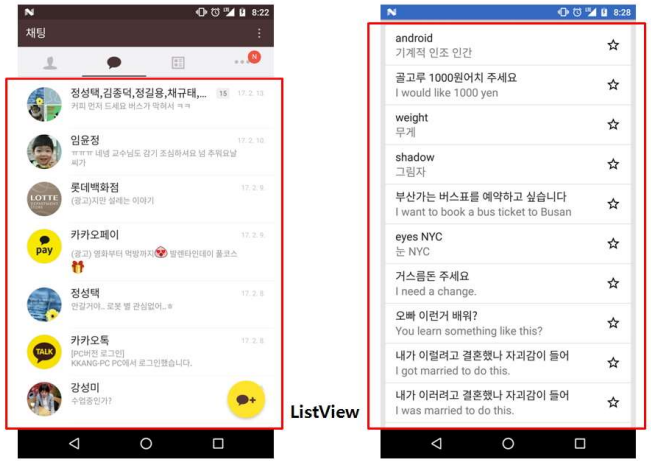
06

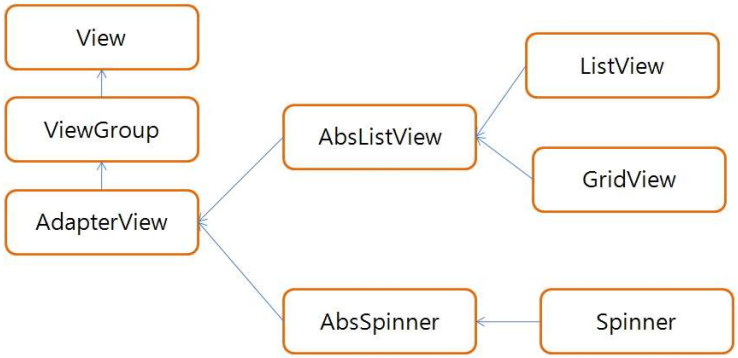
06-1. RecyclerView 와 ListView

다양한 뷰 활용

Adapter 와 AdapterView

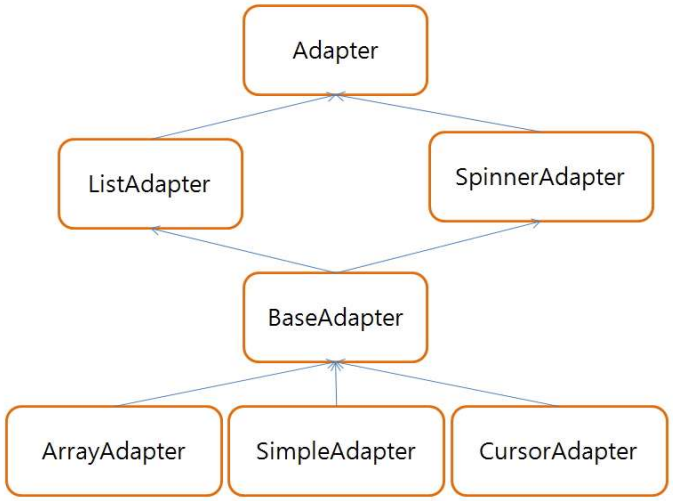
- AdapterView는 항목을 나열하는 뷰





Adapter 와 AdapterView

- AdapterView에 항목이 나열되어 화 면에 데이터가 나오려면 꼭 Adapter라는 클래스를 이용
- Adapter에게 일을 시 키고 Adapter가 AdapterView를 완성해주는 구조



ArrayAdapter

- 항목에 문자열 데이터를 순서대로 하나씩 나열


```
<ListView
    android:id="@+id/main_list"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    />
```

```
String[] datas=getResources().getStringArray(R.array.location);
ArrayAdapter<String> adapter=new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, datas);
listView.setAdapter(adapter);
```

- this: Context 객체
- android.R.layout.simple_list_item1: 항목 하나를 구성하기 위한 레이아웃 XML 파일 정보
- datas: 항목을 구성하는 데이터

