. 다음 안드로이드 3.0 버전의 GMail 앱을 보도록 하죠.



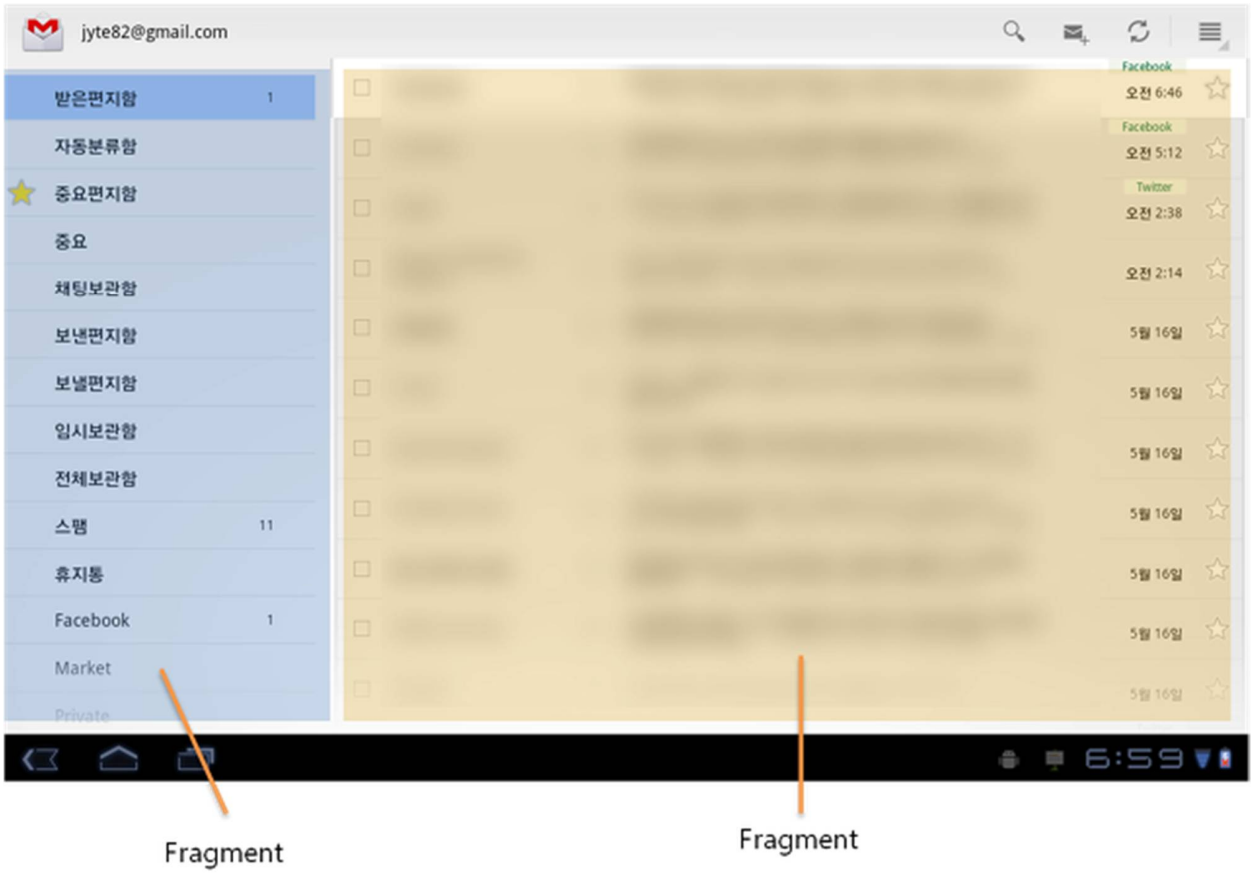
안드로이드 3.0의 G메일. 넓은 화면을 효율적으로 사용하고 있습니다.

기존 버전에서는 메일 리스트와 내용이 각각 전체 화면으로 표시되었던 것에 반해 안드로이드 3.0용에서는 메일 리스트와 내용이 동시에 표시되는 것을 확인할 수 있습니다. 넓은 화면을 효과적으로 활용하는 것이죠.

넓은 화면을 효율적으로 사용하기

그렇다면, 넓은 화면을 효율적으로 사용하려면 단순히 한 화면에 여러 요소들을 넣으면 되는 것일까요? 그렇지 않습니다. 한 화면에 여러 요소가 표시됨과 동시에 각 요소들을 조작하는 코드들은 각각 분리되어 있어야 합니다. 어떻게 보면 기존에 액티비티로 구현되던 요소들이 한 화면에 표시되는 것이죠.

