Android Note:

Internal Storage: Bộ nhớ RAM, thông tin lưu ở Internal Storage chỉ lưu trữ và sử dụng cho riêng từng ứng dụng, các ứng dụng khác không thể truy cập được. Khi gỡ bỏ ứng dụng thì các file ở đây cũng sẽ bị gỡ bỏ

External Storage: Bộ nhớ ngoài(dung lượng lưu trữ), file lưu ở đây không được bảo mật. Có 2 loại external storage là: lưu ở ổ cứng của điện thoại và lưu ở SD Card(thẻ nhớ)

I. Layout: Gồm các loại layout sau

- Framelayout , LinearLayout, RelativeLayout, TableLayout, GridLayout, ConstraintLayout

1. Framelayout: Luôn giữ các view bên trong ở góc trái màn hình và không thể thay đổi location hiển thị, Các view được thiết lập sau sẽ đè lên các View thiết lập trước.

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity" >

<TextView

android:id="@+id/textView"

android:layout\_width="300dp"

android:layout\_height="300dp"

android:background="@color/colorAccent"

android:text="TextView" />

<TextView

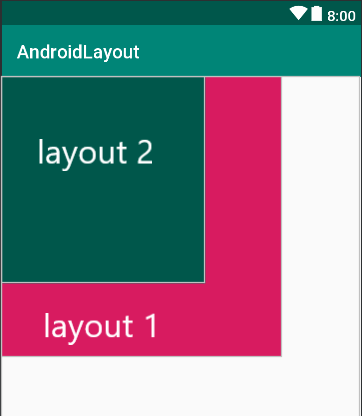
android:id="@+id/textView2"

android:layout\_width="218dp"

android:layout\_height="221dp"

android:background="@color/colorPrimaryDark" />

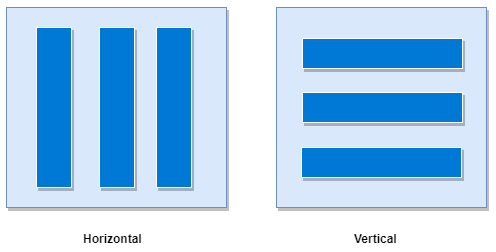
</FrameLayout>



2. LinearLayout: Bố trí các view theo chiều ngang hoặc chiều dọc, mặc định sẽ là chiều ngang.

Orientation = Vertical : Các view bên trong sẽ được xếp từ trên xuống.

Orientation = Horizontal: Các view bên trong sẽ được xếp trái qua phải.



<LinearLayout

xmlns:android1="http://schemas.android.com/apk/res/android"

android1:layout\_width="match\_parent"

android1:layout\_height="match\_parent"

android1:orientation="horizontal" >

<TextView

android1:layout\_width="80dp"

android1:layout\_height="80dp"

android1:text="1"

android1:textColor="#fff"

android1:textSize="15pt"

android1:textAlignment="center"

android1:textStyle="bold"

android1:background="@color/colorAccent" />

<TextView

android1:layout\_width="80dp"

android1:layout\_height="80dp"

android1:text="1"

android1:textColor="#fff"

android1:textSize="15pt"

android1:textAlignment="center"

android1:textStyle="bold"

android1:background="@color/colorPrimary" />

<TextView

android1:layout\_width="80dp"

android1:layout\_height="80dp"

android1:text="3"

android1:textColor="#fff"

android1:textSize="15pt"

android1:textAlignment="center"

android1:textStyle="bold"

android1:background="#8c0520" />

<TextView

android1:layout\_width="80dp"

android1:layout\_height="80dp"

android1:background="#efcd21"

android1:text="4"

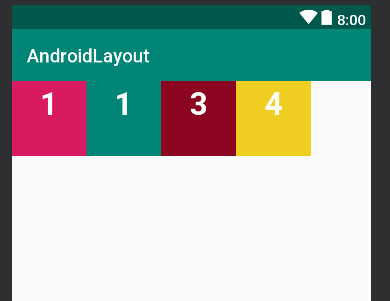
android1:textAlignment="center"

android1:textColor="#fff"

android1:textSize="15pt"

android1:textStyle="bold" />

</LinearLayout>



- android:weightSum="2": Setting tổng bố cục của item cha

- android:layout\_weight="{number}": Setting ở child item để chia bố cục dựa theo weightSum của item cha.