ListView을 위해 제공되는 라이브러리의 XML 파일
- simple_list_item_1: 항목에 문자열 데이터 하나
- simple_list_item_2: 항목에 문자열 데이터 두 개 위아래 나열
- simple_list_item_multiple_choice: 문자열과 오른쪽 체크박스 제공
- simple_list_item_single_choice: 문자열과 오른쪽 라디오버튼 제공

ArrayAdapter

- 개발자가 항목 레이아웃 XML을 직 접 만들어 적용

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">
    <ImageView
        android:id="@+id/main_item_icon"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/icon"
        android:maxWidth="20dp"
        android:maxHeight="20dp"
        android:adjustViewBounds="true"/>
    <TextView
        android:id="@+id/main_item_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/main_item_icon"
        android:layout_marginLeft="16dp"/>
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"/>

</RelativeLayout>
```

ArrayAdapter

- ArrayAdapter에게 데이터를 출력할 TextView가 어떤 뷰인지 id값 지정

```
String[] datas=getResources().getStringArray(R.array.location);
ArrayAdapter<String> adapter=new ArrayAdapter<String>(this, R.layout.main_item, R.id.main_item_name, datas);
listView.setAdapter(adapter);
```

- 항목 선택 이벤트 처리

```
public class MainActivity extends AppCompatActivity implements AdapterView.OnItemClickListener{
    String[] datas;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //.....        listView.setOnItemClickListener(this);
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Toast t=Toast.makeText(this, datas[position], Toast.LENGTH_SHORT);
        t.show();
    }
}
```

- 항목이 동적으로 추가 또는 제거

```
adapter.notifyDataSetChanged();
```

SimpleAdapter

- 항목에 문자열 데이터를 여러 개 나열
- SimpleAdapter에 항목을 구성 하기 위한 데이터


```
ArrayList<HashMap<String, String>> datas=new ArrayList<>();
HashMap<String, String> map=new HashMap<>();
map.put("name","류현진");
map.put("content","제발 좀 MLB에서 봤으면 좋겠어..");
datas.add(map);
map=new HashMap<>();
map.put("name","오승환");
map.put("content","역시 돌직구 장난 아니구만..");
datas.add(map);
```

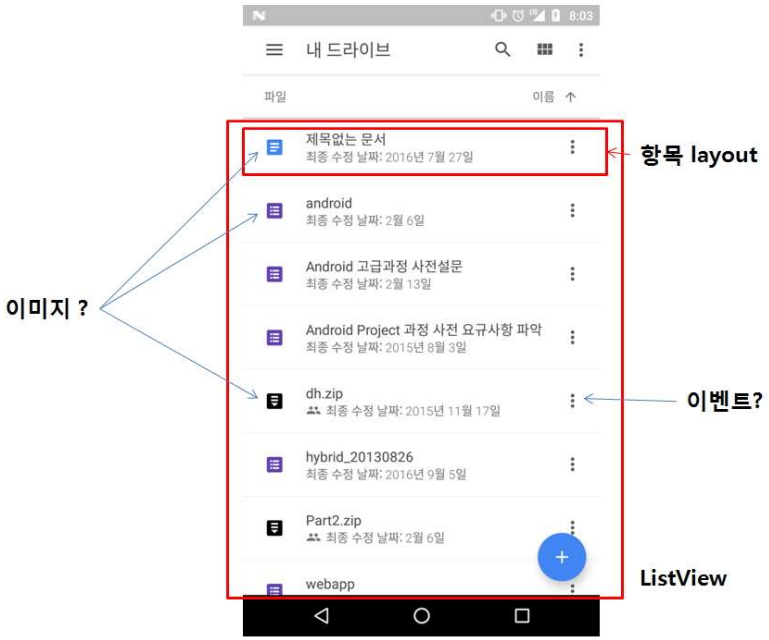
SimpleAdapter

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

- this: Context 객체
- datas: 항목 구성을 위한 데이터. ArrayList<HashMap<String,String>>
- android.R.layout.simple_list_item_2: 한 항목을 위한 레이아웃 XML
- new String[]{"name","content"}: 한 항목의 데이터를 가지는 HashMap에서 데이터를 추출하기 위한 키 값
- new int[]{android.R.id.text1, android.R.id.text2}: 추출된 데이터를 화면에 출력하기 위한 레이아웃 파일 내의 뷰 id 값

Custom Adapter

커스텀 Adapter가 필요한 예



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

Custom Adapter

커스텀 Adapter 작성 방법

