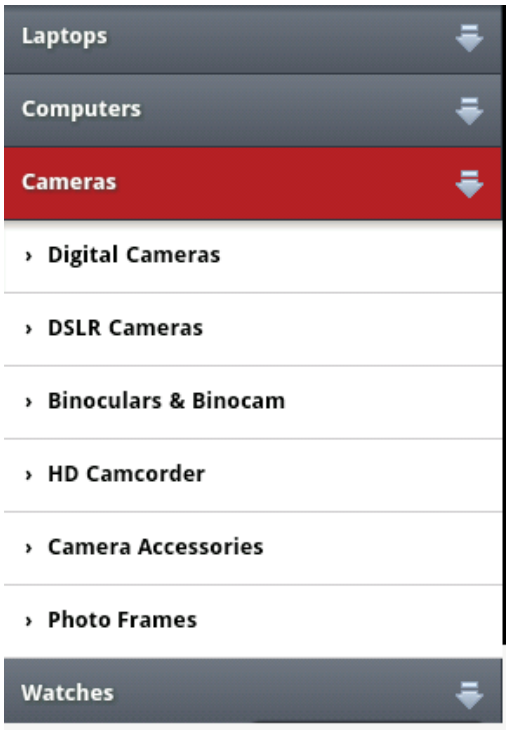
1- Qu'est-ce que ListView ?

ListView est un **viewgroup** qui affiche des éléments selon une liste et peut être déplacé verticalement. **Listview** est une vue importante et est largement utilisé dans les applications Android.

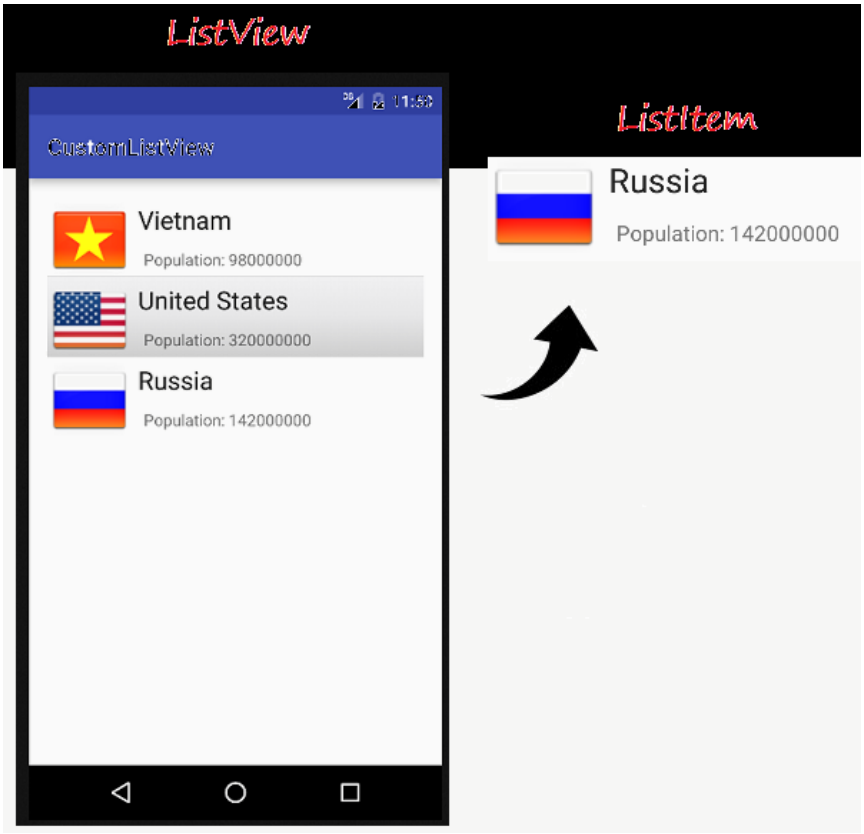
Un simple exemple de **ListView** est votre carnet de contacts, où vous avez une liste de vos contacts affichés dans un **ListView**.

