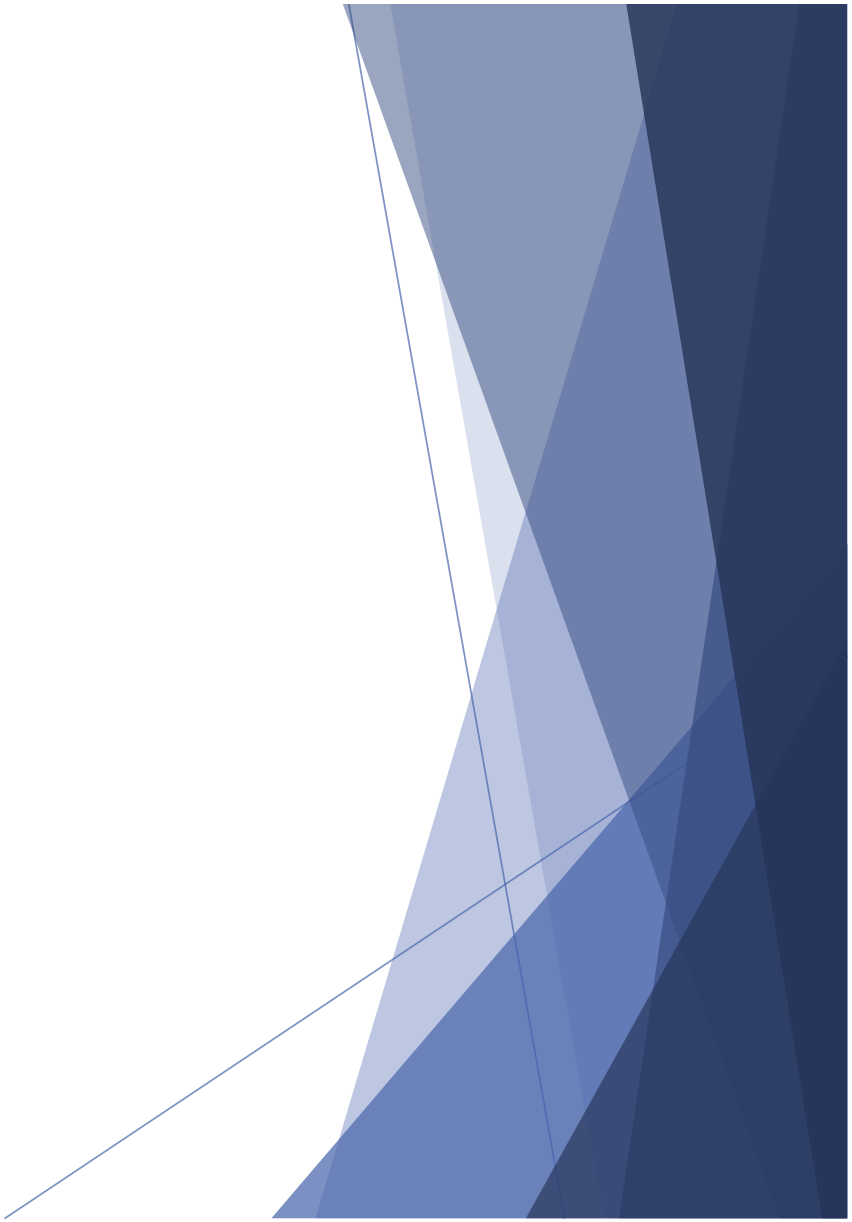
Android Programming

고급 UI 구현

Advanced UI

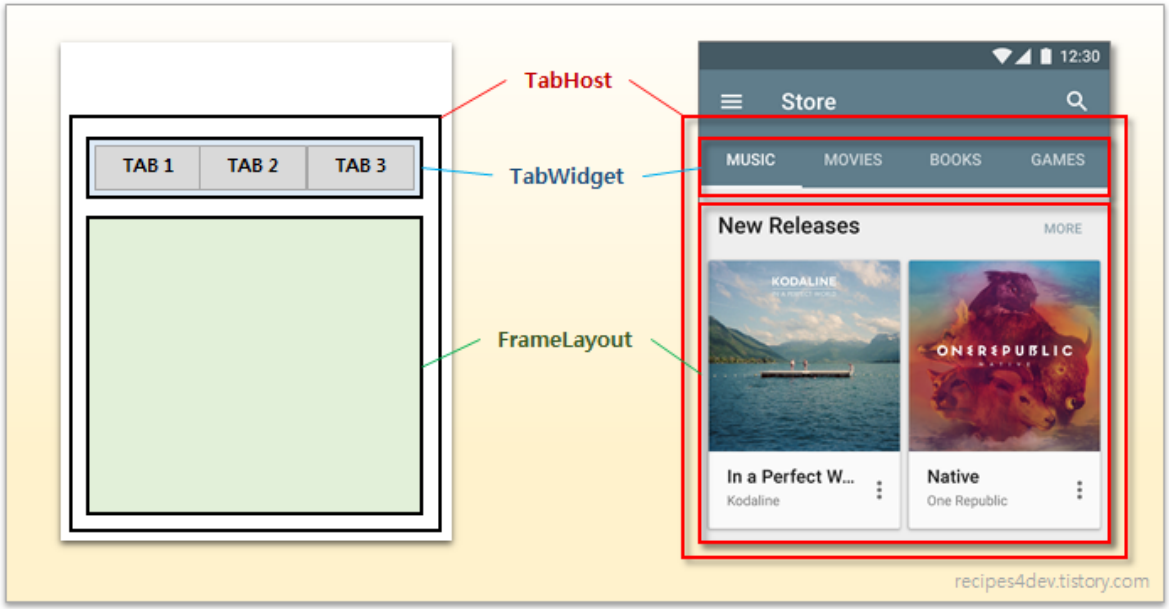
TabHost와 FrameLayout



Tab UI의 구현

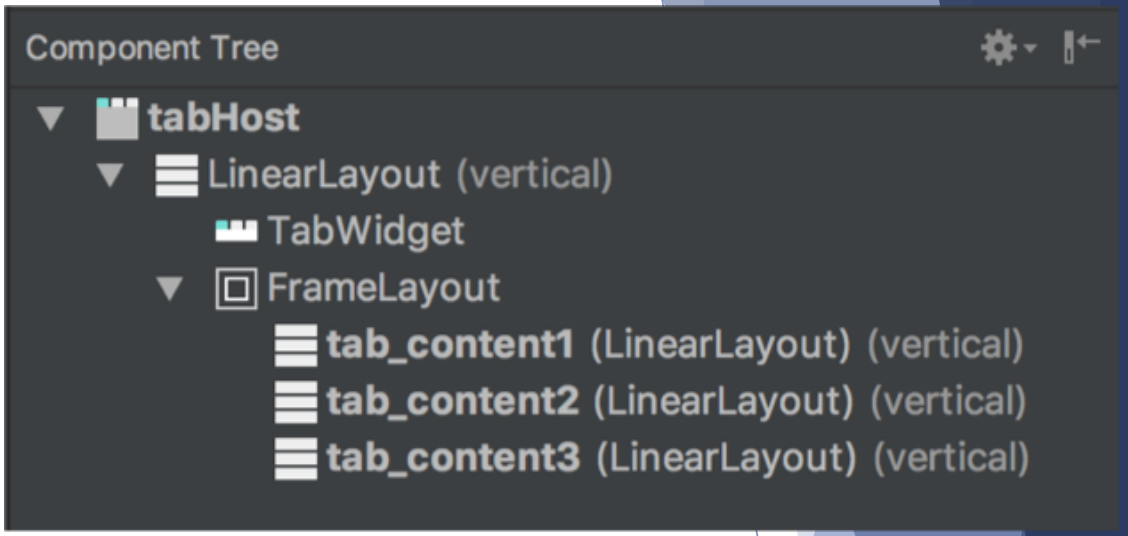
- Tab UI : 화면을 여러 개 준비한 후, 사용자가 버튼을 누르면 버튼과 연결된 화면을 보여주는 User Interface
- TabHost로 탭 화면을 만들려면 TabHost의 구조를 이해해야 함

- TabHost의 구조
  - TabHost : 탭 전체 영역
  - TabWidget : 탭 버튼이 들어갈 영역
  - FrameLayout(Content) : 탭 버튼 클릭시 표시할 화면 영역