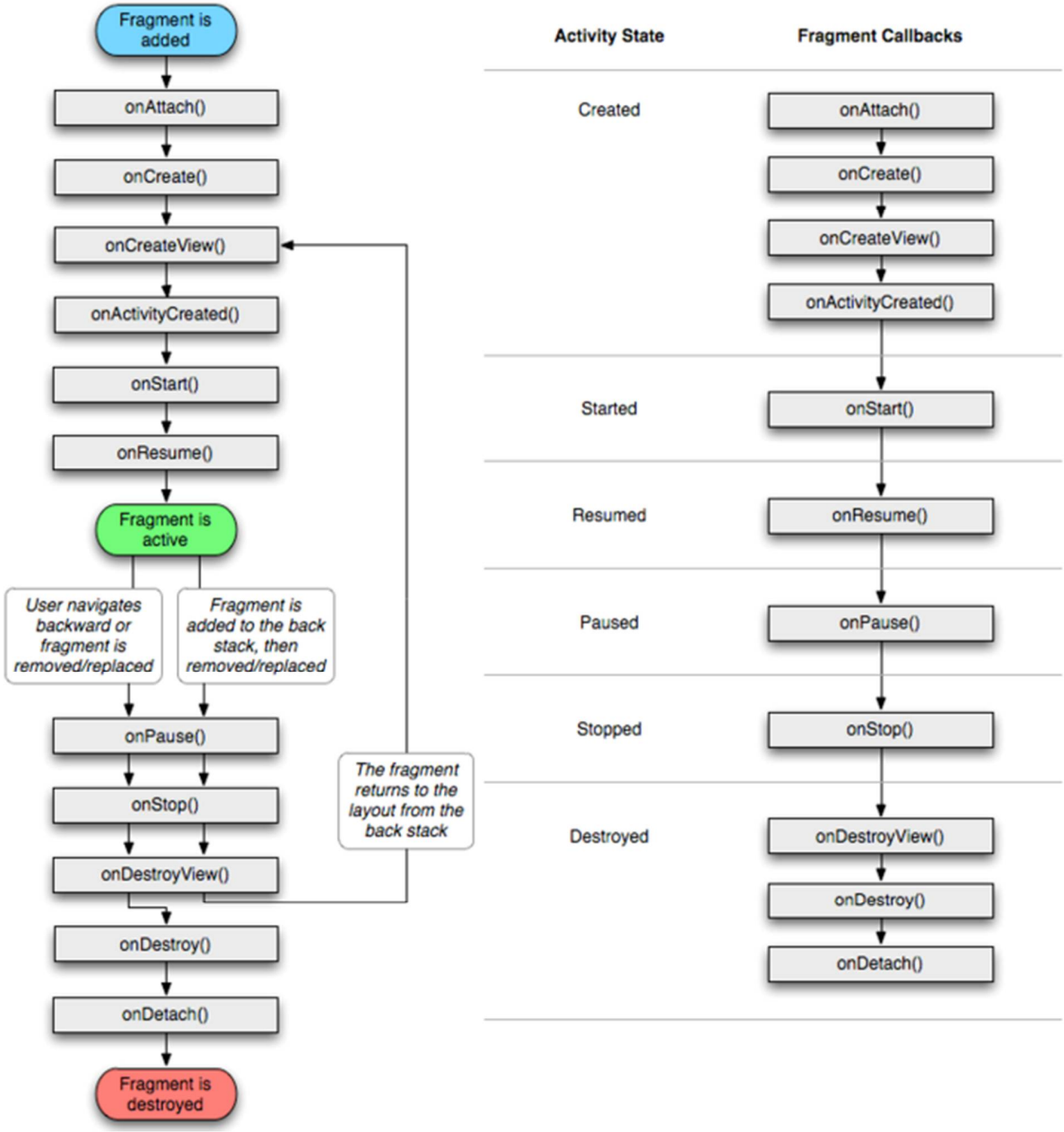
프래그먼트(Fragment)는 이러한 요구사항들을 잘 충족합니다. 액티비티처럼 관련된 코드들을 한곳에 묶을 수도 있고, 일반 뷰처럼 애플리케이션 레이아웃에 프래그먼트를 자유롭게 배치할 수도 있습니다. 다음 화면을 통해 프래그먼트의 배치를 확인할 수 있습니다.