**Chú ý:** Cần setting android:layout\_height="0dp" nếu đang xếp từ trên xuống   
hoặc android:layout\_width="0dp" nếu đang xếp từ trái sang phải để có thể bố cục theo giá trị layout\_weight

3. RelativeLayout: Vị trí của mỗi view bên trong sẽ được xác định so với view khác hoặc so với thành phần cha của nó thông qua ID.

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView

android:layout\_width="150dp"

android:layout\_height="50dp"

android:text="1"

android:textColor="#fff"

android:id="@+id/hop1"

android:textSize="10pt"

android:textAlignment="center"

android:background="@color/colorAccent" />

<TextView

android:layout\_width="150dp"

android:layout\_height="50dp"

android:text="2"

android:textColor="#fff"

android:id="@+id/hop2"

android:textSize="10pt"

android:textAlignment="center"

android:layout\_toRightOf="@+id/hop1"

android:background="@color/colorPrimary" />

<TextView

android:layout\_width="150dp"

android:layout\_height="50dp"

android:text="3"

android:textColor="#fff"

android:id="@+id/hop3"

android:background="#8e0917"

android:textSize="10pt"

android:textAlignment="center"

android:layout\_below="@+id/hop1" />

<TextView

android:layout\_width="150dp"

android:layout\_height="50dp"

android:text="4"

android:textColor="#fff"

android:id="@+id/hop4"

android:textSize="10pt"

android:textAlignment="center"

android:background="#edcb25"

android:layout\_below="@+id/hop2"

android:layout\_toRightOf="@+id/hop3" />

</RelativeLayout>

4. TableLayout: Các view bên trong được xếp theo dạng table. Mỗi row là 1 TableRow , bên trong TableRow sẽ chứa các View.

<TableLayout android:layout\_height="match\_parent"

android:layout\_width="match\_parent"

xmlns:android="http://schemas.android.com/apk/res/android">

<TableRow>

<TextView

android:text="TableLayout Example"

android:background="#c98282"

android:gravity="center"/>

</TableRow>

<TableRow>

<Button android:text="B1 B1" />

<Button android:text="B2"/>

</TableRow>

<TableRow>

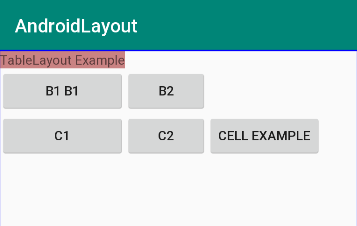
<Button android:text="C1" />

<Button android:text="C2" />

<Button android:text="Cell example" />

</TableRow>

</TableLayout>



5. GridLayout: Layout dạng lưới với số dòng và số cột ta quy định. Mỗi View bên trong sẽ nằm ở mỗi ô trong lưới này.

<GridLayout

xmlns:android1="http://schemas.android.com/apk/res/android"

android1:layout\_width="match\_parent"

android1:layout\_height="match\_parent"

android1:columnCount="2"

android1:rowCount="2"

>

<TextView

android1:layout\_width="80dp"

android1:layout\_height="80dp"

android1:text="1"

android1:textColor="#fff"

android1:textSize="15pt"

android1:textAlignment="center"

android1:textStyle="bold"

android1:background="@color/colorAccent" />

<TextView

android1:layout\_width="80dp"

android1:layout\_height="80dp"

android1:text="2"

android1:textColor="#fff"

android1:textSize="15pt"

android1:textAlignment="center"

android1:textStyle="bold"

android1:background="@color/colorPrimary" />

<TextView

android1:layout\_width="80dp"

android1:layout\_height="80dp"

android1:text="3"

android1:textColor="#fff"

android1:textSize="15pt"

android1:textAlignment="center"

android1:textStyle="bold"

android1:background="#8c0520" />

<TextView

android1:layout\_width="80dp"

android1:layout\_height="80dp"

android1:background="#efcd21"

android1:text="4"

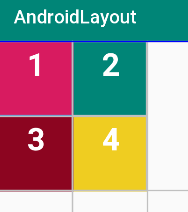
android1:textAlignment="center"

android1:textColor="#fff"

android1:textSize="15pt"

android1:textStyle="bold" />

</GridLayout>



6. ConstraintLayout:

II. Items:

- Button:

android:textAllCaps="true/false" : Setting tất cả chữ hoa(true)/chữ thường(false)

- EditText:

android:hint="{text}": Hiển thị gợi ý

android:drawableLeft="{image}", android:drawableRight="{image}" : Set image trong EditText

- ImageView:

android:scaleType="{setting}" : Set tỉ lệ của image so với khung của image (khung của image không phải là khung activity)

- ImageButton:

android:background="@null" : Bỏ đi phần background của button, chỉ giữ lại image

- RadioButton: Setting các RadioButton cùng 1 Group: kéo thả RadioGroup -> kéo thả RadioButton

<RadioGroup>

<RadioButton/>

<RadioButton/>

</RadioGroup>

**Inflate:**

Instantiates a layout XML file into its corresponding {**@link** android.view.View} objects.

Nó dùng để inflate (bơm, nạp…) cái phần XML (Phần giao diện) vào View.

Giả sử bạn muốn tạo một cái view, thì để biết cái view đó đang chịu trách nhiệm phần giao diện nào (XML) thì bạn phải inflate cái XML đó vào view.

Một trường hợp rất thường gặp là khi bạn hiện thực CustomAdapter cho ListView. Trong method getView(), bạn cần phải inflate cái XML (ở đây là layout cho item) vào biến convertView thì ListView mới biết là phần giao diện nào của item cần được thể hiện ra màn hình.

Lớp LayoutInflater này rất hữu ích nếu bạn dùng để tạo giao diện linh hoạt (có thể thêm bớt các view vào root view) chỉ bằng code thuần .

Drawable:

1. shape: hình dạng  
Trong thẻ shape cần có thuộc tính android:shape để khai báo hình dạng

- solid: mang giá trị color

- size: kích thước(height, width)

- corners: bo góc (radius)

- padding: padding item

- stroke: tạo viền (color: màu viền, width: độ rộng của viền, dashGap(chiều dài nét đứt),dashWidth(chiều dài nét liền): tạo nét đứt của viền)

- grandient: tạo độ đậm nhạt theo góc nghiêng(angle: góc(tối thiểu 45), startColor: màu bắt đầu, centerColor: màu ở giữa, endColor: màu kết thúc)

2. layer-list: xếp chồng các layer lên nhau

- item: là layer(top: cách trên, left: cách trái,…)

+ bitmap: thẻ hình ảnh(src: đường dẫn hình ảnh, gravity: trọng tâm hình ảnh)

3. selector: xử lý trạng thái select

- item: là item(drawable: thẻ drawable, state\_pressed: trạng thái nhấn vào, color: màu chữ)

Note: vì selector duyệt item từ trên xuống nên item mang giá trị default phải để cuối cùng.

4. clip: cắt hình (drawable: hình, clipOrientation: hướng cắt”horizontal/vertical”, gravity: cắt từ vị trí nào)

Note: code behind cần xử lý setImageLevel để có thể load được clip

**Menu**: 2 dạng menu: menu của hệ thống và menu custom

app:showAsAction="always": setting để luôn show menu

**Dialog**: 2 dạng dialog: dialog của hệ thống và dialog custom

Date time:

- Giá trị month được tính từ 0-11 tháng

**Intent**: Có 2 dạng intent(implicit và explicit)  
- implicit Intent: intent có parameter xác định rõ activity để redirect tới(activity khác)  
- explicit Intent: intent không có parameter xác định rõ activity để redirect tới(intent camera, phone, message,…)