En plus de ListView, **Android** vous fournit un autre view similaire qui est **ExpandableListView**.

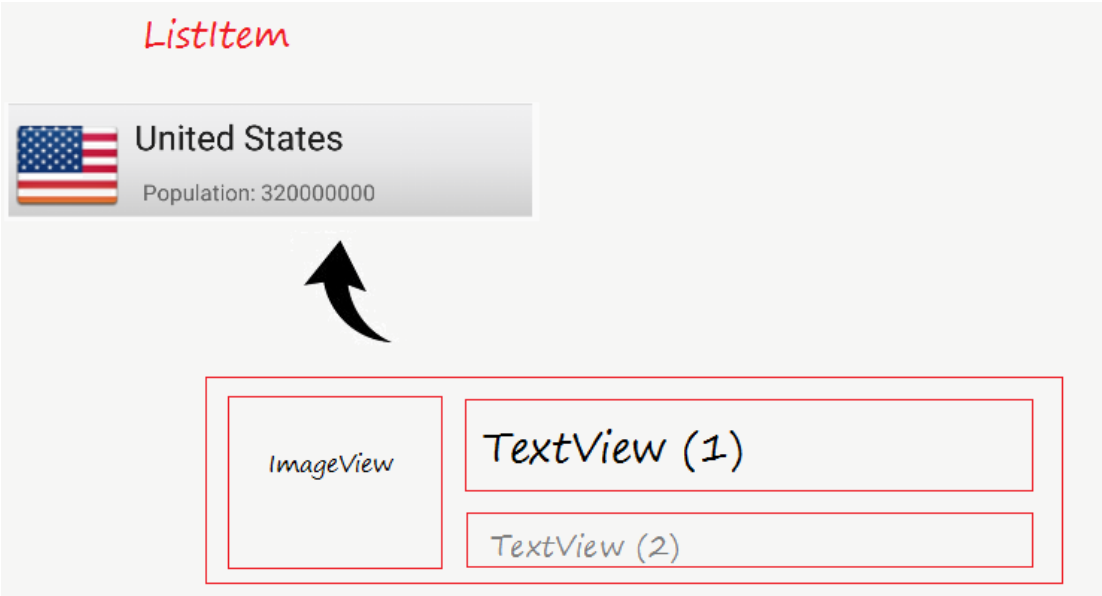


1.1- ListItem

Un ListView est fait à partir d'un groupe de **ListItem**. **ListItem** est une ligne individuelle dans **listview** où les données seront affichées. Toutes les données dans **listview** sont affichées uniquement via **listItem**. On peut considérer **Listview** comme groupe défilable de **ListItems**.



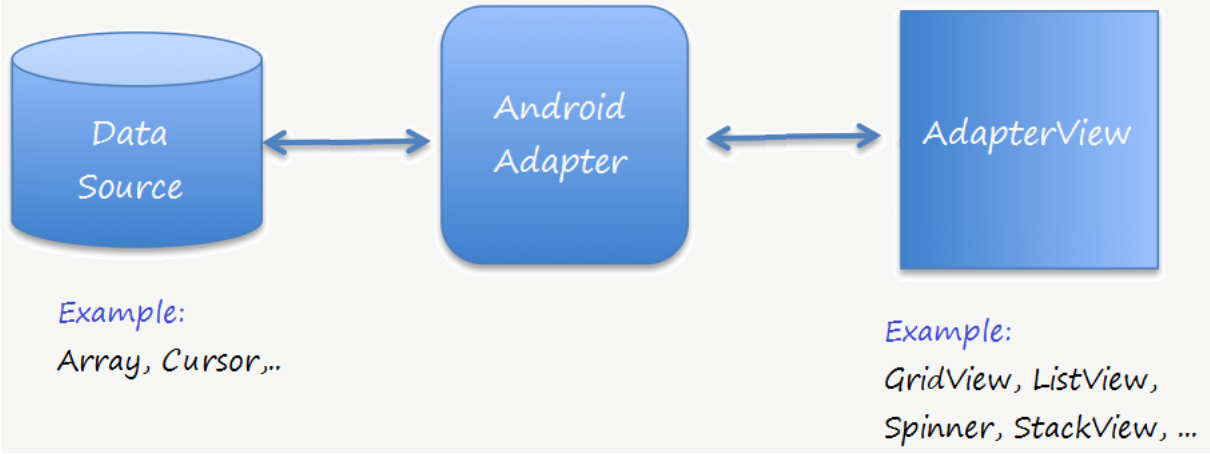
Un ListItem est une pièce de l'interface qui peut être créée par un nombre de View.

