● RecyclerView: 목록 화면 구성을 위해 제공되는 뷰이다. 대부분 앱에서 목록 화면을 RecyclerView로 만들고 있다. RecylcerView의 구성요소는 Adapter, LayoutManager, ViewHolder, ItemDecoration, ItemAnimation 클래스 등이 있다. Adapter, ViewHolder, LayoutManager 클래스는 필수 구성요소이고, ItemDecoration, ItemAnimation 클래스는 추가 구성요소이다.

| | |
|---|---|
| Adapter | RecyclerView 항목 구성. 앱에서 항목을 나열하는 뷰 - RecylcerView, ViewPager2, Spinner, AutoCompleteTextView, ListView 등 -를 사용하려면 항목을 나열하는 뷰를 만들어 주는 Adapter 클래스를 이용해야 한다. RecylcerView처럼 항목을 나열하는 뷰는 Adapter클래스를 사용해 항목을 만든 후에 그 항목을 뷰로 출력한다. |
| ViewHolder | 각 항목 구성 뷰의 재활용을 목적으로 View Holder 역할 |
| LayoutManager | 항목의 배치 |
| ItemDecoration | 항목 꾸미기 |
| ItemAnimation | 아이템이 추가, 제거, 정렬될 때의 애니메이션 처리 |

▶ Adapter와 ViewHolder

1) RecyclerView 화면을 구성하기 위해 액티비티의 레이아웃 XML 파일에 RecyclerView를 준비한다.

```
<androidx.recyclerview.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

2) ViewHolder 클래스를 정의해 준다: RecyclerView에는 Adapter에 ViewHolder의 적용이 강제돼 있기 때문이다. ViewHolder의 역할은 항목을 구성하기 위한 뷰들을 findViewById 해주는 역할을 한다. ViewHolder 객체를 Adapter 내부의 메모리에 유지하면, ViewHolder로 최초에 한 번만 fidnViewById를 사용하면 되고, 이로서 findViewById에 의한 성능 이슈를 해결할 수 있다. ViewHolder 클래스를 상속받아 작성한다. 이곳에서 항목을 구성하기 위한 뷰를 한 번만 FindViewById 할 수 있게 알고리즘을 구현하면 된다. ViewBinding 기법을 이용하면, 더 간단하게 만들 수 있다. 바인딩 객체에 이미 뷰가 등록되어 있으므로 이를 유지해주면 된다.

```
class MyViewHolder extends RecyclerView.ViewHolder {
    ItemBinding binding;
    MyViewHolder(ItemBinding binding) {
        super(binding.getRoot());
        this.binding = binding;
    }
}
```

3) ViewHolder을 내부적으로 이용하는 Adapter을 하나 만들어 준다.

```
Class MyRecylcerAdapter extends RecyclerView.Adapter<MyViewHolder> {
    // 항목 구성 데이터
    private List<String> list;
    public MyRecyclerAdapter(List<String>list) {
        this.list = list;
    }
    // ViewHolder 객체 반환 - 레이아웃 XML 파일의 infate을 담당
    // 항목을 구성하기 위한 레이아웃 XML 파일의 바인딩 객체를 ViewHolder의 생성자로 넘겨주며,
    // 만들어진 ViewHolder 객체를 onCreateViewHolder() 함수에서 반환하면 메모리에 유지했다가
    // onBindViewHolder() 호출 시 매개변수로 전달하는 구조이다.
    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        ItemBinding binding = ItemBinding.inflate(LayoutInflator.from(viewGroup.getConte
        return new MyViewHolder(binding);
    }
    // 각 항목을 구성하기 위해서 호출
    @Override
    public void onBindViewHolder(MyViewHolder viewHolder, int position) {
        String text = list.get(position);
        viewHolder.binding.itemTextView.setText(text);
    }
    // 항목 개수
    @Override
    public int getItemCount() {
        return list.size();
    }
}
```

4) 이렇게 만든 Adapter을 RecyclerView에 적용한다.

```
binding.recyclerView.setLayoutManager(new LinearLayoutManager(this));
binding.recyclerView.setAdapter(new MyRecyclerAdapter(list));
```

▶ LayoutManager: RecylcerView를 사용하면서 반드시 지정해 주어야 하는 필수 구성요소이며, 각 항목을 어떻게 배치할 것인지를 결정한다.

| | |
|---|---|
| LienarLayoutManager | 수평, 수직으로 배치 |
| GridLayoutManager | 그리드 화면으로 배치 |
| StaggeredGridLayoutManager | 높이가 불규칙한 그리드 화면으로 배치 |

☞ LinearLayoutManager: 기본이 세로 방향이지만, 가로로 지정하여 각 항목이 가로로 나열되게 구성할 수 있다.

```
LinearLayoutManager linearManager = new LinearLayoutManager(this);
linearManager.setOrientation(LinearLayoutManager.HORIZONTAL);
binding.recyclerView.setLayoutManager(linearManager);
```

☞ GridLayoutManager: 그리드(grid) 화면 만들기. 두 번째 매개변수는 열의 개수이다.