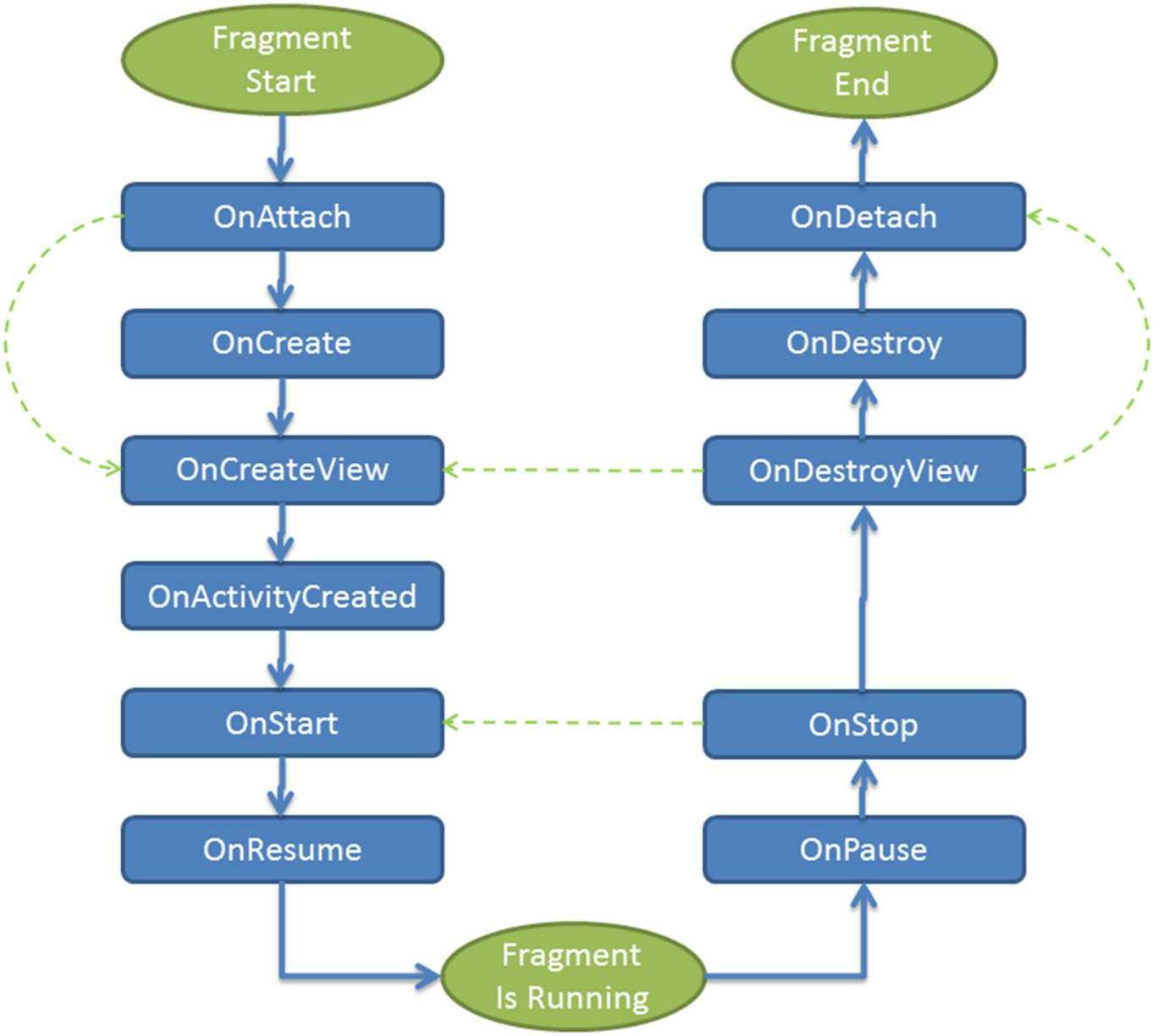


이것으로 프래그먼트의 등장 배경과 간략한 특징에 대해 알아보았습니다.

2. Fragment 의 생애주기

프래그먼트는 액티비티와 같이 프래그먼트의 상태가 계속해서 변하며, 상태가 변할 때마다 그에 해당하는 생애주기 메서드(콜백 메서드)가 호출됩니다. 프래그먼트의 생애주기 메서드 및 각 메서드의 호출 순서는 다음과 같습니다.




Fragment Start
OnAttach
OnCreate
OnCreateView
OnActivityCreated
OnStart
OnResume
Fragment Is Running
OnPause
OnStop
OnDestroyView
OnDestroy
OnDetach
Fragment End

3. 프래그먼트의 생애주기 메서드

프래그먼트 생애주기는 액티비티 생애주기와 매우 유사한 형태를 띄고 있으며, 뷰 생성과 관련된 몇몇 메서드가 더 추가되어 있습니다.

onCreate(Bundle)	액티비티의 onCreate() 콜백 메서드와 유사하게 프래그먼트가 최초로 생성될 때 호출됩니다.
onCreateView(LayoutInflater, ViewGroup, Bundle)	프래그먼트의 UI 를 구성하는 뷰(View)를 반환합니다. UI 를 가지지 않는 프래그먼트일 경우 null 을 반환할 수도 있습니다.
onAttach(Activity)	프래그먼트가 액티비티 레이아웃에 포함되는 순간 호출됩니다. 액티비티 레이아웃에 프래그먼트를 정적으로 배치했다면 액티비티가 시작될 때 같이 호출되며, 동적으로 레이아웃에 추가할 땐 프래그먼트를 레이아웃에 추가하는 순간 호출됩니다.
onActivityCreated(Bundle)	프래그먼트와 연결된 액티비티가 onCreate() 메서드의 작업을 완료했을 때 호출됨. 액티비티의 onCreate() 메서드가 끝났을 때 호출됩니다.
onStart()	프래그먼트가 화면에 표시될 때 호출됩니다. 하지만, 아직 사용자와 상호작용은 할 수 없는 상태입니다.
onResume()	프래그먼트가 사용자와 상호작용을 할 수 있게 되었을 때 호출됩니다. 즉, 프래그먼트가 완전히 화면에 표시되어 제 역할을 수행할 수 있게 된 상태입니다.
onPause()	액티비티의 onPause()와 유사하게 프래그먼트가 사용자와 상호작용을 할 수 없게 될 때 호출됩니다. 프래그먼트가 아직 화면에 표시되고 있는 상태이나, 다른 요소에 의해 프래그먼트가 가려져 상호작용을 하지 못하는 상태입니다.
onStop()	프래그먼트가 화면에서 보이지 않게 될 때 호출됩니다. 액티비티가 화면에서 보이지 않게 될 때 onStop() 메서드가 호출되는 것과 유사합니다.
onDestroyView()	프래그먼트가 화면에서 사라진 후, 뷰의 현재 상태가 저장된 후 호출됩니다. 여기에서 저장된 뷰의 상태는 액티비티와 유사하게 Bundle 형태로 저장되며, 저장된 뷰의 상태는 onCreate() 및 onCreateView()에서 다시 불러들일 수 있습니다.
onDestroy()	프래그먼트가 더 이상 사용되지 않을 때 호출됩니다.
onDetach()	프래그먼트가 액티비티 레이아웃에서 제거될 때 호출됩니다.