Tab UI의 구현

: TabHost UI를 위한 Layout 구조




```
<?xml version="1.0" encoding="utf-8"?>
<TabHost>
    <LinearLayout>
        <TabWidget
            android:id="@android:id/tabs"/>
        <FrameLayout
            android:id="@android:id/tabcontent">
        </FrameLayout>
    </LinearLayout>
</TabHost>
```

TabWidget과 FrameLayout의 id는 변경 불가

Tab UI의 구현

: TabHost UI를 위한 TabSpec 등록

- 탭을 눌렀을 때 원하는 화면을 보여주려면 TabHost에 TabSpec을 추가해 주어야 함

- TabSpec : TabHost에 추가할 탭 화면의 태그, 탭에 표시할 내용(Indicator), 내용 등으로 구성

<TabSpec 등록의 예>

```
TabHost tabHost = findViewById(R.id.tabHost);
tabHost.setup();

TabHost.TabSpec spec = tabHost.newTabSpec("tab1");
spec.setIndicator("Tab 1");
spec.setContent(R.id.tab_content1);
tabHost.addTab(spec);
```

Tab UI의 구현

: TabHost UI 구현 실습

[실습] TabHostExample

1) TabHost 기본 형태를 구성합니다

2) FrameLayout 내에 다음과 같이 배경색이 다른 세 개의 LinearLayout을 Tab Content 영역으로 추가합니다.

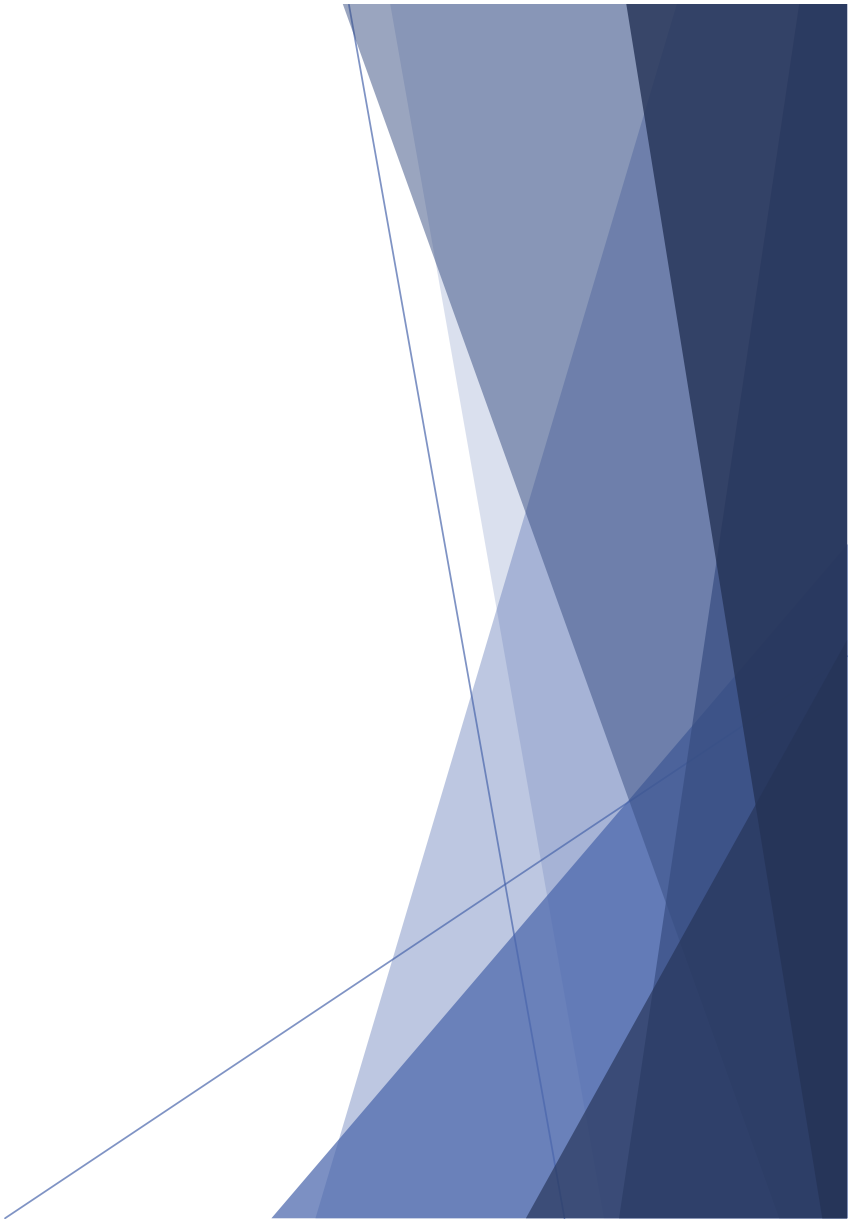
- id: tab_content1, background: #FF0000 (빨강)
- id: tab_content2, background: #00FF00 (녹색)
- id: tab_content3, background: #0000FF (파랑)

3) TabSpec을 이용, TabHost에 2)에서 작성한 세 Content 영역을 등록합니다. 각각의 탭 Indicator는 다음과 같이 합니다.

- tab_content1 -> “RED”, tag: red_tag
- tab_content2 -> “GREEN”, tag: green_tag
- tab_content3 -> “BLUE”, blue_tag