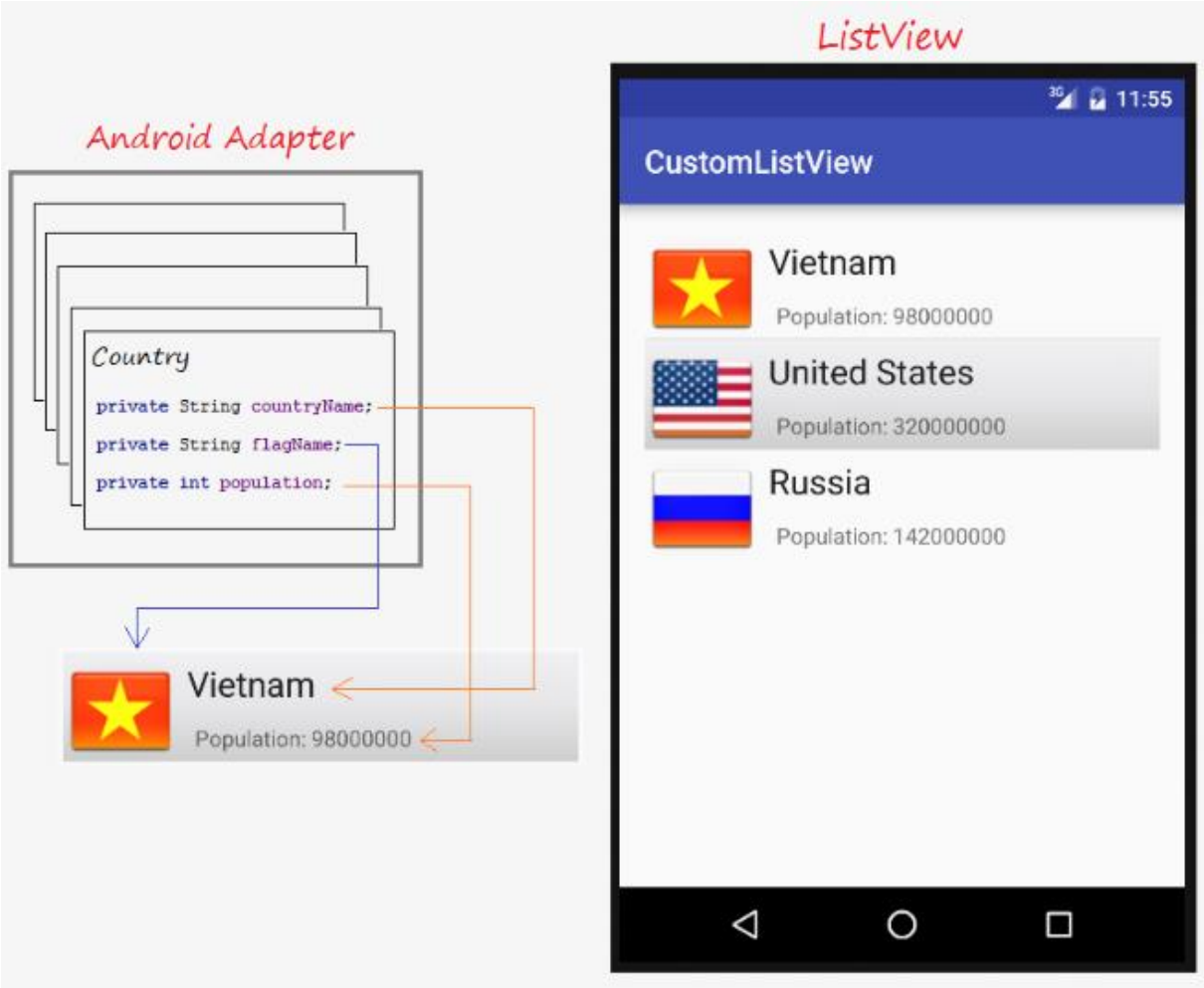


1.2- Adapter (L'adaptateur)

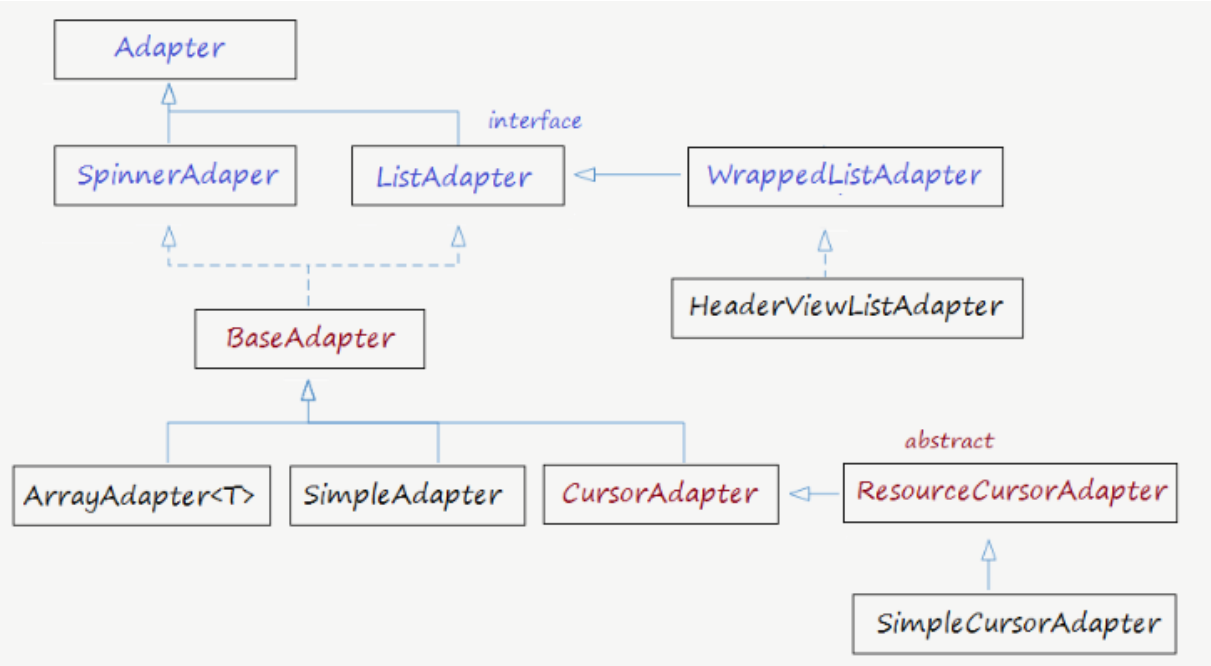
Android Adapter (L'adaptateur) est un pont entre des View (par exemple comme ListView) et les données sous-jacentes pour ce **View**. Un **Adapter** gère des données et adapte les données dans les lignes individuelles (**ListItem**) du view.