4. Fragrment 관련 클래스

Fragment

ListFragment

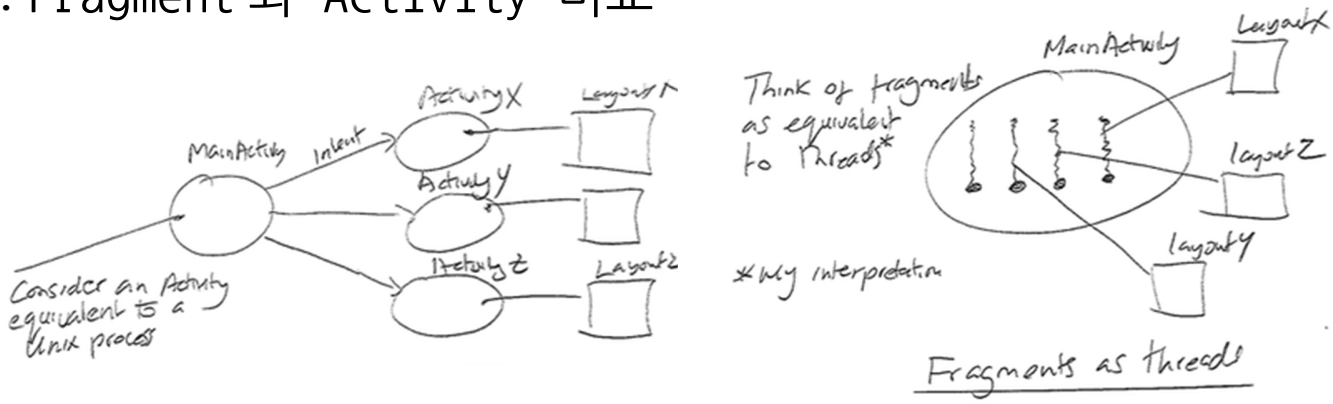
FragmentActivity

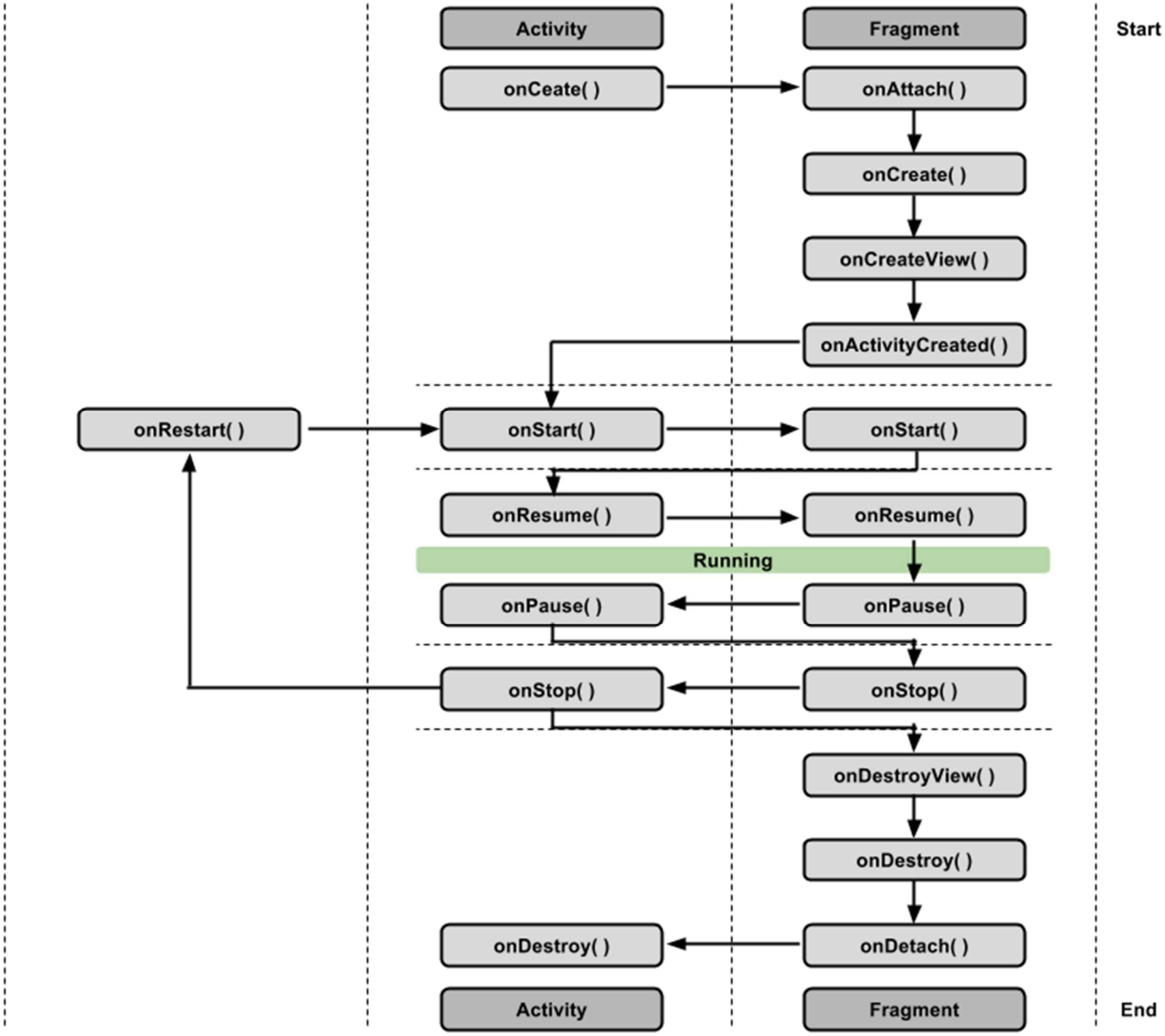
FragmentPagerAdapter

FragmentTransaction

FragmentManager

5. Fragment 와 Activity 비교





6. Activity 와 Fragment 간 데이터 주고 받기

Fragment 는 여러 Activity 에서 사용될 수 있으므로 Activity 에 독립적으로 구현되어야 합니다.
Fragment 는 getActivity() 메소드로 Attach 되어 있는 Activity 를 가져올 수 있습니다.