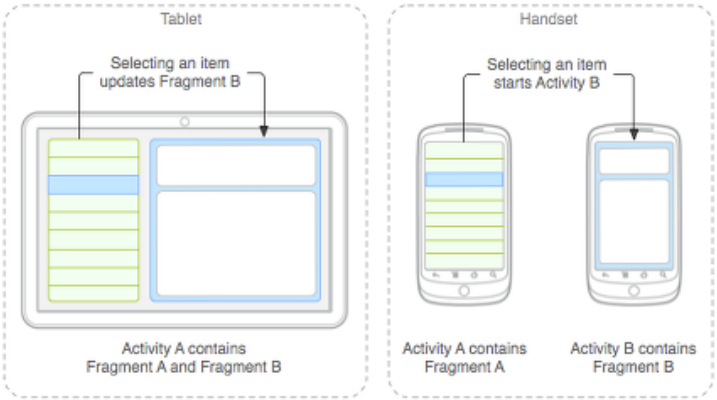
Advanced UI

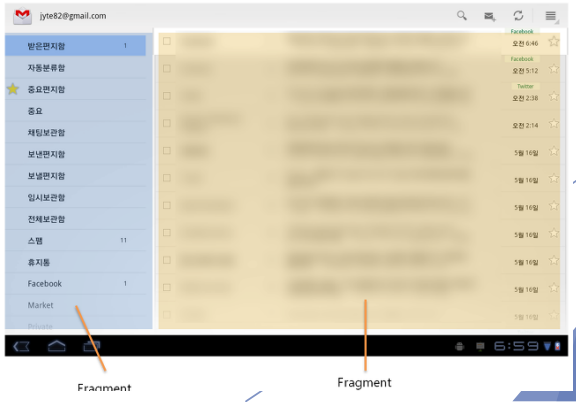
Fragment



Fragment

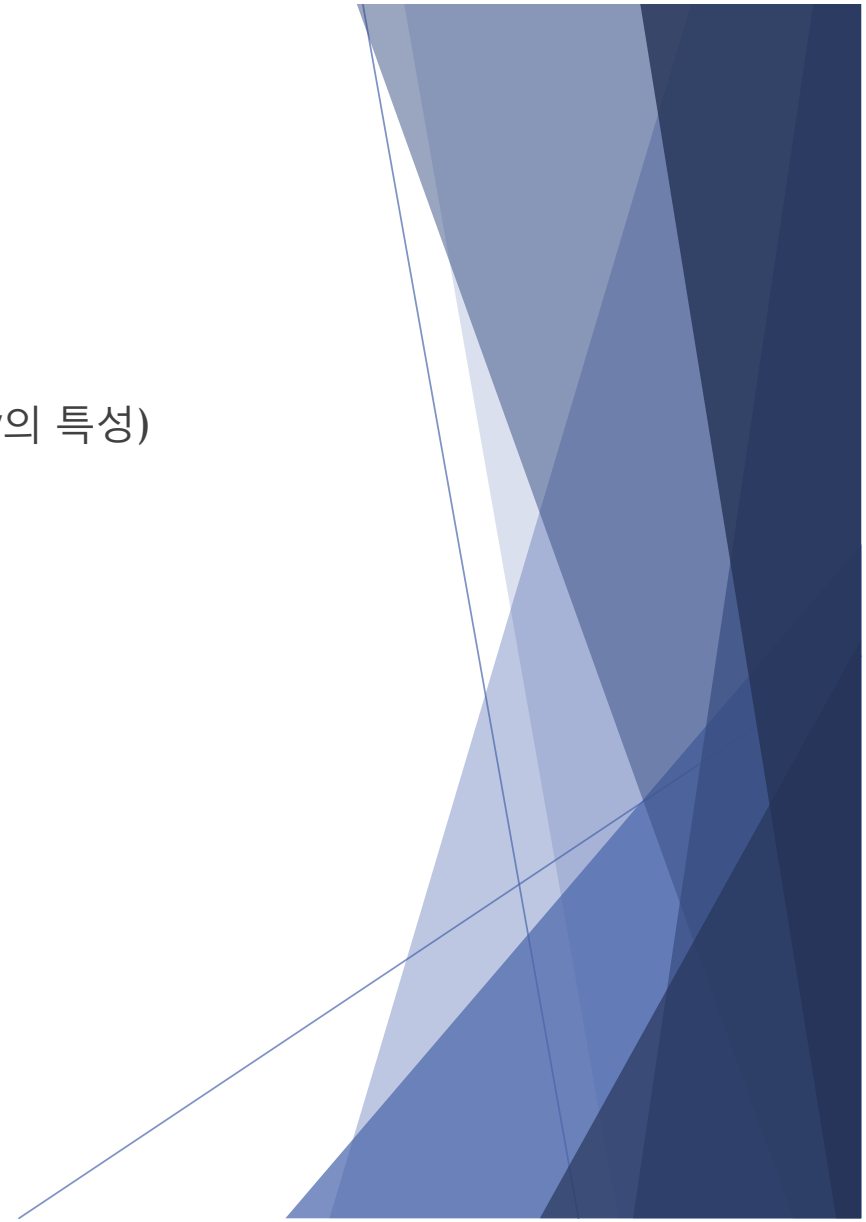
- Activity의 기본 개념은 한 화면에 보이는 모든 것을 관리하는 개념
- 다양한 태블릿 디바이스, 다이나믹한 앱 개발을 위해 Activity 개념으로는 부족, Fragment 개념이 도입(화면 분할 개념, 컴포넌트화)
- 기존 Activity는 하나의 화면밖에 사용할 수 없게 설계되지만 Fragment는 Activity와 비슷한 LifeCycle을 가지면서 여러 방식으로 화면을 넣을 수 있는 방법을 지원
- Android 3.0(HoneyComb)부터 지원





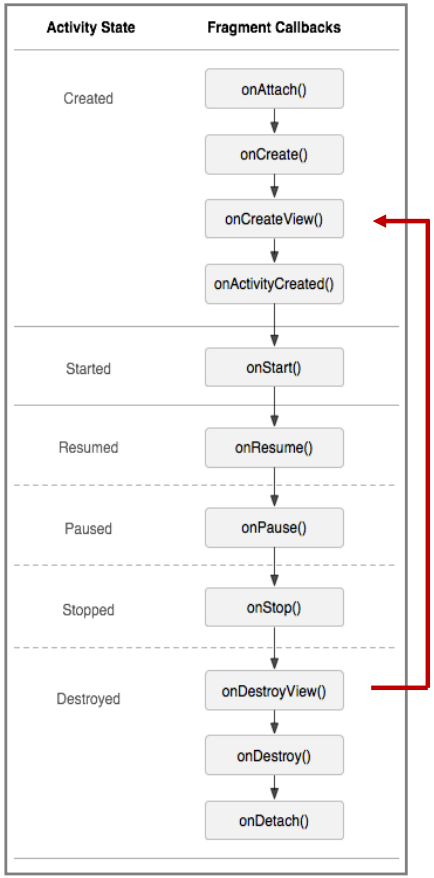
Fragment
: 특징

- 뷰 그룹의 일부가 되거나 레이아웃의 일부가 될 수 있음 (View의 특성)
  - Activity 내에서 다른 뷰와 공존할 수 있음
  - Activity 내에서만 존재, 단독으로는 존재하지 않음
  - Activity의 조각 혹은 Sub Activity로 활용
- 자신만의 라이프 사이클을 가지고 있음 (Activity의 특성)
  - Back Stack 사용 가능
  - 컨텍스트 객체는 아니며, 라이프 사이클을 액티비티에 종속됨



Fragment

: Fragment의 생명 주기(Life Cycle)



1.onAttach: fargment가 activity에 추가되고 나면 호출.
Activity가 파라미터로 전달된다.

2.onCreate: 프래그먼트가 생성될 때 호출된다. 프래그먼트의 필수 컴포넌트을 초기화한다.
Activity의 onCreate() 메소드와 비슷

3.onCreateView: 실제 사용할 뷰를 만드는 작업을 한다.
LayoutInflater를 인자로 받아 layout 으로 설정된 XML을 연결

4.onActivityCreated:
Activity에서 Fragment를 모두 생성하고 난 다음에 (Activity의 onCreate()가 마치고 난 다음)에 호출.

5.onDestroyView: Fragment의 View가 모두 소멸될 때 호출
View에 관련된 모든 자원들이 사라지게 된다.

6.onDestroy: Fragment를 더 이상 사용하지 않을 때.
Activity와의 연결은 아직 끊어진 상태는 아님.

7.onDetach: Activity와의 연결을 끊으며 Fragment에 관련된 모든 자원들이 사라지게 된다.

Fragment

: Fragment 작성

- Fragment에서 사용할 Layout 작성
  - 작성법은 Activity의 Layout 작성법과 동일
- Fragment를 상속받아 클래스를 생성

<Fragment 생성의 예>

```
public class FirstFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_first, container, false);
        return rootView;
    }
}
```

Fragment

: Layout에서 Frament 사용하기

- Activity의 Layout XML 내에서 Fragment는 <fragment> 태그로 등록하여 사용
- class 속성에 Fragment의 클래스 명을 입력해 줘야 함

<Layout 내 fragment 넣기>

```
<fragment
    android:id="@+id/first_fragment"
    class="com.example.android.fragmentexample.FirstFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Fragment

: Layout에 선언된 Frament 객체 얻기

- Layout 내에 선언된 Fragment는 FragmentManager의 findFragmentById 메서드를 이용하여 객체를 얻어와야 한다

<Layout에 선언된 Fragment 객체 얻기>

```
FirstFragment firstFragment = (FirstFragment)getSupportFragmentManager()
                                    .findFragmentById(R.id.first_fragment);
```

Fragment

: Activity 코드에서 Fragment 다루기

- Activity 코드 내에서 Fragment를 추가하거나 빼거나 교체하거나 할 때에는 FragmentTransaction 클래스를 얻어서 처리해야 함
- 트랜젝션은 beginTransaction 메서드로 시작, commit() 메서드로 수행됨

<Activity 코드에서 Fragment 다루기>

```
getSupportFragmentManager()
        .beginTransaction()
        .replace(R.id.container, firstFragment)
        .commit();
```

Fragment

: 액티비티와의 연결 및 통신

- Fragmemt는 그 자신만으로는 존재할 수 없고, Activity 내에서 동작해야 함
- 자신을 품고 있는 Activity의 객체를 받아오려면 getActivity() 메서드를 이용

MainActivity에 있는 메서드 ➡

<Fragment에서 Activity 객체 활용하기>

```
MainActivity activity = (MainActivity)getActivity();
activity.changeFragment("second");
```

Fragment

: 액티비티와의 연결 및 통신

[실습] FragmentExample

1) 프래그먼트 2개를 만듭니다.

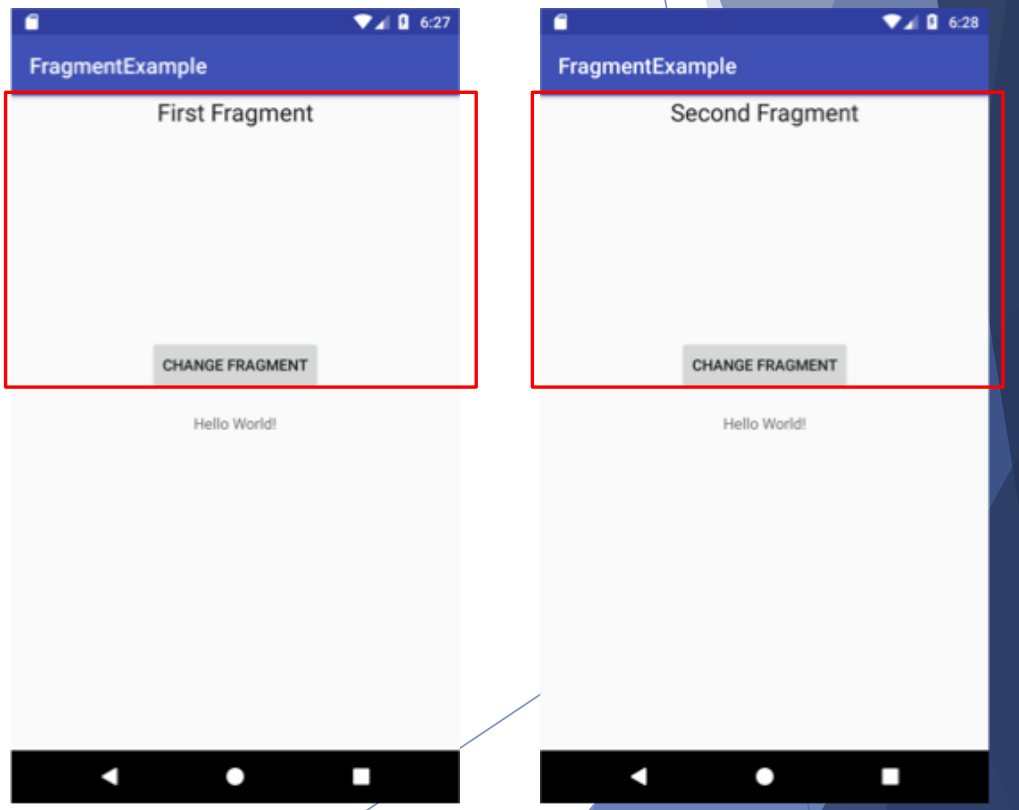
- FirstFragment
- SecondFragment

2) 다음과 같이 프래그먼트를 생성합니다.

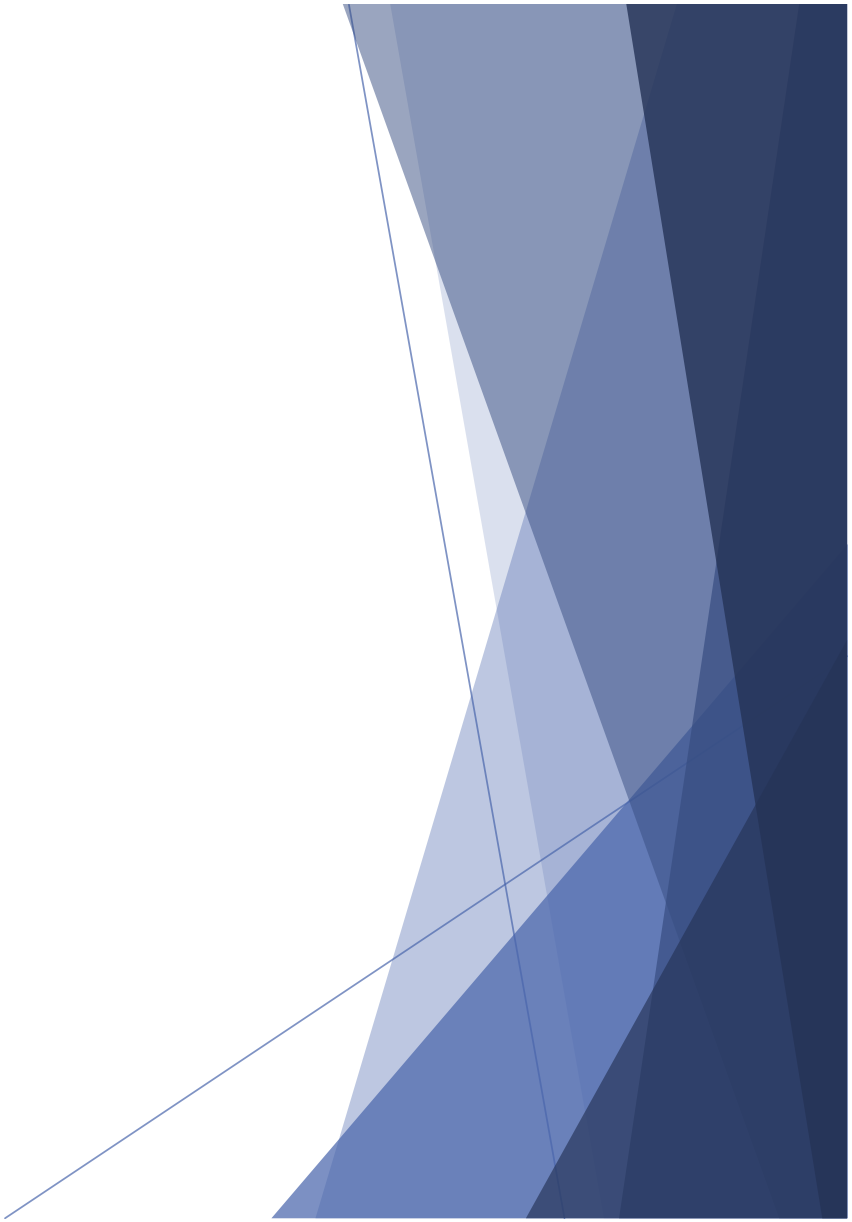
- FirstFragment : Main 레이아웃에 포함
- SecondFragment : 별도로 new 로 생성

3) 다음과 같이 프래그먼트 내 버튼의 기능을 구현합니다.

- 메인 액티비티에 changeFragment 메서드를 구현하고
- 프래그먼트의 버튼을 누르면 메인 액티비티에서 프래그먼트를 교체합니다.



AdapterView

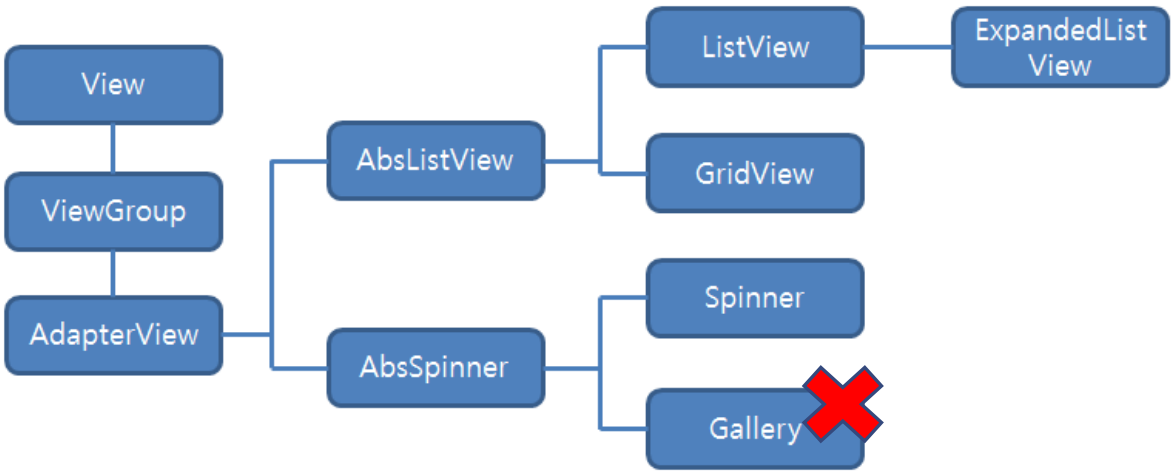


AdapterView

- AdapterView는 항목을 나열하는 뷰를 지칭
- 하나의 뷰에 데이터를 나열하고 그 중 하나를 사용자가 선택하도록 하는 뷰

- AdapterView의 종류
  - ListView
  - ExpandableListView
  - GridView
  - Spinner

- 가장 대표적인 AdapterView는 ListView

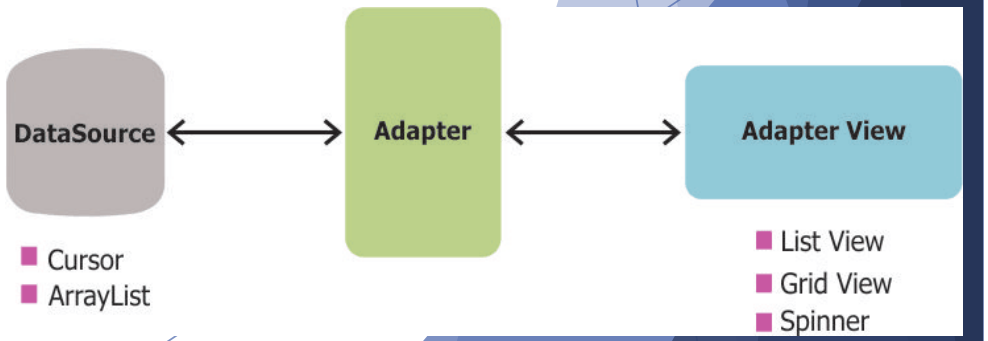


AdapterView

: AdapterView, Adapter 그리고 DataSource

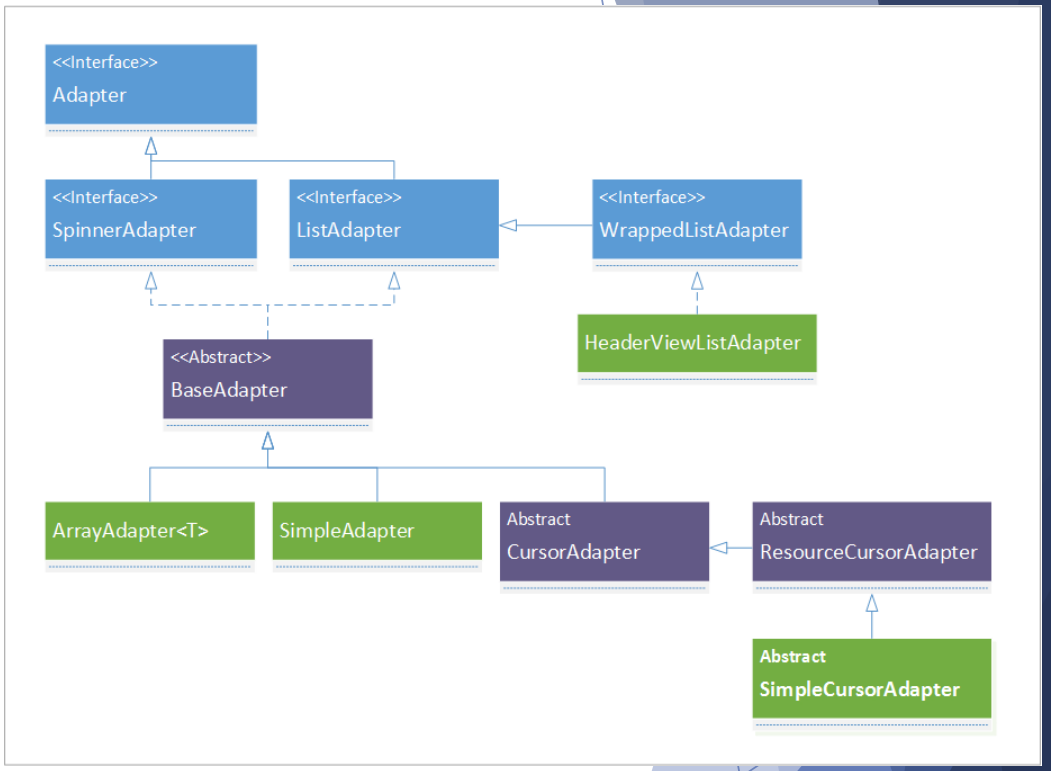
- AdapterView에는 직접 데이터를 받아오거나 뷰를 그리지 않음
- DataSource의 데이터를 받아와 View를 완성하여 AdapterView에 전달하는 역할은 Adapter가 수행