Vous pouvez lier des **Adapter** avec **Android ListView** via la méthode ***setAdapter***. Maintenant, nous allons voir comment **Adapter** fonctionne à l'aide de l'image ci-dessous.



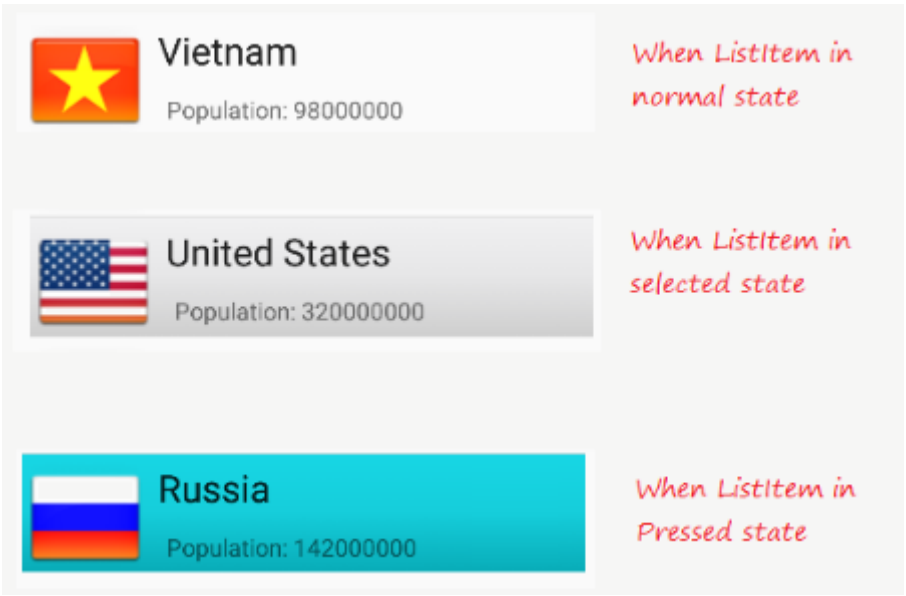


Il existe plusieurs **View** ont eu besoin d'**Android Adapter** en vue de gérer des données affichées, ces **View** sont les sous-classes de la classe **AdapterView**, vous pouvez voir l'illustration suivante :



1.3- ListView Selector

Pour rendre **ListView** plus beau, vous devez personnaliser les effets, tels que modifier la couleur d'arrière-plan de **ListItem** lorsque le curseur se déplace dessus ou modifier la couleur d'arrière-plan lorsqu'il est sélectionné. Vous pouvez voir un exemple pour personnaliser le **ListView Selector** à la fin de ce document.



2- ListView basic utilisant ArrayAdapter

2.1- ArrayAdapter

ArrayAdapter a utilisé à afficher le **ListView** avec des **ListItem** simple, **ListItem** peut être fabriqué à partir seulement un **TextView**, **CheckedTextView**, **EditText**, etc. ...
Au cas où vous voulez avoir un **ListView** avec **ListItem** plus compliqué, vous pouvez créer manuellement un **Adapter** personnalisé.

Travail à réaliser :

En utilisant les différents éléments vus plus haut (**ListView**, **ListItem**, **ArrayAdpter**, etc. …) concevoir une application mobile qui affiche les 22 pays arabes avec leur population. Lancer l'émulateur pour tester votre code.

Remarque : Pour plus de détails sur les différents points vus plus haut consulter :

https://o7planning.org/fr/10435/tutoriel-android-listview

I. Fragments

I.1 Définition et Utilité

Un fragment représente un comportement ou une portion d'interface utilisateur dans une activité. Il est possible de combiner plusieurs fragments dans une seule activité pour construire une interface à plusieurs panneaux, et réutiliser un fragment dans plusieurs activités.