**Shared Preferences**: File để lưu lại 1 số thông tin cơ bản của Application, thông tin này sẽ tồn tại cùng với Application, khi user gỡ App thì thông tin này cũng sẽ mất.  
- Có thể sử dụng trong bài toán lưu thông tin nhằm xác định là App mới cài đặt(để hiển thị màn hình welcome ban đầu)  
- Share preferences không có support việc lưu object, chỉ có thể lưu listString, String, int, long, boolean. Để lưu object sẽ thực hiện convert object->Json->String rồi thực hiện lưu, và đọc từ String->Json->object

**Animation**:

- fillAfter="false/true": không giữ lại hoặc giữ lại trạng thái cuối cùng của animation. Default là false

- android:interpolator="@android:anim/linear\_interpolator": Tạo độ mượt hơn khi loop animation

1. Alpha: Hiệu ứng làm mờ

- fromAlpha: Độ mờ bắt đầu  
- toAlpha: Độ mờ kết thúc  
- duration: Thời gian chuyển từ fromAlpha sang toAlpha  
- repeatCount="số lần lặp lại": nếu setting repeatCount=" infinite" sẽ lặp lại vô tận  
- repeatMode=" restart/reverse": hiệu ứng chuyển từ fromAlpha sang toAlpha. Default là restart

2. Rotate: Hiệu ứng xoay

- fromDegrees="0": Góc xoay bắt đầu  
- toDegrees="90": Góc xoay kết thúc  
- pivotX="50%": Vị trí xoay trục tung  
- pivotY="50%": Vị trí xoay trục hoành  
- duration: Thời gian chuyển từ fromDegrees sang toDegrees  
- repeatCount="số lần lặp lại": nếu setting repeatCount=" infinite" sẽ lặp lại vô tận  
- repeatMode=" restart/reverse": hiệu ứng chuyển từ fromDegrees sang toDegrees. Default là restart

3. Scale: Setting phóng to thu nhỏ

- fromXScale="1.0": Vị trí phóng to/ thu nhỏ theo trục X  
- toXScale="2.0": Vị trí phóng to/ thu nhỏ theo trục X  
- fromYScale="1.0": Vị trí phóng to/ thu nhỏ theo trục Y  
- toYScale="2.0": Vị trí phóng to/ thu nhỏ theo trục Y  
- pivotX="50%": Vị trí phóng to/ thu nhỏ trục X  
- pivotY="5%": Vị trí phóng to/ thu nhỏ trục Y  
- repeatCount="số lần lặp lại": nếu setting repeatCount=" infinite" sẽ lặp lại vô tận   
- repeatMode==" restart/reverse": hiệu ứng chuyển từ fromDegrees sang toDegrees. Default là restart  
- duration="2000": Thời gian chuyển từ phóng to/thu nhỏ

4. Translate: Setting di chuyển (nếu có n setting translate sẽ được xử lý theo thứ tự trên xuống)

- fromXDelta="0": Vị trí di chuyển bắt đầu theo trục X  
- toXDelta="500": Vị trí di chuyển kết thúc theo trục X  
- fromYDelta="0": Vị trí di chuyển bắt đầu theo trục Y  
- toYDelta ="500": Vị trí di chuyển kết thúc theo trục Y  
- repeatCount="số lần lặp lại": nếu setting repeatCount=" infinite" sẽ lặp lại vô tận   
- repeatMode==" restart/reverse": hiệu ứng chuyển từ fromDegrees sang toDegrees. Default là restart  
- duration="2000": Thời gian di chuyển  
- startOffset="6000": thời gian chờ để start translate(dùng thuộc tính này để timing setting thứ tự thực hiện cho các translate)

Volley: Sử dụng để xử lý data trên internet

Mp4: Cần sử dụng Video View để có thể load

Ứng dụng nghe nhạc:

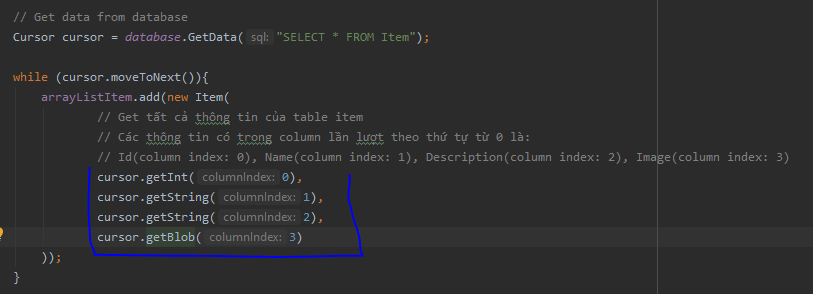
Play: [if pause] start music, Button = PAUSE ----// ---- Pause: [if start] pause music, Button = PLAY  
Next: next music, Button = PAUSE  
Previous: previous music, Button = PAUSE  
Stop: stop music, Button = PLAY

Q/A:

Câu hỏi 1:  
Tại sao phải Re-init media player ở button stop

Câu hỏi 2:  
2 xử lý này khác nhau không và khác nhau như nào?  
Sử dụng variable jobName truyền vào để insert data  
database.QueryData("INSERT INTO Job VALUES(null, '" + jobName + "')");

Câu hỏi 3: Tại sao chỗ này con trỏ cursor có thể hiểu được là get Blob ?



SQLite: extends từ SQLiteOpenHelper

Nếu sử dụng method POST để add data vào database thì sẽ override method getParams (type Map<String, String>)để add param để push lên server cùng thời điểm khởi tạo request.

Fragment:   
tạo bằng kéo thả xml  
tạo bằng code:  
Note: fragmentTransaction chỉ thực hiện commit 1 lần  
Khi chỉ 1 fragment thì dùng fragmentTransaction.add, nếu từ 2 fragment trở lên thì dùng fragmentTransaction.replace

Trong fragment thì không thể findViewById như ở Activity được  
Sử dụng getActivity() để có thể get được parent Activity nơi mà Fragment đang tồn tại

Fragment có thể work với ListView theo cách thông thường hoặc theo ListView đặt biệt dành riêng cho Fragment. Nếu sử dụng ListView đặt biệt dành riêng cho fragment thì phải extends ListFragment và ListView id phải sử dụng @android:id/list.  
Q/A : Id tự đặt và id lấy trong android khác nhau như nào?  
Ví dụ: ListView tự đặt id và ListView sử dụng @android:id/list

**RecyclerView**: dạng danh sách có thể chia hang cột dễ dàng.  
Sử dụng **RecyclerView.setLayoutManager(LinearLayoutManager)** để setting format hiển thị của recyclerView là theo dạng LinearLayout.  
Sử dụng **RecyclerView.setAdapter(Adapter)** để thực hiện setting data adapter vào recyclerView  
Sử dụng **RecyclerView.ItemDecoration** để tạo line giữa các item trong list với nhau.

**Adapter**:  
Tạo viewHolder để xử lý giữ chỗ trong quá trình load/reload row\_item. ViewHolder sẽ Mapping layout của row\_item

**ViewPager**:  
Cơ chế hoạt động, khi open Fragment hiện tại thì ViewPager đã load trước Fragment trước và sau kế nó.  
Default là load trước là 1, chúng ta có thể sử dụng ViewPager.setOffscreenPageLimit(number), với number là số load trước, để thay đổi số load trước.  
Tham khảo: <https://viblo.asia/p/viewpager-giu-trang-thai-cua-fragment-4P8560DRZY3>  
  
**Fragment**:  
- Sử dụng **FragmentTransaction.add**, **FragmentTransaction.replace** để add hoặc replace Fragment lên View  
- Sử dụng **FragmentTransaction.addToBackStack** để add Fragment đó vào backstack để khi thực hiện **getFragmentManager.popBackStack** thì sẽ có thể back được về Fragment trước đó được.  
- Xử lý load lại **Fragment**: vì mỗi lần next Fragment thì thông tin Fragment cũ sẽ bị delete nên cần phải load lại,   
Note: Nên xử lý reload data tại thời điểm nextpage chứ không xử lý ở onResume để tránh trường hợp khi user click Home sau đó quay trở lại Fragment thì cũng bị reload không mong muốn.