Activity 로 이벤트 콜백 메소드 만들기
Fragment 내에서 발생하는 이벤트를 Activity 와 공유하기 위해서는 Fragment 에서 이벤트 콜백 인터페이스를 정의하고 Activity 에서 그 인터페이스를 구현해야 합니다.

예를 들면
Activity 내에 두개의 Fragment 가 있고, 하나는 최신기사의 제목 목록을 보여주는 ListFragment 이고 다른 Fragment 는 선택된 기사의 상세내용을 보여주는 DetailFragment 입니다.
사용자가 ListFragment 에서 최신기사의 제목을 선택했을 때 선택한 선택된 기사의 상세 내용을 보여주기 위해서는 선택됬다는 이벤트를 DetailFragment 에서 알아채야 합니다.

이때 가장 쉽게 생각할 수 있는 방법은
ListFragment 에서 사용자가 선택 했을때 DetailFragment 를 바로 호출하는 방법입니다.

```
DetailFragment detailFrag =
(DetailFragment)getActivity().getFragmentManager().findFragmentById( ... );
detailFrag.viewArticle(...)
```

Fragment 에서는 Activity 를 가져올 수 있는 getActivity() 메소드가 존재하기 때문에 위처럼 작성하는 것도 가능합니다.
그럼 위 코드의 문제점은 뭘까요? ListFragment 가 기사를 보여주는 DetailFragment 와 연관되어 있기 때문에 ListFragment 는 재사용이 가능하도록 독립적으로 설계되지 않았습니다.

그럼 ListFragment 가 홀로 설수 있는 방법은 무엇을 까요?

1. ListFragment 내에 이벤트 인터페이스를 정의합니다.

```
public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
}
```

2. Activity 에서 Fragment 의 이벤트 인터페이스를 구현합니다.

```
... Activity implements ListFragment.OnArticleSelectedListener{
...
      public void onArticleSelected(Uri articleUri){
            // DetailFragment 를 가져와서 이벤트를 처리합니다.
            DetailFragment detailFrag =
(DetailFragment)getFragmentManager().findFragmentById( ... );
            detailFrag.viewArticle(...)
      }
}
```

3. ListFragment 에서 Activity 에 대한 참조를 얻습니다.

Fragment 는 onAttach lifecycle 콜백함수에서 Activity 에 대한 참조를 얻을 수 있습니다.

```
    OnArticleSelectedListener mListener;

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement
OnArticleSelectedListener");
        }
    }
```

4. ListFragment 의 이벤트 콜백 함수에서 Activity 에서 인터페이스를 구현한 이벤트 콜백 함수를 호출합니다.

```
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Send the event and Uri to the host activity
        mListener.onArticleSelected(noteUri);
    }
```

복잡하지만 Fragment 와 Activity 중간에 인터페이스를 두어 Fragment 를 독립적으로 구현할 수 있습니다.

7. Fragment 간 데이터 주고 받기

Activity 에서는 Intent.putExtra()와 Intent.getExtra()와 같은 method 를 이용하여 데이터를 주고 받았지만 Fragment 에서는 setArguments()와 getArguments()메소드를 이용하여 데이터를 주고 받을 수 있습니다.

setArguments() 메소드는 fragment 의 Bundle 을 저장합니다. getArguments() 메소드는 이 전달된 정보를 얻기 위해 Bundle 을 retrieve 하죠.

아래 코드는 fragment1 에서 fragment2 로 정보를 전달합니다. frag2_container 는 layout xml 에서 지정한 fragment container 의 ID 입니다.

```
Fragment2Activity fragment2 = new Fragment2Activity();
Bundle args = new Bundle();
String message = "Message from Fragment1";
   if(null==fragmentManager.findFragmentByTag(TAG2)){
       Fragment2Activity fragment2 = new Fragment2Activity();
       args.putString("msg", message);
       fragment2.setArguments(args);
       fragmentTransaction.replace(R.id.frag2_container,fragment2);
       String tag = null;
       fragmentTransaction.addToBackStack(tag);
       fragmentTransaction.commit();
}
```

#1 에 있는 Fragment2Activity 는 fragment2 의 자바 클래스를 말합니다. fragment2 라는 자바 클래스 인스턴스를 생성했습니다.

#2 에서는 args 라고 하는 Bundle 객체를 생성했습니다.

#3 에서는 message 라고 하는 String 객체를 생성했죠. 이 메세지는 fragment2 에 전달 될 겁니다.

#4 번에서는 TAG2 라는 fragment 가 있는지 여부를 체크합니다. TAG2 는 fragment2 를 말하는 거니까 fragment2 가 있는지 여부를 체크하는 겁니다.

#5 번은 Fragment2Activity 자바클래스의 인스턴스를 만들고 #6 번에서는 args 번들에 3 번에서 정의했던 메세지를 세팅합니다.

#7 에서는 fragment2 에 이 Bundle 을 세팅하구요.

#8 번과 #9 번에서는 이전의 fragment 를 navigating 하는 부분이구요.

fragment 도 activity 처럼 stack 을 사용합니다. back stack 에 FragmentTransaction 을 추가하려면 commit() 하기 전에 FragmentTransaction 의 addToBackStack() 메소드를 호출해야 합니다.

위 코드에서 보면 처음에는 xml 에 있는 fragment 가 화면에 보였다가 두번째 fragment 로 replace 됐습니다. 그리고 이전 fragment 는 BackStack 에 넣구요. 그래서 Back 버튼을 누르면 이전의 FragmentTransaction 이 돌아와서 이전의 fragment 가 화면에 보여질 겁니다.