Un fragment a son propre cycle de vie, reçoit ses propres entrées et peut être ajouté, modifié ou supprimé de manière dynamique. Il est toujours inclus dans une activité, et son cycle de vie est directement affecté par celui de l'activité qui le contient. Tant que l'activité conteneur est en état d'exécution, le fragment peut être manipulé de manière indépendante (ajouté, modifié ou supprimé).

Mais si l'activité est détruite, le fragment l'est aussi.

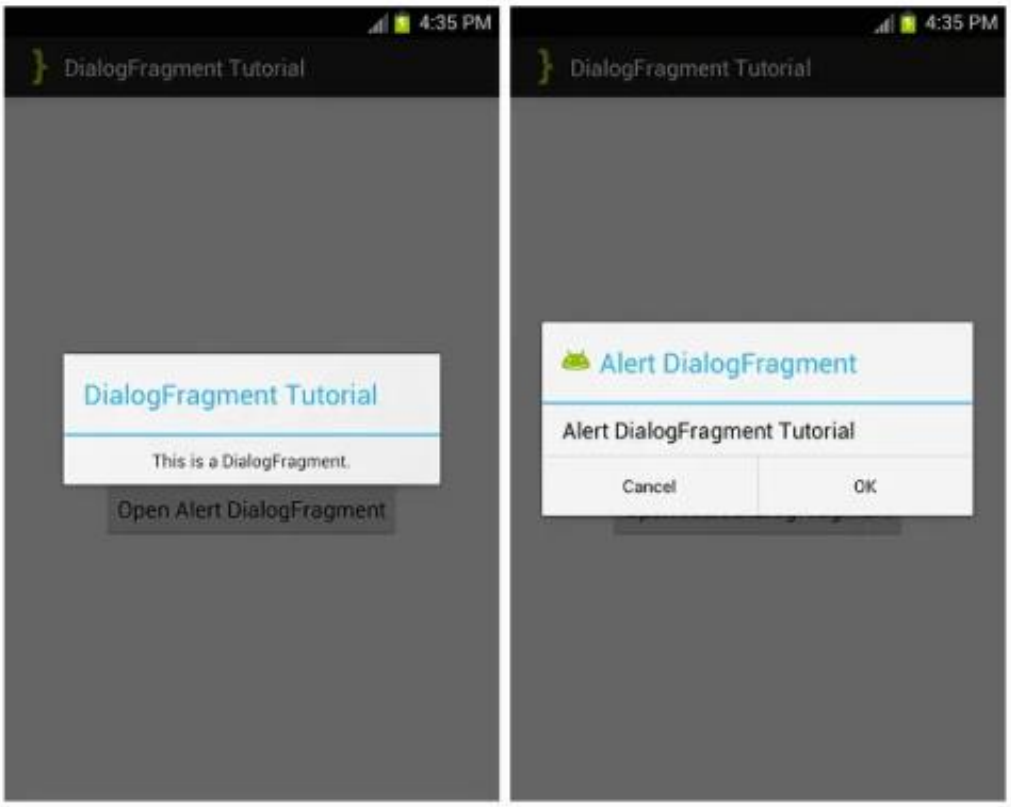
A la base, les fragments ont été créés pour supporter la conception d'interfaces graphiques flexibles sur des écrans de tailles différentes, et permettre ainsi aux appareils dont la taille d'écran est plus importante (comme les tablettes) de profiter de cet espace pour combiner et interchanger des composants graphiques.

I.2 Création d'un Fragment

Pour créer un fragment, il faut créer une classe qui hérite de **android.app.Fragment**, ou une de ses sous-classes. Parmi ces sous-classes on peut citer :