**NestedScrollView:** Là 1 layout cho phép scroll nếu View quá dài so với màn hình **Bottom sheet:**- Có 2 loại button sheet: **Persistent Bottom sheet** và **Modal Bottom sheet**- Sử dụng BottomSheetBehavior.setState(BottomSheetBehavior.STATE\_EXPANDED) để setting expand bottomsheet hoặc BottomSheetBehavior.setState(BottomSheetBehavior.STATE\_COLLAPSED) để setting collap bottomsheet  
- Sử dụng ButtonSheetBehavior.addBottomSheetCallback: để check StateChanged mà xử lý của bottomsheet **Persistent Bottom sheet:** khi show bottom sheet thì phần màn hình còn lại vẫn work được  
(Tạo layout cho bottomsheet, tạo layout cho phần còn lại -> include 2 layout vào main layout)  
**Modal Bottom sheet: (BottomSheetDialog** và **BottomSheetDialogFragment)** khi show bottom sheet thì phần màn hình còn lại bị disable  
(Tạo layout cho bottomsheet, thực hiện show bottomsheet trên main layout) **BottomSheetDialog:**- Sử dụng **BottomSheetDialog.setContentView(View dialog)** để thực hiện setting bottomsheet  
- Sử dụng **BottomSheetDialog.setCancelable(false)** để không cho user thao tác bên ngoài hoặc click nút back dẫn tới close bottomsheet  
- Sử dụng **BottomSheetDialog.dismiss)** để close bottomsheet  
**BottomSheetDialogFragment:**- Sử dụng **BottomSheetDialog.setContentView(View dialog)** để thực hiện setting bottomsheet  
- Sử dụng **BottomSheetDialog.setCancelable(false)** để không cho user thao tác bên ngoài hoặc click nút back dẫn tới close bottomsheet  
- Sử dụng **BottomSheetDialog.dismiss)** để close bottomsheet **SearchView Filter:**- s **Gson:**  
- Sử dụng **gson** để xử lý với data Json <-> Object  
- Sử dụng **gson.toJson**: convert **object** -> **Json**  
Note: Cần đặt tên name giống với key ở Json file để khi generate data từ object sang json sẽ đúng. Trong một số trường hợp không thể đặt tên name giống được(vì cú pháp synctax của java code) thì sẽ sử dụng annotation để setting: @SerializeName(“name”)  
  
**Retrofit call API**: (trong retrofit có hỗ trợ gson, nên cần include libs của retrofit và gson cùng version)

**Room database:**  
- Tạo @Entity(tableName = "user"): Entity  
- Tạo @Dao: xử lý truy vấn tới database  
- Tạo @Database(entities = User.class, version = 1): Tạo database, Nếu có sự thay đổi Entity trong database thì cần phải thực hiện migrate

Cấu trúc của Reactive Programming: Gồm 2 thành phần cơ bản là Observable và Subscriber

**Observable**  
Là phần sẽ tạo ra các sự kiện. Bạn sẽ lắng nghe các sự kiện này để có hành động thích hợp. Giờ đây sẽ không còn khái niệm đợi chờ một tác vụ thực hiện xong rồi lần lượt tới các tác vụ khác.

**Observer**  
Là những đoạn code sẽ lắng nghe Observable. Nó nhận dữ liệu từ Observable phát ra. Bạn sẽ cần phải override lại những methods để Observer làm việc.  
  
**onSubscribe**(): Method được gọi khi một Observer đăng ký lắng nghe một Observable.

**onNext**(): Method này được gọi khi Observable bắt đầu bắn dữ liệu.

**onError**(): Nếu có bất kỳ lỗi lầm gì thì hàm onError() sẽ được gọi tên.

**onComplete**(): Được gọi khi một Observable hoàn thành tất cả các tác vụ.

**Libs thông dụng:**  
**CircleImageView**: libs đã xử lý sẵn Image theo phong cách của màn hình detail  
<https://github.com/hdodenhof/CircleImageView>  
  
**RecyclerView**: libs về list data, thường sử dụng hiển thị danh sách  
<https://developer.android.com/jetpack/androidx/releases/recyclerview>  
  
**Gson**: libs về xử lý serialization/deserialization object  
<https://github.com/google/gson>