TP 10 : CoordinatorLayout

Introduction :

Qu'est-ce qu'un CoordinatorLayout ?

Ce Layout nous permet d'ajouter de nouvelles dépendances entre plusieurs vues adjacentes. On peut leur affecter un comportement et de ce fait les vues peuvent par exemple se déplacer ensemble. C'est un Layout qui fournit un niveau de contrôle supplémentaire sur les événements entre les vues enfant.

A quoi ça sert ?

Comme son nom l'indique, le but d'un CoordinatorLayout est de coordonnée un ensemble de vues. De ce fait, les différents vues sont coordonnées les unes par rapport aux autres. Le CoordinatorLayout est majoritairement utilisé en tant que conteneur pour une interaction spécifique avec une ou plusieurs vues enfant.

Objectif du TP :

Dans ce TP, on va montrer l'utilité d'un CoordinatorLayout dans une première partie en créant un simple « Floating Action Button » qui va déclencher une « Snackbar ». Puis dans une deuxième partie nous ferons le lien avec le CollapsingToolbarLayout.

Déroulement du TP :

En 2015, Google a introduit une nouvelle bibliothèque de design (Design Support Library), dans laquelle est arrivé le CoordinatorLayout, et cela fonctionne pour l'API 7 et plus.

Première partie :

https://drive.google.com/file/d/0B1C0qUnbjNjueTM2RzJmZlN3VTA/view?usp=sharing

On va commencer par créer un nouveau projet Android en lui spécifiant « Minimum SDK API 9: Android 2.3 » et « Empty Activity ». Pour commencer, il ne faut pas oublier d'ajouter dans le gradle la dépendance pour le Support Design Library :

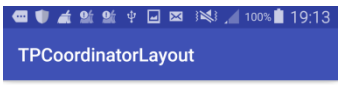
```
compile 'com.android.support:design:22.2.0'
```

Ensuite on vient chercher dans la palette un FloatingActionButton que l'on vient mettre en bas à droite de notre RelativeLayout :

A ce FloatingActionButton on vient lui définir un « ID : fab » et lui associer une fonction au clique « onClick : execute ». Puis dans MainActivity.java on vient y mettre le code nécessaire pour déclencher la Snackbar sur le clique :

```
public class MainActivity extends AppCompatActivity {

    private CoordinatorLayout activity_main;
    private FloatingActionButton fab;

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

        activity_main = (CoordinatorLayout) findViewById(R.id.activity_main);
        fab = (FloatingActionButton) findViewById(R.id.fab);
    }

    public void execute(View v){
        Snackbar.make(activity_main, "Hello !", Snackbar.LENGTH_SHORT).show();
    }
}
```

Lorsqu'on exécute ce code, on remarque lors du clique la Snackbar qui vient se mettre au-dessus du FloatActionButton :



Pour y remédier, il faut utiliser un CoordinatorLayout au lieu du RelativeLayout pour que le bouton soit coordonné par rapport à la Snackbar :

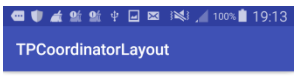
```
<android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

Et on n'oublie pas de mettre le FloatActionButton en bas à droite de l'écran étant donné que l'on est plus dans un RelativeLayout :

```
android:layout_gravity="end|bottom"/>
```

Enfin, dans MainActivity.java on effectue les changements nécessaires :

```
private CoordinatorLayout activity_main;

activity_main = (CoordinatorLayout) findViewById(R.id.activity_main);
```



Deuxième partie :

https://drive.google.com/file/d/0B1C0qUnbjNjuc1VLQS1IZ0RkWU0/view?usp=sharing

Dans cette partie, nous ferons le lien entre le CoordinatorLayout et le CollapsingToolbarLayout qui permet de créer une barre d'action (Toolbar) réductible. Par exemple on peut avoir une image qui se réduira au fur et à mesure que le contenu en dessous scroll pour se transformer en Toolbar :

On commence par définir un AppBarLayout qui est un LinearLayout vertical qui implémente beaucoup de concept du material design pour les barres d'actions et on en aura besoin pour gérer le scroll. Chaque composant (élément fils) de l'AppBarLayout doit indiquer la manière dont il souhaite se comporter lors du scroll et cela soit à l'aide de la méthode « setScrollFlags » ou l'attribut « app:layout_scrollFlags ». On va donc l'utiliser avec le CoordinatorLayout pour avoir les comportements souhaités. En plus de ça, on aura besoin d'un élément scrollable (ListView, NestedScrollView ou RecyclerView) qui sera déclaré dans le CoordinatorLayout mais à l'extérieur de notre AppBarLayout. Dans notre cas, on utilisera une NestedScrollView avec un simple TextView dedans.

```
<android.support.design.widget.AppBarLayout
    android:id="@+id/appbar"
    android:layout_height="192dp"
    android:layout_width="match_parent">
```

Ensuite, à l'intérieur de cet AppBarLayout on va créer un CollapsingToolbarLayout qui se trouve être un « wrapper » pour les Toolbars. Comme dit plus haut, on va l'utiliser en combinaison du AppBarLayout pour implémenter l'effet de toolbar qui se réduit.

Il possède plusieurs attributs, ici nous en utiliserons deux :

- app:contentScrim : Permet de spécifier le drawable (image, background) utilisée par la Toolbar quand il est au top de la vue.
- app:scrollFlags : Cet attribut permet de spécifier le comportement du composant quand l'élément rentre dans le scroll et quitte le scroll. Il possède deux valeurs : d'abord « scroll » (indispensable pour les éléments qui doivent scroller en dehors de l'écran, les vues qui n'utilisent pas cette valeur reste fixées en haut de l'écran), et enterAlways (Assure que tout scroll vers le bas rend cet élément visible).

```
<android.support.design.widget.CollapsingToolbarLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:toolbarId="@+id/toolbar"
    app:layout_scrollFlags="scroll|enterAlways"
    app:layout_scrollInterpolator="@android:anim/decelerate_interpolator"
    app:contentScrim="?attr/colorPrimary">
```

Ensuite, on va déclarer ce qu'on veut mettre dans notre AppBarLayout. Pour nous, ce sera une Image et une Toolbar. L'attribut qu'il faut prendre en compte ici est « app:layout_collapseMode ». Cet attribut permet de spécifier aux différents composants leur comportement lors du scroll et de qu'elle façon ils se réduisent.

Pour l'image, l'attribut peut prendre comme valeur « parallax », de ce fait l'image se réduira en utilisant un effet parallax lors du scroll. De plus, un attribut optionnel peut être utilisé pour indiquer la vitesse du parallax (app:layout_collapseParallaxMultiplier).

```
<ImageView
    android:id="@+id/app_bar_image"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:src="@android:drawable/sym_def_app_icon"
    android:scaleType="centerCrop"
    app:layout_collapseMode="parallax"/>
```

Dans le cas de la Toolbar, cet élément aura pour valeur « pin », ce qui indiquera que la Toolbar sera fixée en haut de la vue lors du scroll vers le haut.

Ce qui permet de donner les deux effets voulus :

- Disparition de l'image au fur et à mesure du scroll (vers le haut) et cela en utilisant un effet parallax.
- Apparition de la Toolbar en haut de la vue.

```
<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_height="?attr/actionBarSize"
    android:layout_width="match_parent">
</android.support.v7.widget.Toolbar>
```

Pour finir, on déclare une NestedScrollView (ou RecyclerView si on veut) dans le CoordinatorLayout mais seulement à l'extérieur de l'AppBarLayout. En lui affectant un attribut «app:layout_behavior» qui possède une valeur «@string/appbar_scrolling_view_behavior», cela permet au CoordinatorLayout de réagir au scroll de la NestedScrollView. De ce fait, lorsque l'on scroll la NestedScrollView on affecte le AppBarLayout et son contenu. On peut aussi venir ajouter un FloatingActionButton si on veut.

```
<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingViewBehavior">

    <TextView
        android:text="Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris laoreet,
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView10" />
</android.support.v4.widget.NestedScrollView>
```

Au final, cela donne une image qui se réduira au fur et à mesure que le contenu en dessous scroll :