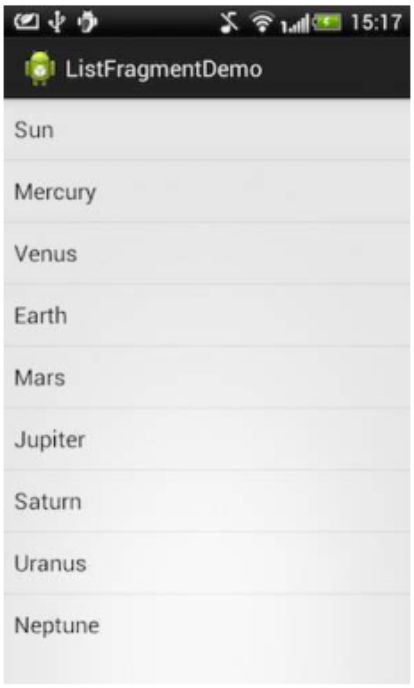
- **DialogFragment**

Affiche une boîte de dialogue flottante au lieu d'utiliser des méthodes dans votre activité, car il peut être rajouté à la pile des fragments de l'activité



- **ListFragment**

Affiche une liste d'éléments gérés par un adaptateur, et fournit ainsi plusieurs méthodes pour gérer une **listView**.



- **PreferenceFragment**

Affiche une hiérarchie d'objets Préférence comme liste. Permet de créer une activité settings pour votre application.

Pour créer un fragment, il faut suivre les étapes suivantes :

1. Créer l'interface du fragment

- Commencer par définir une interface pour le fragment dans un fichier **Layout** séparé
- Implémenter la méthode de transition **onCreateView** pour construire l'interface du fragment
  - Elle doit retourner un objet **View** représentant la racine du fragment
    Utiliser un **LayoutInflater** pour construire la hiérarchie d'objets graphiques à partir des ressources **XML**

2. Ajouter le fragment à une activité

Le fragment peut être ajouté de manière statique ou dynamique à l'activité.

- Statique : Déclarer le fragment dans le fichier **layout** XML de l'activité, grâce à l'élément XML <fragment>. Pour cela, indiquer comme attribut **android:name** la classe Java du fragment à instancier dans le **layout**
- Dynamique : Insérer dans le code de l'activité le fragment créé dans un conteneur (**ViewGroup**) existant dans l'interface. Pour cela, un **FragmentManager** est utilisé.

Un **FragmentManager** permet de gérer les fragments d'une activité particulière. Pour l'utiliser, il suffit d'appeler directement **getFragmentManager** à partir de n'importe quel emplacement de votre activité.

Le **FragmentManager** permet de :

• Manipuler les fragments existants dans votre application grâce aux méthodes **findFragmentById()** ou **findFragmentByTag()**

• Gérer la pile de retour (**BackStack**), accessible via le bouton Back de l'appareil

- Dépiler un fragment de la pile, avec **popBackStack()**

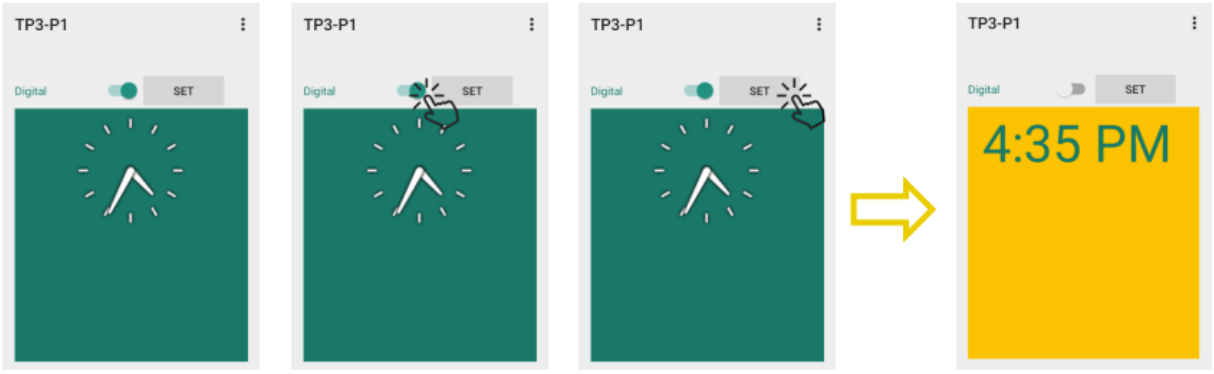
o Associer un **Listener** aux changements de cette pile avec **addOnBackStackChangedListener()**