- DataSource
  - Content Provider
  - SharedPreference
  - Resources
  - SQLite
  - File System
  - Network 등



AdapterView

: Adapter



- AdapterView에 DataSource로부터 받은 데이터를 View로 렌더링하여 제공

- 안드로이드 내장 Adapter
  - ArrayAdapter : 문자열 배열을 DataSource로 함
  - SimpleAdapter : 한 항목에 여러 데이터를 넣어야 할 경우에 사용
  - CursorAdapter : 안드로이드 내장 Database의 결괏값을 직접 이용할 때 사용

- 필요할 시에는 Custom Adapter를 직접 구현하여 활용해야 할 경우

AdapterView

: ArrayAdapter를 이용한 데이터와 ListView의 연결

- ListView는 가장 널리 사용되는 AdapterView

- ArrayAdapter는 한 항목에 문자열 하나를 나열할 때 사용

[실습] ArrayAdapterExample

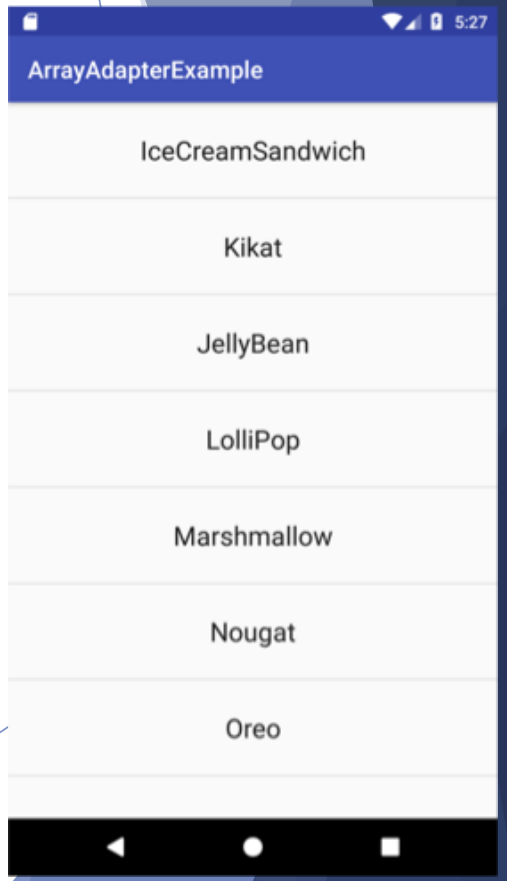
1) 프로젝트를 만들고 예제 파일을 다운받아 리소스들을 적합한 위치에 복사

- list_item_main.xml (리스트 항목 레이아웃 파일)
- arrays.xml (배열 리소스)

2) 복사한 리소스들을 확인합니다.

3) activity_main.xml을 LinearLayout으로 바꾸고 listView를 넣은 후 다음과 같이 설정합니다

- id: list



AdapterView

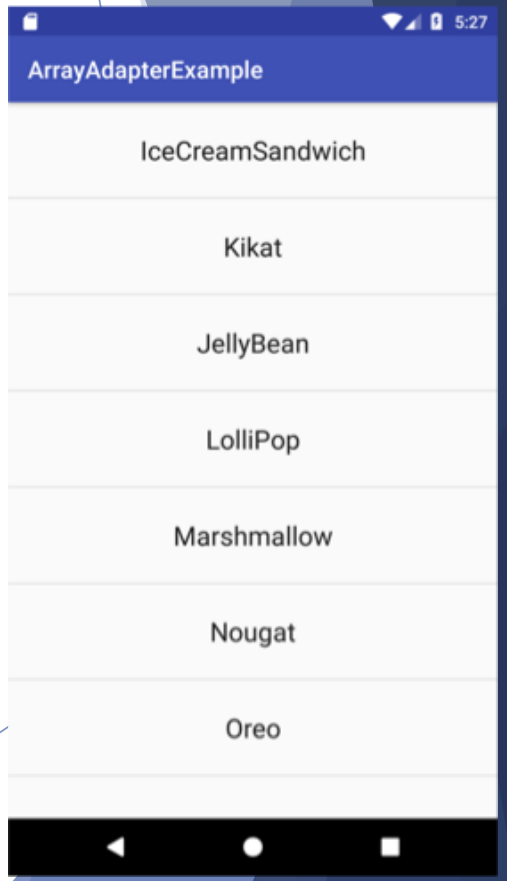
: ArrayAdapter를 이용한 데이터와 ListView의 연결

[실습] ArrayAdapterExample

4) ArrayAdapter를 선언하고 데이터와 데이터를 표시할 뷰를 연결합니다.

```
String[] data = getResources()
                    .getStringArray(R.array.android_list);

ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
                android.R.layout.simple_list_item_1, data);
```



<리스트 뷰를 위한 안드로이드 내장 xml>

- simple_list_item_1 : 문자열 데이터 하나
- simple_list_item_2 : 문자열 데이터 둘
- simple_list_item_multiple_choice : 문자열과 오른쪽 체크박스
- simple_list_item_single_choice : 문자열과 오른쪽 라디오 버튼

AdapterView

: ArrayAdapter를 이용한 데이터와 ListView의 연결

[실습] ArrayAdapterExample

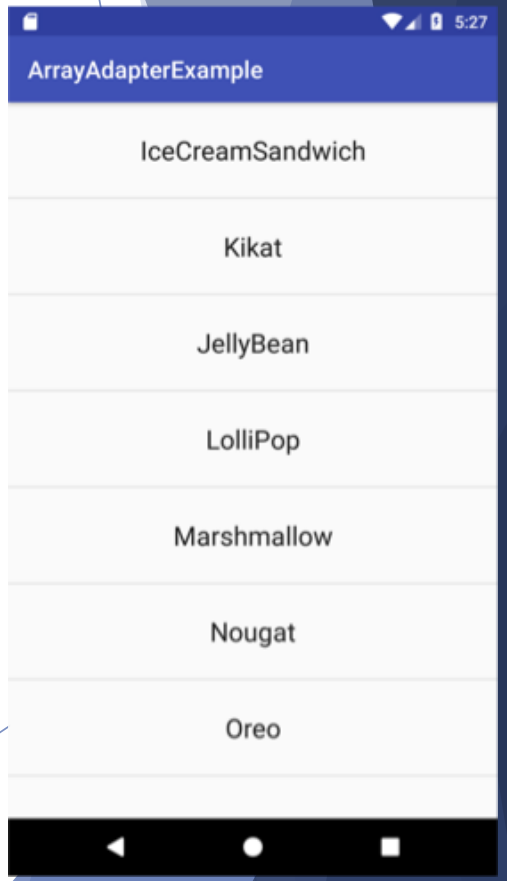
5) ListView에 어댑터를 연결합니다. (연결하지 않으면 내용이 표시되지 않음)

```
list.setAdapter(adapter);
```

6) 리스트 뷰를 클릭했을 때, 데이터를 가져올 수 있도록 AdapterView.OnItemClickLister를 구현합니다.

```
public class MainActivity extends AppCompatActivity
        implements AdapterView.OnItemClickListener {
    @Override
    public void onItemClick(AdapterView<?> adapterView,
                            View view,
                            int position,
                            long id) {

    }
}
```



AdapterView

: ArrayAdapter를 이용한 데이터와 ListView의 연결

[실습] ArrayAdapterExample

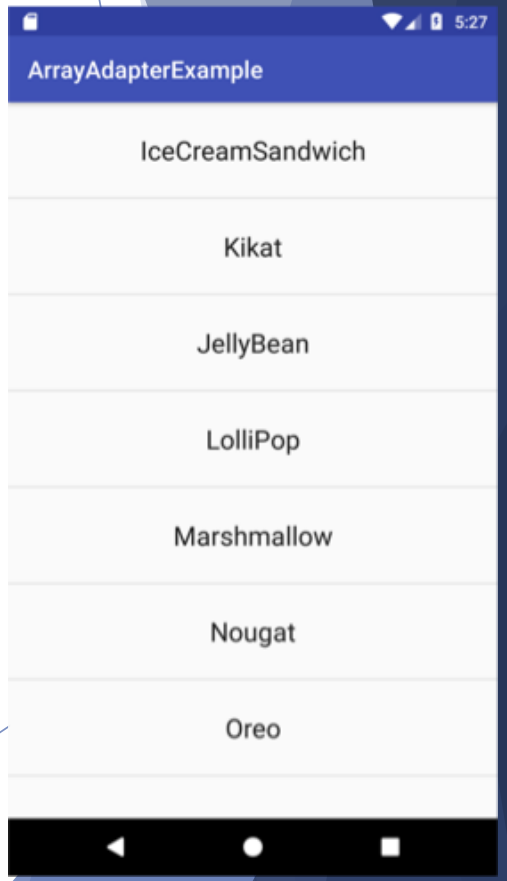
5) ListView에 이벤트 리스너를 등록

```
list.setOnItemClickListener(this);
```

추가 실습) ListView에 연결된 ArrayAdapter에 안드로이드 기본 제공 레이아웃이 아닌 커스텀 레이아웃을 연결해 봅니다.

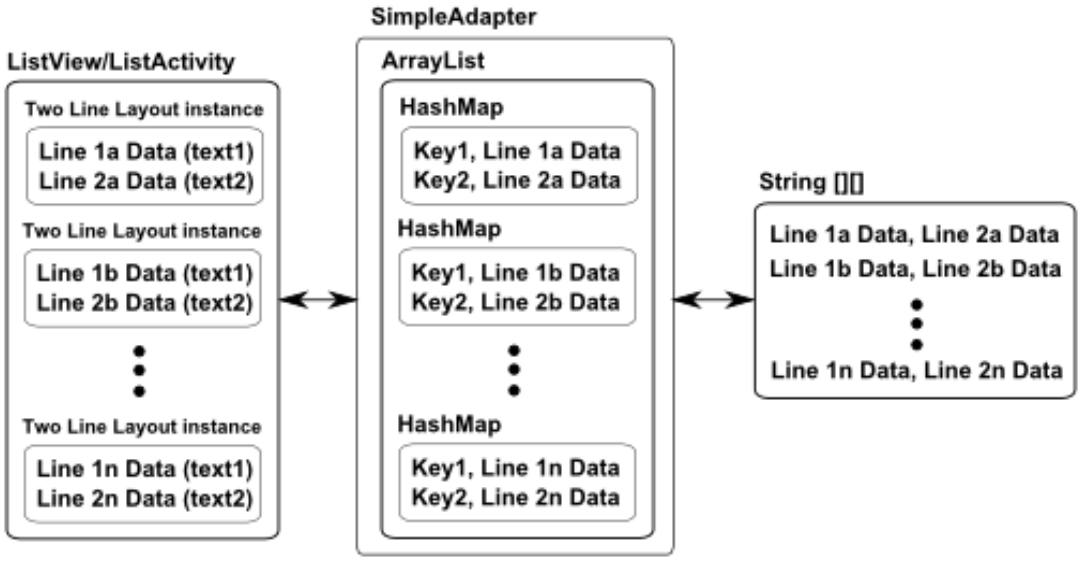
```
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
        R.layout.list_item_main, R.id.textView, data);
```

이때 주의해야 할 점은, 커스텀 레이아웃의 경우, ArrayAdapter가 데이터를 어느 ID에 연결해야 할지 알 수 없으므로 명시적으로 데이터를 연결할 위젯의 ID를 명시해 줘야 합니다.



AdapterView

: SimpleAdapter



- 한 항목에 데이터를 여러 개 나열해야 할 때
- HashMap을 가지고 있는 ArrayList를 만들어 SimpleAdapter에 전달해 주는 구조

[실습] SimpleAdapterExample

1) 프로젝트를 만듭니다.

2) activity_main.xml을 LinearLayout으로 바꾸고, ListView를 배치한 후, 다음과 같이 설정합니다.

- id: list

3) 필요한 변수들을 선언합니다.

```
private ListView list;  //  목록을 보여줄 위젯
private SimpleAdapter adapter;  //  어댑터
private ArrayList<HashMap<String, String>> data;     // 데이터
```

AdapterView
: SimpleAdapter

[실습] SimpleAdapterExample

4) 데이터로 넘겨줄 ArrayList를 초기화하고 데이터를 담습니다.

```
//  ArrayList 초기화
data = new ArrayList<>();

//  데이터를 넣읍시다
HashMap<String, String> map = new HashMap<>();
map.put("name", "채치수");
map.put("position", "센터");

data.add(map);
```

AdapterView

: SimpleAdapter

[실습] SimpleAdapterExample

5) 어댑터를 설정하고 리스트 뷰에 연결합니다.

맵 내에서 가져올 데이터의 키값 배열

```
adapter = new SimpleAdapter(this,
        data,
        android.R.layout.simple_list_item_2,
        new String[] {"name", "position"},
        new int[] {android.R.id.text1, android.R.id.text2});
list.setAdapter(adapter);
```

맵 내에서 가져올 데이터를 연결할 위젯의 아이디

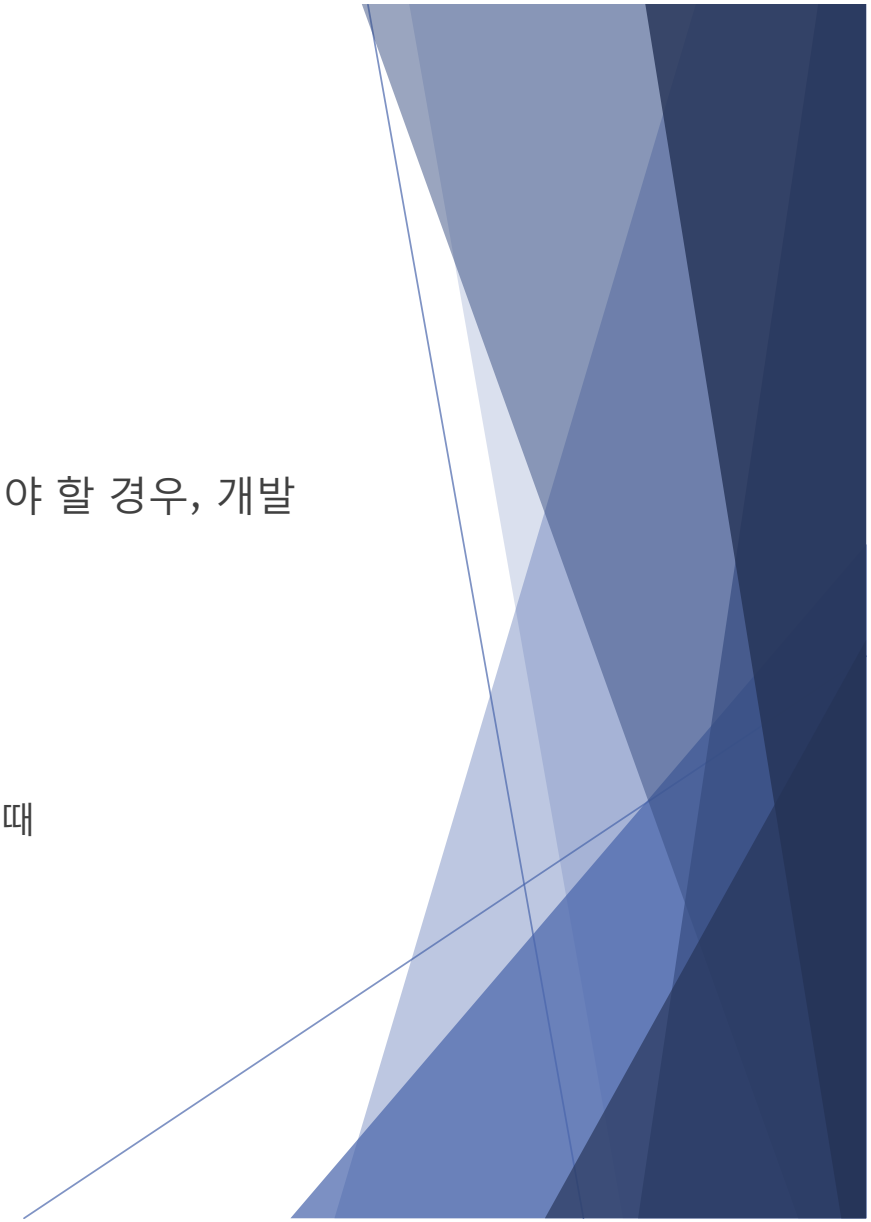
6) OnItemClickListener를 구현하고 아이템을 선택했을 때 Toast로 띄워주는 기능을 구현합니다.

AdapterView

: 커스텀 Adapter

- 내장 Adapter 들은 데이터의 문자열 출력 정도의 기능만 제공
- 그 이상의 데이터를 표현하며 다양한 방식의 이벤트를 구현해야 할 경우, 개발자가 직접 Adapter를 구현해야 함

- 직접 만들어야 할 상황
  - 개발자 알고리즘대로 항목의 데이터가 설정되어야 할 때
  - 개발자 알고리즘대로 항목별 뷰의 이벤트를 다르게 처리해야 할 때
  - 개발자 알고리즘대로 항목별 레이아웃을 다르게 적용해야 할 때



AdapterView

: 커스텀 Adapter

- 항목별 데이터를 추상화한 VO 클래스를 생성
  - 한 항목에 여러 가지의 데이터가 들어간다면 VO를 생성하고 관리하는 것이 좋음

```
class MovieVO {
    public int image;
    public String title;
    public String year;
}
```

  - VO는 단순히 데이터를 담고 있는 객체이기 때문에 데이터를 은닉해야 할 필요가 없어 getter/setter를 구현하지 않음

AdapterView

: 커스텀 Adapter

- Custom Adapter를 구현
  - BaseAdapter, ArrayAdapter, SimpleAdapter 등을 상속받아 Custom Adapter를 구현

```
class MovieAdapter extends ArrayAdapter<MovieVO> {
        public MovieAdapter(@NonNull Context context,
                            int resource,
                            @NonNull ArrayList<MovieVO> data) {
            super(context, resource);
                        //  ...
        }
        @Override
        public int getCount() {
            //  ...
        }
        @NonNull
        @Override
        public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
            //  ...
        }
    }
```

AdapterView

: 커스텀 Adapter

**getView의 두 번째 인자로 contentView가 넘어오는데,
최초로 보여지는 시점에는 null이므로 이때 한 번만 초기화**

- 레이아웃 객체를 담을 ViewHolder의 구현
  - 한 항목에 여러 가지의 데이터가 들어간다면 VO를 생성하고 관리하는 것이 좋음

```
class MovieHolder {
    public ImageView imageView;
    public TextView tvTitle;
    public TextView tvYear;

    public MovieHolder(View root) {
        imageView = root.findViewById(R.id.imageView);
        tvTitle = root.findViewById(R.id.tvTitle);
        tvYear = root.findViewById(R.id.tvYear);
    }
}
```

  - 각 항목의 레이아웃을 초기화할 때 LayoutInflater를 이용, inflate 하게 되는데, 이때 꽤 많은 부담이 발생
  - 바뀌는 것은 레이아웃이 아니라 데이터일 뿐이므로 최초로 나오는 한 순간만 초기화 작업을 수행

AdapterView

: 커스텀 Adapter

[실습] CustomAdapterExample

1) 프로젝트를 생성하고 다음의 작업을 수행합니다.

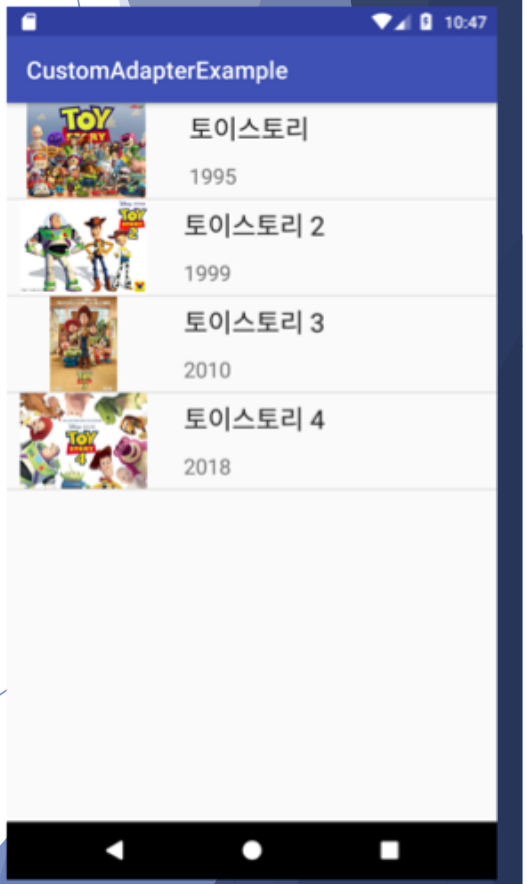
- activity_main.xml 을 LinearLayout으로 바꾸고 id는 list로 합니다.
- 샘플 파일에 포함된 list_item_movie.xml 을 /res/layout 디렉터리에 복사하고 확인합니다.
- 샘플 파일에 포함된 drawables.zip 압축을 풀고 파일들을 /res/drawables 디렉터리에 복사합니다.

2) 데이터를 담을 MovieVO 클래스를 생성합니다.

3) 이미 만들어진 리스트 항목 뷰를 저장할 MovieHolder 클래스를 생성합니다.

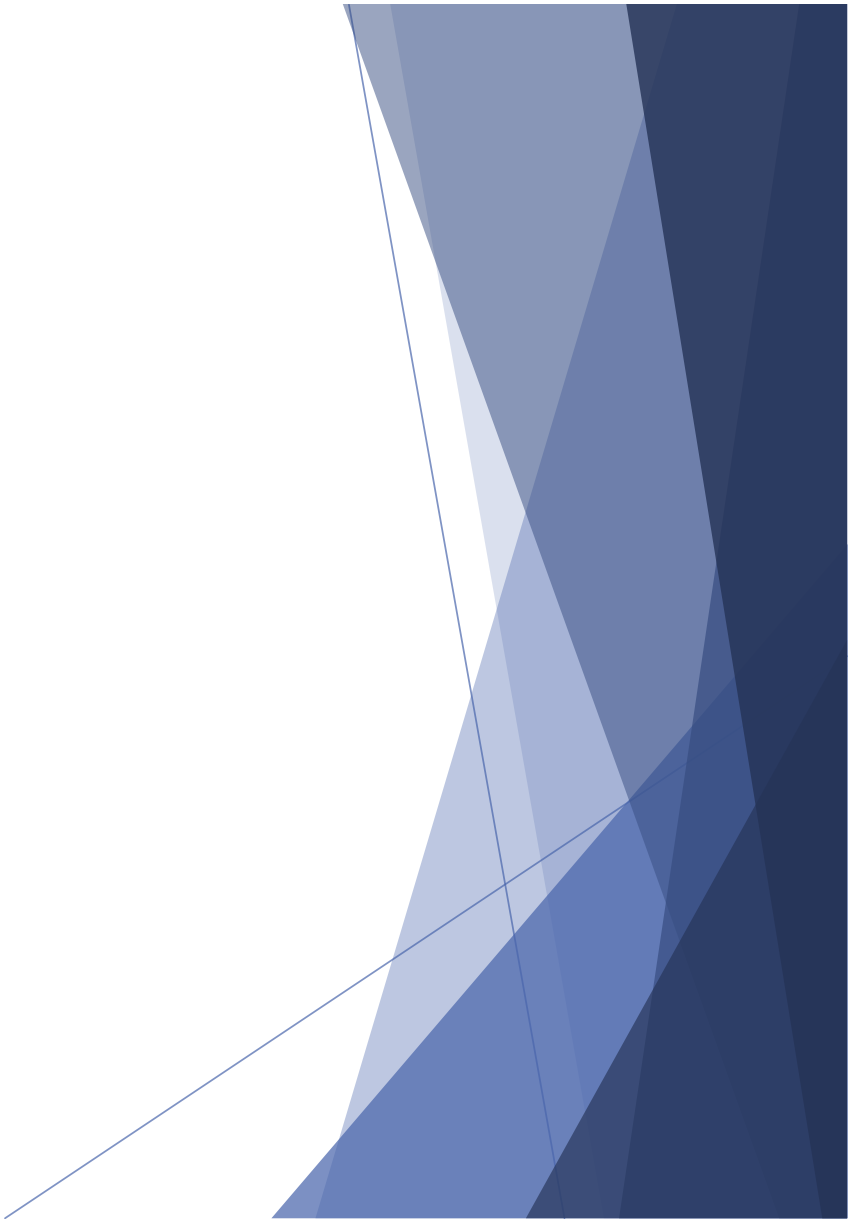
4) ArrayAdapter를 상속한 MovieAdapter를 만들고 어댑터를 생성, 데이터를 넣습니다.

5) 리스트에 어댑터를 연결하고 결과를 확인합니다.



Advanced UI

ViewPager

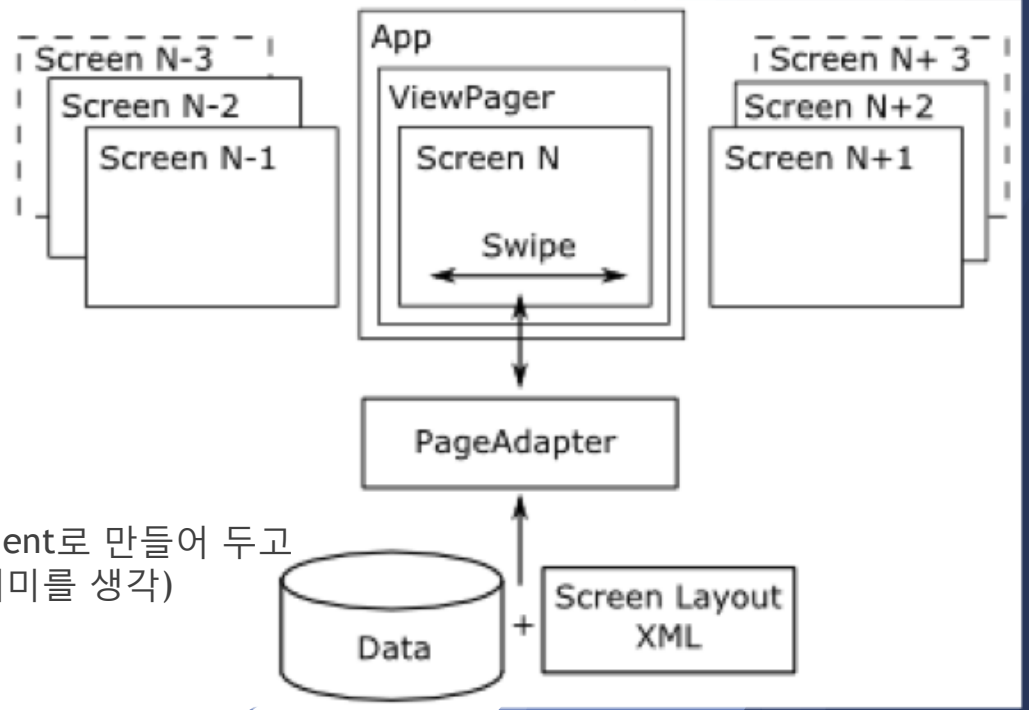


ViewPager

- 좌우로 슬라이드(Swipe) 하여 사용하는 액티비티(혹은 프래그먼트)를 전환하는 ViewGroup

- ViewPager는 support v4에서 제공하는 클래스
- ViewPager는 AdapterView의 일종
  - PageAdapter를 구현하여 등록해 주어야 함

- 구현하는 두 가지 방법
  - PagerAdapter: 기본 어댑터
  - **FragmentPagerAdapter** : 페이저의 각 화면을 Fragment로 만들어 두고 전환시키는 방식. 선호되는 방식(Fragment의 존재 의미를 생각)



ViewPager

[실습] ViewPagerExample

1) 프로젝트를 만듭니다.

2) activity_main.xml을 LinearLayout으로 변경하고, ViewPager를 배치한 후 다음과 같이 설정합니다.

- id: pager

```
<android.support.v4.view.ViewPager
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

pager는 support v4에서 제공하는 클래스이므로
android.support.v4.View.ViewPager 전체를 기술해 주어야 합니다.

3) Fragment 세 개를 새로 만들고, 각 페이지가 구분될 수 있도록 UI를 다르게 구성합니다.

ViewPager

[실습] ViewPagerExample

3) FragmentPagerAdapter를 상속받은 MyPagerAdapter를 구현합니다.

```
class MyPagerAdapter extends FragmentPagerAdapter {
        ArrayList<Fragment> fragments;

        public MyPagerAdapter(FragmentManager fm) {
            super(fm);
            //  Pager에 제공할 Fragment 생성 및 등록
        }

        @Override
        public Fragment getItem(int position) {
            return fragments.get(position); //  Pager에서 선택된 position의 Fragment 반환
        }

        @Override
        public int getCount() {
            return fragments.size();  //  Pager에서 사용할 수 있는 총 Fragment의 개수
        }
    }
```

ViewPager

[실습] ViewPagerExample

4) Pager에 어댑터를 등록

```
ViewPager pager = findViewById(R.id.pager);
MyPagerAdapter adapter = new
MyPagerAdapter(getSupportFragmentManager());
pager.setAdapter(adapter);
```