Bundle 을 통해서 위에 setArguments()메세지를 통해 세팅된 정보를 어떻게 다음 fragment 에서 이 메세지를 받고 display 하는지에 대해서도 궁금할 겁니다. 그 답은 Bundle 을 사용해서 fragment 에서 메세지를 받는 것은 getArgument() 메소드를 사용한다 입니다.
아래에 샘플 코드가 있습니다.

```
String messageReceived = "";
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        Bundle bundle = getArguments();
        if(bundle != null) {
            messageReceived = bundle.getString("msg");
        }
}
```

8. 예제 1.

프래그먼트의 특징에 대해 간단히 알아보았으니, 우선 프래그먼트를 레이아웃에 추가하는 방법에 대해 알아보도록 하겠습니다. 프래그먼트를 레이아웃에 추가하는 방법은 여러 가지가 있지만, 그 중에서도 가장 간단한 방법인 XML 레이아웃을 사용하여 추가하는 방법을 알아보겠습니다.

[어플리케이션 정보]

액티비티

MainActivity (MainActivity.java)

레이아웃

activity_main.xml (Activity)

fragment_main.xml (Fragment)

우선, 프래그먼트에 표시할 레이아웃 파일을 작성합니다.

[activity_main.xml]

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <fragment
        android:id="@+id/fragment_one"
        class="com.androidhuman.example.SimpleFragment.Main$FragmentOne"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

다음과 같이 fragment_one.xml 레이아웃 파일을 추가한 후,TextView 하나를 추가합니다.

[fragment_main.xml]

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

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fragment One" />

</LinearLayout>
```

액티비티 코드에 프래그먼트 클래스를 추가한 후, 프래그먼트에 표시할 뷰를 지정해주기 위해 onCreateView() 다음과 같이 메서드를 오버라이드합니다. 프래그먼트에서 표시할 뷰를 반환하기 위해 onCreateVIew()의 인자 중 하나인 LayoutInflater를 사용합니다.

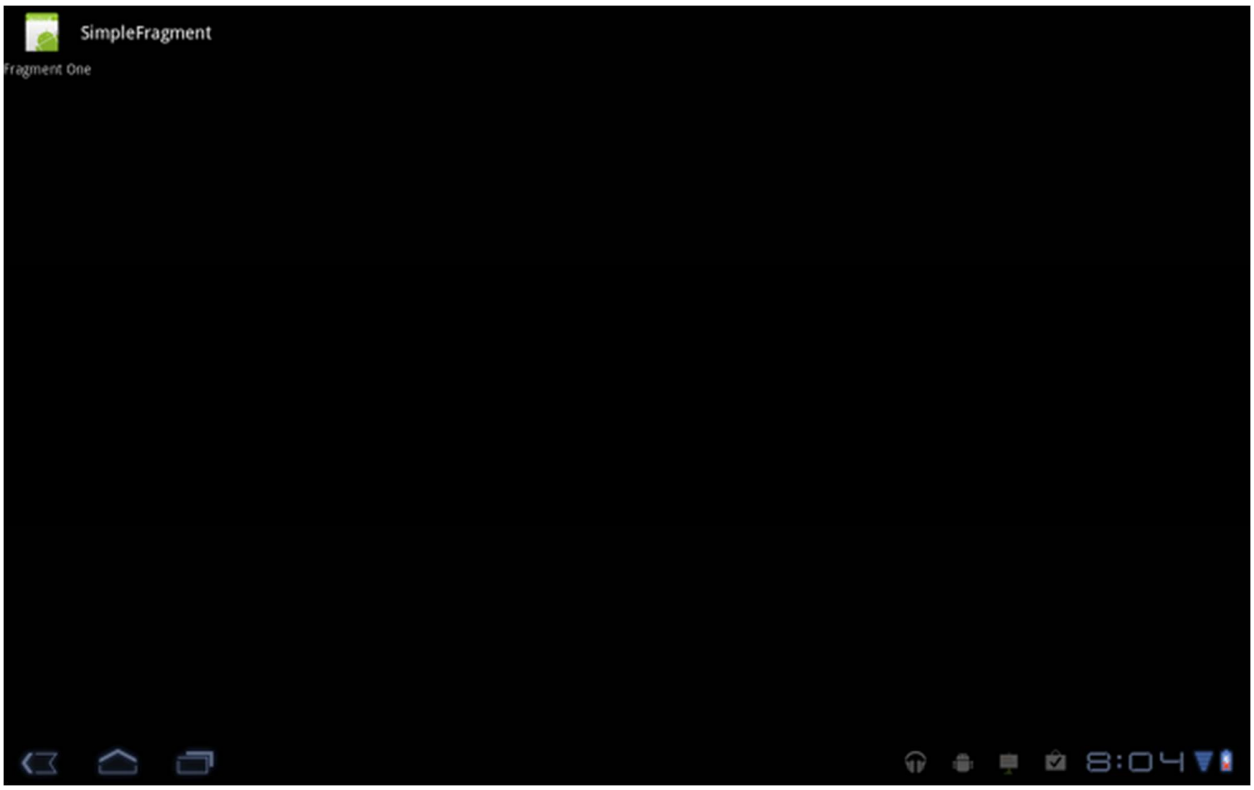
[MainActivity.java]

```
public class Main extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public static class FragmentOne extends Fragment{

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                Bundle savedInstanceState) {
            return inflater.inflate(R.layout.fragment_one, null);
        }

    }

}
```

다음, 액티비티의 레이아웃을 작성합니다. 프래그먼트를 XML 레이아웃 파일에서 선언할 때는 프래그먼트의 클래스(class)와 프래그먼트를 구분할 수 있는 id 혹은 tag 를 필히 지정해야 합니다. 여기에서는 프래그먼트를 구분하기 위해 id 를 지정해 주었습니다.

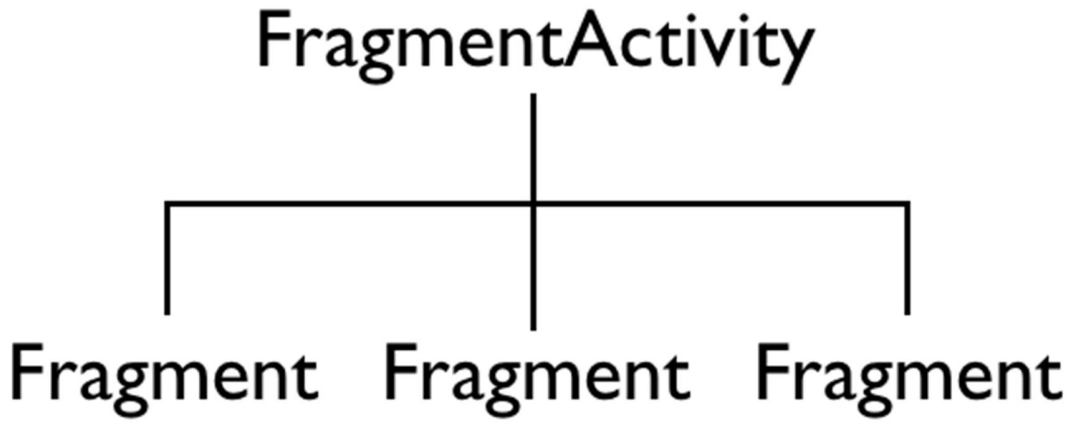
이 예제에서 프래그먼트 클래스가 Main 클래스의 내부 클래스로 선언되어 있기 때문에 Main$FragmentOne 과 같이 클래스 이름을 지정해 주었습니다.

이것으로 모든 구현이 끝났습니다. 생각보다 간단하지요? 예제를 실행하면 다음과 같이 프래그먼트가 표시되는 것을 확인할 수 있습니다.



9. 예제 2.

Fragment 를 사용하려면 그것을 사용하는 상위 activity 가 FragmentActivity 여야 합니다.



Fragment 생명주기는 activity 와 흡사하며 주의 할 점이 activity 에서 this 로 쓰던 부분을 getActivity()로 바꿔 써주셔야 합니다.

큰 구성은 FragmentActivity 를 상속받은 MainActivity.java 가 있고 Fragment 를 상속받은 OneFragment, TwoFragment, ThreeFragment.java 세가지 클래스를 만든다음 activity_main.xml 에 있는 ll_fragment 영역에 위에만든 세가지 fragment 를 삽입하는 방식입니다.

먼저 MainActivity 구성입니다.

위 그림과 같이 activity_main.xml 레이아웃 하나에 fragment 가 교체 될 영역과 그 영역을 컨트롤 할

메뉴 뷰 를 만들어 줍니다.

```
〈MainActivity〉
public class MainActivity extends FragmentActivity implements OnClickListener {

	final String TAG = "MainActivity";
```

```
int mCurrentFragmentIndex;
public final static int FRAGMENT_ONE = 0;
public final static int FRAGMENT_TWO = 1;
public final static int FRAGMENT_THREE = 2;

@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_main);

	Button bt_oneFragment = (Button) findViewById(R.id.bt_oneFragment);
	bt_oneFragment.setOnClickListener(this);
	Button bt_twoFragment = (Button) findViewById(R.id.bt_twoFragment);
	bt_twoFragment.setOnClickListener(this);
	Button bt_threeFragment = (Button) findViewById(R.id.bt_threeFragment);
	bt_threeFragment.setOnClickListener(this);

	mCurrentFragmentIndex = FRAGMENT_ONE;

	fragmentReplace(mCurrentFragmentIndex);
}

public void fragmentReplace(int reqNewFragmentIndex) {

	Fragment newFragment = null;

	Log.d(TAG, "fragmentReplace " + reqNewFragmentIndex);

	newFragment = getFragment(reqNewFragmentIndex);

	// replace fragment
	final FragmentTransaction transaction = getSupportFragmentManager()
			.beginTransaction();

	transaction.replace(R.id.ll_fragment, newFragment);

	// Commit the transaction
	transaction.commit();
```

```
}

private Fragment getFragment(int idx) {
	Fragment newFragment = null;

	switch (idx) {
	case FRAGMENT_ONE:
		newFragment = new OneFragment();
		break;
	case FRAGMENT_TWO:
		newFragment = new TwoFragment();
		break;
	case FRAGMENT_THREE:
		newFragment = new ThreeFragment();
		break;

	default:
		Log.d(TAG, "Unhandle case");
		break;
	}

	return newFragment;
}

@Override
public void onClick(View v) {

	switch (v.getId()) {

	case R.id.bt_oneFragment:
		mCurrentFragmentIndex = FRAGMENT_ONE;
		fragmentReplace(mCurrentFragmentIndex);
		break;
	case R.id.bt_twoFragment:
		mCurrentFragmentIndex = FRAGMENT_TWO;
		fragmentReplace(mCurrentFragmentIndex);
		break;
```

```
            case R.id.bt_threeFragment:
                  mCurrentFragmentIndex = FRAGMENT_THREE;
                  fragmentReplace(mCurrentFragmentIndex);
                  break;