La manipulation des fragments par le **FragmentManager** est possible grâce à un ensemble de **FragmentTransactions**. Une transaction est une opération sur des fragments (ajout, suppression, remplacement…) dans une activité, en réponse à une interaction utilisateur. Une transaction peut être stockée dans la pile de retour de l'activité, pour permettre à l'utilisateur de revenir en arrière, grâce à la méthode **addToBackStack()**.

II. **Exercice 1** : Fragments

II.1 Objectif

L'objectif de cet exercice est de montrer comment créer et interchanger des fragments dans une application Android. L'objectif est d'obtenir le résultat suivant :



II.2 Création des fragments

Pour créer les fragments, commencer par :

- Créer une application avec une simple activité vide.
- Créer le **layout** pour le premier fragment. Pour cela, créer un fichier frag_digital.xml sous le répertoire **layout**
- Y insérer un objet **AnalogClock**, et utiliser la couleur d'arrière-plan de votre choix
- Faire de même pour créer un fragment frag_numeric avec une horloge de type **TextClock**
- Créer une classe FragmentClock qui étend la classe Fragment.
- La première chose à faire est d'implémenter la méthode onCreateView qui représente le comportement du fragment dès son apparition. Dans cette méthode, nous allons indiquer quel fragment charger. Pour charger le fragment frag_digital, par exemple, la méthode onCreateView doit retourner :

Inflater.inflate(R.layout.*frag_digital*, container **false**);

II.3 Création de l'activité principale

Pour créer l'activité principale :

- Commencer par créer son interface. Pour cela, commencer par insérer un switch et un bouton côte à côte. Le bouton définit comme méthode onClick la fonction : setTime
- Créer une balise fragment qui va contenir le fragment en question.

```
<fragment
        android :layout_width=”match_parent”
        android :layout_height=”421dp”
        android :id=”@+id/fragment”
        android:name=”tp4.fragmentClock”
/>
```

L’objectif étant de cliquer sur le bouton Set, puis d’afficher l’horloge digitale sur le switch est activé, et l’horloge numérique s’il est désactivé. Il faut donc maintenant implémenter la fonction setTime.

```
public void  setTime(View v){
        Fragment f = new FragmentClock ();
        Bundle b = new Bindle ();
        b.putBoolean(“digitalOK”, s.isChecked());
        f.setArguments(b);

        FragmentManager fm=getFragmentManager(),
        FrgamentTransaction tr = fm.beginTransaction();
        tr.replace(R.id.fragment,f) ;
        tr.commit() ;
}
```

L’objet Bundlepermet d’envoyer un ensemble de paramètres, sous format clef/valeur, au fragment.
En l’occurrence, dans notre cas, nous allons lui envoyer l’état du switch (checked ou unchecked), dans le paramètre booléen digitalOK.
Le fragment manager va se charger de placer le fragment à l’endroit qui lui est associé, soit dans la balise fragment.
Pour définir quel fragment charger, modifier le code de la classe FragmentClock comme suit :

```
public class FragmentClock extends Fragment {
        boolean digitalOK = true;

        @override
        public void setArguments(Bundle args) {
                supper.setArguments(args);
                digitalOK = args.getBoolean("digitalOK");
        }

        @override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                        Bundle savedInstanceState){
                View v = null;
                If (digitalOK) {
                        v = inflater.inflate(R.layout.frag_digital, container,false);
                }else{
                        v = inflater.inflate(R.layout.frag_numeric, container,false);
                }
                return v;
        }
}
```

Réaliser l'application comme indiqué dans les étapes précédentes. Lancer l'émulateur pour tester votre code.