```
public class DriveVO {
    public String type;
    public String title;
    public String date;
}
```

```
public class DriveAdapter extends ArrayAdapter<DriveVO>{
    Context context;
    int resId;
    ArrayList<DriveVO> datas;

    public DriveAdapter(Context context, int resId, ArrayList<DriveVO> datas){
        //...
    }

    @Override
    public int getCount() {
        //...
    }

    @NonNull
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {        //...
    }
}
```

Custom Adapter

- getCount( ) 함수는 전체 항목의 개수를 판단

```
public int getCount() {
    return datas.size();
}
```

- getView( ) 함수는 한 항목을 구성하기 위해 자동 호출

```
public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater=(LayoutInflater)context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        convertView=inflater.inflate(resId, null);
        ImageView typeImageView=(ImageView)convertView.findViewById(R.id.custom_item_type_image);
        //……
        final DriveVO vo=datas.get(position);
        titleView.setText(vo.title);
        dateView.setText(vo.date);
        if(vo.type.equals("doc")){
            typeImageView.setImageDrawable(ResourcesCompat.getDrawable (context.getResources(),
R.drawable.ic_type_doc, null));
        }
        //……
        menuImageView.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                Toast toast=Toast.makeText(context, vo.title+" menu click",Toast.LENGTH_SHORT);
                toast.show();
            }
        });
        return convertView;
    }
```

Custom Adapter

커스텀 Adapter 추가 고려 사항

- 레이아웃 초기화 성능 이슈 : LayoutInflater
- 레이아웃 초기화를 최초에 한 번만 수행

```
public View getView(int position, View convertView, ViewGroup parent) {
   if(convertView==null){
      LayoutInflater inflater=(LayoutInflater)context.getSystemService (Context.LAYOUT_INFLATER_SERVICE);
      convertView=inflater.inflate(resId, null);
      DriveHolder holder=new DriveHolder(convertView);
      convertView.setTag(holder);
   }
   //...
   return convertView;
}
```

- 뷰 획득 시 성능 이슈 : findViewById
- 획득한 뷰를 저장했 다가 그다음 이용 시 findViewById( ) 함수를 호출하지 않고 저장된 뷰를 그대로 이용

Custom Adapter

```
public class DriveHolder {
    public ImageView typeImageView;
    public TextView titleView;
    public TextView dateView;
    public ImageView menuImageView;