```
GridLayoutManager gridManager = new GridLayoutManager(this, 2);
bidning.recyclerView.setLayoutManager(gridManager);
```

열의 개수 외에 방향 지정도 가능하다. 가로로 지정하면 세로로 2칸씩 차지하면서 가로로 나열된다.

```
GirdLayoutManager gridManager = new GridLayoutManager(this, 2, GridLayoutManager.HORIZON
```

항목이 위부터 나열되는 것이 아니라, 아래부터 나열되게 할 수 있다. 마지막 매개 변수 값을 true로 지정하면 항목이 아래부터 위쪽으로 나열되게 구성한다.

```
GridLayoutManager gridManager = new GridLayoutManager(this, 2, GridLayoutManager.VERTICA
```

☞ StaggeredGridLayoutManager: 각 항목의 데이터가 차지하는 면적에 따라 크기가 다양하게 배치된다. 한 항목이 길 때는 그 옆에 여러 항목이 배치된다.

```
StaggeredGridLayoutManager sgManager = new StaggeredGridLayoutManager(2, StaggeredGridLa
```

▶ ItemDecoration: 각 항목을 다양하게 꾸밀 수 있다. 세 개의 함수를 제공하며, 필요할 때 각 함수를 재정의해서 작성하면 된다.

| | |
|---|---|
| onDraw | 항목을 배치하기 전에 호출 |
| getItemOffsets | 각 항목을 배치할 때 호출 |
| onDrawOver | 모든 항목이 배치된 후에 호출 |

ItemDecoration을 이용하려면, ItemDecoration을 상속받는 개발자 클래스를 정의한다. 그런 다음, 필요한 함수를 재정의한다.

☞ getItemOffsets(): 각 항목이 구성될 때마다 호출되며, 첫 번째 매개변수로 항목을 구성하기 위한 사각형 정보가, 두 번째 매개변수로 항목을 구성하기 위한 뷰가 전달된다. 이 두 매개변수를 이용하여 항목을 다양하게 구성할 수 있다. 첫 번째 매개변수로 항목을 구성하기 위한 사격형 정보가, 두 번째 매개변수로 항목을 구성하기 위한 뷰가 전달된다. 이 두 매개변수를 이용하여 항목을 다양하게 구성할 수 있다.

```
class MyItemDecoration extends RecyclerView.ItemDecoration {
    @Override
    public void getItemOffsets(Rect outRect, View view, RecyclerView parent, RecyclerVie
        super.getItemOffsets(outRect, view, parent, state);
        // 항목의 index 값 획득
        int index = parent.getChildAdapterPosition(view) + 1;

        if (index % 3 == 0) {
            //left, top, right, bottom
            outRect.set(20, 20, 20, 60);
        } else {
            outRect.set(20, 20, 20, 20);
        }

        view.setBackGroundColor(0xFFECE9E9);
        ViewCompat.setElevation(view, 20.0f);
    }
}
```

각 항목의 네 방향에 20픽셀로 여백을 두었다. 그리고 index 값으로 계산하여 항목을 3개씩 묶어, 그다음 3개 항목이 나올 때의 세로 방향 여백을 60으로 지정하여 좀 더 멀리 떨어지도록 구성하였다. 또한, 각 항목에 배경색을 지정하였으며, 위에 떠 있는 듯한 효과를 주었다.

이렇게 구성한 ItemDecoration을 RecyclerView에 적용시키면 된다.

```
recyclerView.addItemDecoration(new MyItemDecoration());
```

☞ onDraw(): 항목이 출력되기 전에 어떤 동작(예를 들면 RecyclerView에 그림 그리기)을 할 수 있다.

```
@Override
public void onDraw(Canvas c, RecyclerView parent, RecylcerView.State state) {
    super.onDraw(c, parent, state);
    //RecyclerView의 사이즈 계산
    int width = parent.getWidth();
    int height = parent.getHeight();

    Paint paint = new Paint();
    paint.setColor(Color.RED);
    c.drawRect(0, 0, width/3, height, paint);
    paint.setColor(Color.BLUE);
    c.drawRect(width/3, 0, width/3*2, height, paint);
    paint.setColor(Color.GREEN);
    c.drawRect(width/3*2, 0, width, height, paint);
}
```

RecyclerView에 항목이 추가되기 전에 사각형 3개를 각각의 색상으로 채웠으며, 이후 항목이 추가되어 onDraw() 함수에서 그린 그림 위에 항목이 위치하게 된다.

☞ onDrawOver(): RecyclerView에 항목이 모두 추가된 후에 그림을 그리기 위한 함수이다.

```
@Override
public void onDrawOver(Canvas c, RecyclerView parent, RecyclerView.State state) {
    super.onDrawOver(c, parent, state);
    // RecyclerView의 사이즈 계산
    int width = parent.getWidth();
    int height = parent.getHeight();
    // 이미지 사이즈 계산
    Drawable dr = ResourcesCompat.getDrawable(getResources(), R.drawable.android, null);
    int drWidth = dr.getIntrinsicWidth();
    int drHeight = dr.getIntrinsicHeight();

    int left = width/2 - drWidth/2;
    int top = height/2 - drHeight/2;
    c.drawBitmap(BitmapFactory.decodeResource(getResources(), R.drawable.android), left,
}
```

항목이 추가 된 후 항목 위에 반투명 이미지를 그린 예이다.

---

```
// activity_main.xml - 전체 화면을 위한 xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.recyclerview.widget.RecyclerView xmlns:android="http://schemas.android.com/apk
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:id="@+id/recycleView">
</androidx.recyclerview.widget.RecyclerView>
```

```
// item.xml - 항목 하나를 위한 xml
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/itemTextView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="15sp"
    android:padding="8dp">

</TextView>
```

```
// MainActivity.java

package com.android.practicerecyclerview;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.res.ResourcesCompat;
import androidx.core.view.ViewCompat;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import android.graphics.BitmapFactory;
import android.graphics.Canvas;
```

```
import android.graphics.DrawFilter;
import android.graphics.Rect;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.android.practicerecyclerview.databinding.ActivityMainBinding;
import com.android.practicerecyclerview.databinding.ItemBinding;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityMainBinding binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        List<String> list = new ArrayList<>();
        for (int i = 0; i < 20; i++) {
            list.add("Item=" + i);
        }

        // 항목을 어떻게 배치할 건지 정하는 역할.
        binding.recycleView.setLayoutManager(new LinearLayoutManager(this));
        binding.recycleView.setAdapter(new MyAdapter(list));
        // 항목을 꾸미는 역할
        binding.recycleView.addItemDecoration(new MyItemDecoration());

    }

    // 항목을 구현하기 위한 필요한 View들을 선언하고 획득한다.
    // 그리하여 그 뷰 객체들을 홀딩하고 있는 역할.
    // 예를 들면 항목이 이미지 뷰, 텍스트 뷰1, 텍스트 뷰2로 구성된다고 치면,
    // 그 뷰들을 선언하고 획득하여 객체들을 가지고 있는 역할.
    // 1) 변수들을 선언하고, findViewById로 뷰의 객체를 획득한다.
    // 2) binding 객체를 선언한다
    class MyViewHolder extends RecyclerView.ViewHolder {
        ItemBinding binding;

        MyViewHolder(ItemBinding binding) {
            super(binding.getRoot());
            this.binding = binding;
        }
    }

    // 항목을 완성하는 역할. ViewHolder의 View을 이용해서 각각의 항목에
    // 데이터를 넣거나, 이벤트를 걸거나 해서 항목을 완성한다
    class MyAdapter extends RecyclerView.Adapter<MyViewHolder> {
        private List<String> list;

        public MyAdapter(List<String> list) {
            this.list = list;
        }
```

```
        // MyViewHolder 준비
        @Override
        public MyViewHolder onCreateViewHolder(ViewGroup viewGorup, int i) {
            ItemBinding binding = ItemBinding.inflate(LayoutInflater.from(viewGorup.getC
            return new MyViewHolder(binding);
        }

        // 항목 구성: 항목이 x개면 x번 호출된다
        @Override
        public void onBindViewHolder(MyViewHolder viewHolder, int position) {
            String text = list.get(position);
            viewHolder.binding.itemTextView.setText(text);
        }

        @Override
        public int getItemCount() {
            return list.size();
        }
    }

    class MyItemDecoration extends RecyclerView.ItemDecoration {
        @Override
        public void getItemOffsets(Rect outRect, View view, RecyclerView parent, Recycle
            super.getItemOffsets(outRect, view, parent, state);
            int index = parent.getChildAdapterPosition(view) + 1;
            if (index % 3 == 0) {
                outRect.set(20, 20, 20, 60);
            } else {
                outRect.set(20, 20, 20, 20);
            }
            view.setBackgroundColor(0xFFECE9E9);
            ViewCompat.setElevation(view, 20.0f);
        }

        @Override
        public void onDrawOver(Canvas c, RecyclerView parent, RecyclerView.State state)
            super.onDrawOver(c, parent, state);
            int width = parent.getWidth();
            int height = parent.getHeight();
            Drawable dr = ResourcesCompat.getDrawable(getResources(), R.drawable.androic
            int drWidth = dr.getIntrinsicWidth();
            int drHeight = dr.getIntrinsicHeight();
            int left = width / 2 - drWidth / 2;
            int top = height / 2 - drHeight / 2;
            c.drawBitmap(BitmapFactory.decodeResource(getResources(), R.drawable.androic
        }
    }
}
```

Resources: 깡쌤의 안드로이드 프로그래밍