            }

      }

}
```

전체적으로 보면 fragment 마다 int 으로된 구분자를 주고 메뉴 버튼이 클릭 될 때마다 해당 fragment 의 index 를 mCurrentFragmentIndex 라는 int 형 변수에 담아 fragment 교체 매소드를 태우는 식입니다.

onCreate 부터 보면 메뉴 버튼등록과 리스너를 달고 mCurrentFragmentIndex 에 첫 화면 fragment 인 FRAGMENT_ONE 를 넣어주고 fragmentReplace 매소드를 타게됩니다.

fragmentReplace 매소드에는 넘겨받은 fragment index 를 가지고 getFragment 매소드를 가서 실질적인 fragment 의 객체 생성을 하고 그 객체를 리턴해줍니다.

다시 fragmentReplace 매소드로 돌아와 리턴받은 fragment 로 ll_fragment 라는 레이아웃 영역에 교체를 하게 됩니다. 다른 버튼을 눌렀을 때도 이와 같은 방식으로 동작하게 됩니다.

Fragment 들은

```
public class OneFragment extends Fragment implements OnClickListener {

      @Override
      public View onCreateView(LayoutInflater inflater, ViewGroup container,
                  Bundle savedInstanceState) {

            View v = inflater.inflate(R.layout.onefragment, container, false);
```

```
            Button button = (Button) v.findViewById(R.id.bt_ok);
            button.setOnClickListener(this);

            return v;
      }

      @Override
      public void onClick(View v) {

            switch (v.getId()) {

            case R.id.bt_ok:
                  Toast.makeText(getActivity(), "OneFragment", Toast.LENGTH_SHORT)
                              .show();
                  break;

            }

      }

}
```

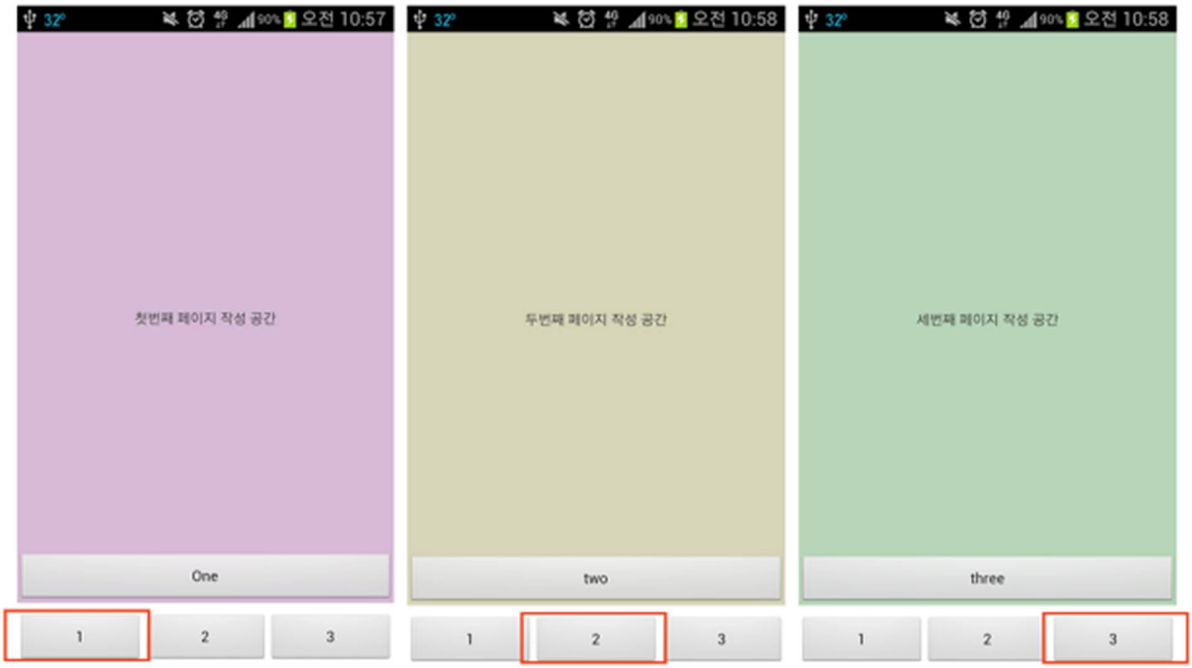
이처럼 Activity 와 비슷한 구조이고

마지막으로 activity_main.xml 구조입니다.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:background="#ffffffff"
        android:orientation="vertical" >

        <LinearLayout
            android:id="@+id/ll_fragment"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1" />

        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="10dp"
            android:layout_marginTop="10dp"
            android:background="#ffffffff"
            android:orientation="horizontal" >

            <Button
                android:id="@+id/bt_oneFragment"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="1" />

            <Button
                android:id="@+id/bt_twoFragment"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="2" />

            <Button
                android:id="@+id/bt_threeFragment"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="3" />
        </LinearLayout>
    </LinearLayout>

</RelativeLayout>
```

ll_fragment 라는 영역이 Fragment 들이 교체 될 영역입니다.



10. 예제 3. Fragment 와 ViewPage

11. Reference

http://androidhuman.com/469
http://androidhuman.com/470
http://muzesong.tistory.com/84
http://ismydream.tistory.com/135

https://android-developers.googleblog.com/2011/02/android-30-fragments-api.html
http://blog.daum.net/mailss/19
http://www.kmshack.kr/2013/02/fragment-파헤치기-1-fragment-개념/
http://www.kmshack.kr/2013/02/fragment-파헤치기-2-fragment-lifecycle 생명주기/