    public DriveHolder(View root){
        typeImageView=(ImageView)root.findViewById(R.id.custom_item_type_image);
        titleView=(TextView)root.findViewById(R.id.custom_item_title);
        dateView=(TextView)root.findViewById(R.id.custom_item_date);
        menuImageView=(ImageView)root.findViewById(R.id.custom_item_menu);
    }
}
```

- Holder를 Adapter에서 메모리에 유지

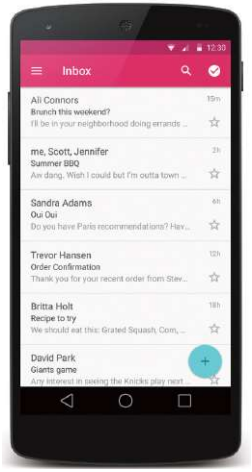
```
DriveHolder holder=new DriveHolder(convertView);
convertView.setTag(holder);
```

- 저장한 객체를 다시 획득해서 사용

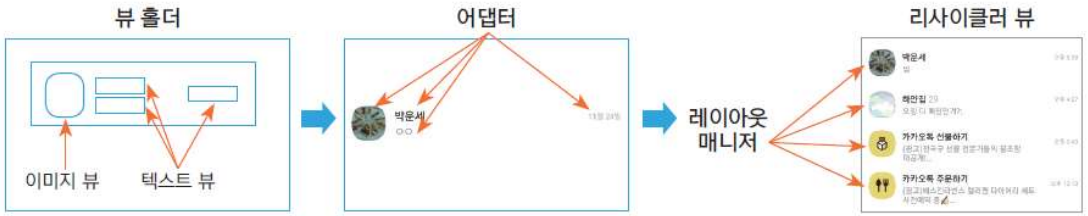
```
DriveHolder holder=(DriveHolder)convertView.getTag();
```

11-4 리사이클러 뷰 – 목록 화면 구성

리사이클러 뷰 기초 사용법



- 구성 요소
  - ViewHolder(필수): 항목에 필요한 뷰 객체를 가집니다.
  - Adapter(필수): 항목을 구성합니다.
  - LayoutManager(필수): 항목을 배치합니다.
  - ItemDecoration(옵션): 항목을 꾸밉니다.



11-4 리사이클러 뷰 – 목록 화면 구성

- 뷰 홀더 준비
  - 각 항목에 해당하는 뷰 객체를 가지는 뷰 홀더는 RecyclerView.ViewHolder를 상속받아 작성

• 뷰 홀더 준비

```
class MyViewHolder(val binding: ItemMainBinding): RecyclerView.ViewHolder(binding.root)
```

11-4 리사이클러 뷰 – 목록 화면 구성

- 어댑터 준비
  - 각 항목을 만들어 주는 역할
  - getItemCount(): 항목 개수를 판단하려고 자동으로 호출됩니다.
  - onCreateViewHolder(): 항목의 뷰를 가지는 뷰 홀더를 준비하려고 자동으로 호출됩니다.
  - onBindViewHolder(): 뷰 홀더의 뷰에 데이터를 출력하려고 자동으로 호출됩니다.

• 어댑터 준비

```
class MyAdapter(val datas: MutableList<String>):
    RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    override fun getItemCount(): Int {
        TODO("Not yet implemented")
    }
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
    RecyclerView.ViewHolder {
        TODO("Not yet implemented")
    }
    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        TODO("Not yet implemented")
    }
}
```

11-4 리사이클러 뷰 – 목록 화면 구성

• 항목의 개수 구하기

```
override fun getItemCount(): Int = datas.size
```

• 항목 구성에 필요한 뷰 홀더 객체 준비

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder =
    MyViewHolder(ItemMainBinding.inflate(LayoutInflater.from(parent.context),
        parent, false))
```

11-4 리사이클러 뷰 – 목록 화면 구성

• 뷰에 데이터 출력

```
override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
    Log.d("kkang","onBindViewHolder : $position")
    val binding = (holder as MyViewHolder).binding
    // 뷰에 데이터 출력
    binding.itemData.text = datas[position]
    // 뷰에 이벤트 추가
    binding.itemRoot.setOnClickListener {
        Log.d("kkang", "item root click : $position")
    }
}
```

11-4 리사이클러 뷰 – 목록 화면 구성

- 리사이클러 뷰 출력

• 리사이클러 뷰 출력

```
class RecyclerViewActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val binding = ActivityRecyclerViewBinding.inflate(layoutInflater)
        setContentView(binding.root)
        val datas = mutableListOf<String>()
        for(i in 1..10){
            datas.add("Item $i")
        }
        binding.recyclerView.layoutManager = LinearLayoutManager(this)
        binding.recyclerView.adapter = MyAdapter(datas)
        binding.recyclerView.addItemDecoration(DividerItemDecoration(this,
        LinearLayout Manager.VERTICAL))
    }
}
```

11-4 리사이클러 뷰 – 목록 화면 구성

- 항목을 동적으로 추가 · 제거
  - 항목을 구성하는 데이터에 새로운 데이터를 추가하거나 제거한 후 어댑터의 notifyDataSetChanged() 함수를 호출

• 항목 추가

```
datas.add("new data")
adapter.notifyDataSetChanged()
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare