Par NIDHAL JELASSI
jelassi.nidhal@gmail.com

DÉVELOPPEMENT MOBILE

PROG. ANDROID

Chapitre 4 : Eléments graphiques avancés

INCLUSION DE LAYOUT

- Les interfaces peuvent aussi inclure d'autres interfaces, permettant de factoriser des morceaux d'interface. On utilise dans ce cas le mot clef include.

- Les valeurs des attributs spécifié dans le layout de la balise <include> (android:layout_*) écrasent celles des attributs du layout initial du fichier XML.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
<include android:id="@+id/include01"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         layout="@layout/accueil"
         ></include>
</LinearLayout>
```

INCLUSION : PROBLÈME

- Si le layout correspondant à accueil.xml contient lui aussi un LinearLayout, on risque d'avoir deux imbrications de LinearLayout inutiles car redondants :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android">
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android">
  <TextView ... />
  <TextView ... />
</LinearLayout>
</LinearLayout>
```

INCLUSION : SOLUTION

- Un layout doit contenir un unique root element, du type View ou ViewGroup. On ne peut pas mettre une série de TextView sans root element. Pour résoudre ce problème, on peut utiliser le tag « merge ». Si l'on réécrit le layout accueil.xml comme cela :

```
<merge xmlns:android="http://schemas.android.com/apk/res/android">
  <TextView ... />
  <TextView ... />
</merge>
```

INCLUSION : EXEMPLE

- L'inclusion de celui-ci dans un layout linéaire transformera :

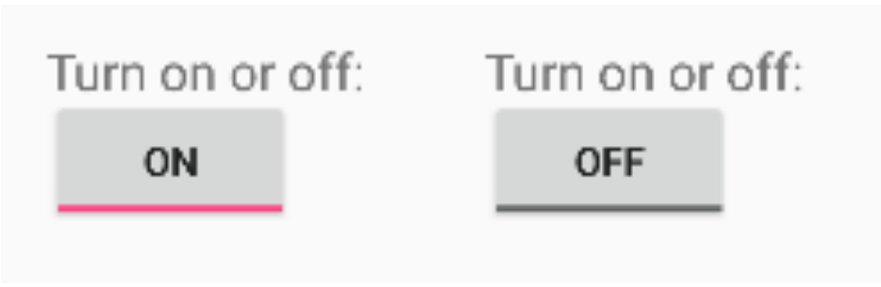
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android">
  <include android:id="@+id/include01" layout="@layout/acceuil"></include>
</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android">
  <TextView ... />
  <TextView ... />
</LinearLayout>
```

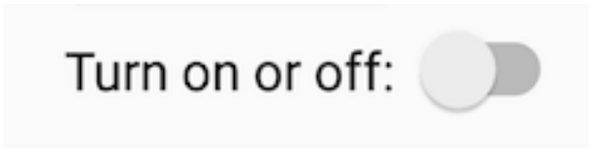
TOGGLE BUTTON SWITCH

TOGGLE BUTTON – SWITCH

- Un bouton « **Toggle** » représente un commutateur qui permet à l'utilisateur d'activer ou de désactiver des options en affichant ON et OFF.



- Android fournit également la classe `Switch`, qui ressemble à un commutateur à bascule pour l'activation et la désactivation.



- Les deux sont des extensions de la classe `CompoundButton`.

TOGGLE BUTTON

- Pour utiliser un bouton Toggle, il faut l'ajouter au layout XML :

```
<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/my_toggle"
    android:text="@string/turn_on_or_off"
    android:onClick="onToggleClick"/>
```

PROPRIÉTÉS

- On peut implémenter le traitement à faire en réponse à l'événement de click sur le bouton Toggle, il suffit d'ajouter l'attribut `android:onClick`.
- Dans cette fonction, pour détecter le changement d'état du bouton, créez un objet `CompoundButton.OnCheckedChangeListener` et affectez-le au bouton toggle en invoquant la méthode `setOnCheckedChangeListener()`.
- A noter que pour ajouter un label au bouton Toggle, il est nécessaire d'ajouter un TextView et ne pas compter sur l'attribut `android:text`.

TOGGLE BUTTON – EXEMPLE

```
public void onToggleClick(View view) {
   ToggleButton toggle = findViewById(R.id.my_toggle);
   toggle.setOnCheckedChangeListener(new
                              CompoundButton.OnCheckedChangeListener()
{
      public void onCheckedChanged(CompoundButton buttonView,
                              boolean isChecked) {
         StringBuffer onOff = new StringBuffer().append("On or off? ");
         if (isChecked) { // The toggle is enabled
            onOff.append("ON ");
         } else { // The toggle is disabled
            onOff.append("OFF ");
         }
         Toast.makeText(getApplicationContext(), onOff.toString(),
                              Toast.LENGTH_SHORT).show();
      }
   });
}
```

SWITCH

- L'attribut `android:text` permet de définir un String qui apparaît à gauche du bouton.

```
<Switch
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/my_switch"
        android:text="@string/turn_on_or_off"
        android:onClick="onSwitchClick"/>
```

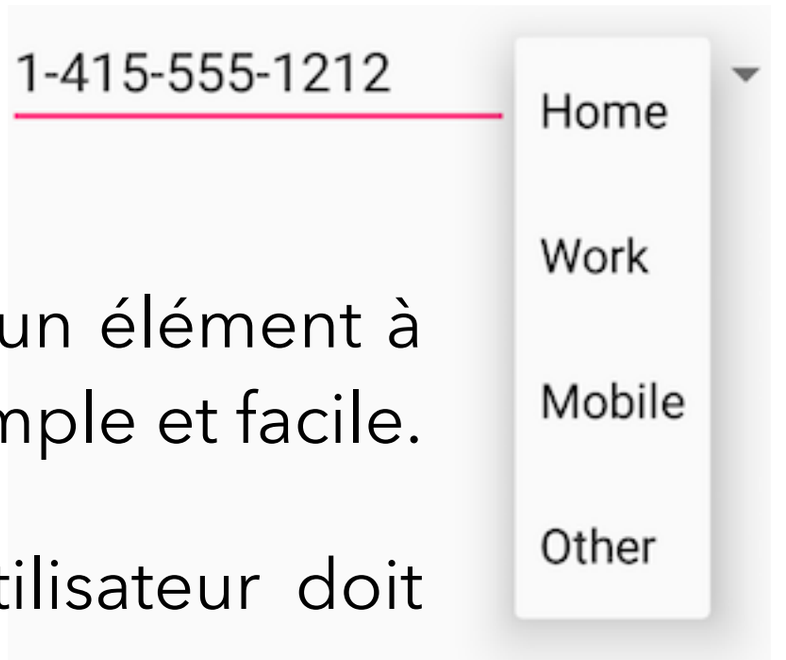
- On peut vérifier l'état du bouton via la méthode `setChecked(boolean)` sauf que l'utilisation de cette méthode annule l'exécution de la méthode appelée par l'attribut `android:onClick()`.

SWITCH - BOUTON

```
public void onSwitchClick(View view) {
   Switch aSwitch = findViewById(R.id.my_switch);
   aSwitch.setOnCheckedChangeListener(new
                              CompoundButton.OnCheckedChangeListener() {
      public void onCheckedChanged(CompoundButton buttonView,
                              boolean isChecked) {
         StringBuffer onOff = new StringBuffer().append("On or off? ");
         if (isChecked) { // The switch is enabled
            onOff.append("ON ");
         } else { // The switch is disabled
            onOff.append("OFF ");
         }
         Toast.makeText(getApplicationContext(), onOff.toString(),
                              Toast.LENGTH_SHORT).show();
      }
   });
}
```

SPINNER

SPINNER



- Un Spinner permet à l'utilisateur de sélectionner un élément à partir d'un ensemble d'éléments d'une manière simple et facile.
- Pour faire défiler les éléments d'un Spinner, l'utilisateur doit cliquer sur le composant dans l'interface.
- En général, il est recommandé d'utiliser les Spinner si l'utilisateur a le choix entre plus de 3 éléments. Si le nombre d'éléments est <= 3 et que votre interface n'est pas chargée, il est préférable d'opter pour les boutons radios au lieu d'un Spinner.

SPINNER

- Même s'il permet à l'utilisateur de scroller facilement, cette option n'est pas recommandé.

- Pour créer un Spinner, on utilisera la classe « `Spinner` » qui créera une View qui affichera des valeurs comme des Views enfants. On va ainsi suivre ces étapes là :

1. Créer un element Spinner dans votre Layout XML. Les valeurs sont créés et stockés dans un tableau et on utilisera, pour les afficher dans le Spinner un ArrayAdapter.

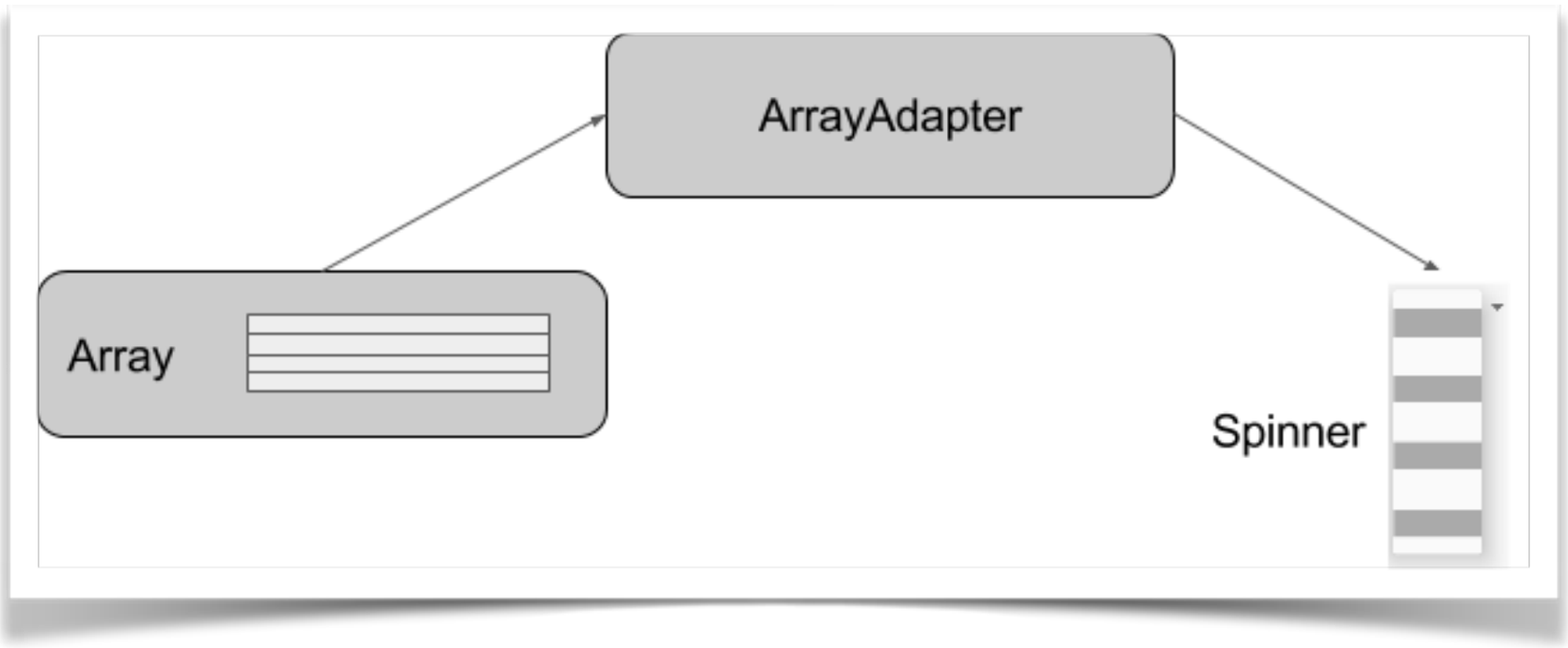
2. Créer l'élément Spinner (Java) et son Adapter.

3. Pour définir le callback de l'élément sélectionné dans le Spinner, mettez à jours l'activité qui utilise le Spinner pour implémenter l'interface `AdapterView.OnItemSelectedListener`.

SPINNER : CRÉATION

```
<Spinner
   android:id="@+id/label_spinner"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>
```

- Un Adapter est comme un intermédiaire entre 2 interfaces incompatibles. (Ex : L'adaptateur d'une carte mémoire avec un PC)



SPINNER : AJOUT D'ELEMENTS

- Les éléments du Spinner peuvent provenir de n'importe quel source mais doivent impérativement être fournies par un `SpinnerAdapter` ou un `ArrayAdapter`.
- Nous commençons par ajouter un tableau de String dans le fichier strings.xml

```
<string-array name="labels_array">
        <item>Home</item>
        <item>Work</item>
        <item>Mobile</item>
        <item>Other</item>
</string-array>
```

- On peut utiliser un `CursorAdapter` si les éléments proviennent d'un fichier ou d'une base de données.

SPINNER : IMPLÉMENTATION

- AdapterView permettra d'affecter les valeurs des éléments au Spinner.

```
public class MainActivity extends AppCompatActivity implements
            AdapterView.OnItemSelectedListener {
```

- En implémentation cette interface, on hérite des méthodes suivantes qui prennent en argument `AdapterView<?>`. The `<?>` précise la flexibilité de la méthode a accepter tout type de AdapterView.
  - `onItemSelected()` (obligatoire)
  - `onNothingSelected()` (optionnel)

SPINNER

- Dans le callback onCreate() de notre activité :

```
Spinner spinner = findViewById(R.id.label_spinner);
    if (spinner != null) {
            spinner.setOnItemSelectedListener(this);
    }
```

- Ensuite, créer une instance de ArrayAdapter en passant en argument le tableau de Strings (éléments) et le layout correspondant à chaque élément du Spinner.

```
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
                R.array.labels_array, android.R.layout.simple_spinner_item);
```

SPINNER : AFFICHAGE DES ÉLÉMENTS

- Il ne reste qu'à choisir la manière dont le Spinner va afficher les éléments au click de l'utilisateur :

```
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
```

- Et, enfin, appliquer l'Adapter au Spinner :

```
spinner.setAdapter(adapter);
```

SPINNER : ONITEMSELECTED()

- A la sélection d'un élément, c'est la méthode callback OnItemSelected() qui est invoquée

```
public void onItemSelected(AdapterView<?> adapterView, View view, int
                    i, long l) {
    String spinner_item = adapterView.getItemAtPosition(i).toString();
}
```

- Arguments :
  - view **View** : La Vue dans laquelle l'Adapter View a été cliqué
  - int **pos** : La position de la vue
  - long **id** : L'id de l'élément (vue sélectionné)

RECYCLERVIEW

RECYCLERVIEW VS LISTVIEW

- Dans une `ListView` classique, le système se charge de créer toutes les vues qui correspondent aux éléments de votre liste avant même qu'elles ne soient visible à l'écran. Ce système peut saturer la mémoire de votre machine virtuelle et lever une erreur **OutOfMemoryError**.

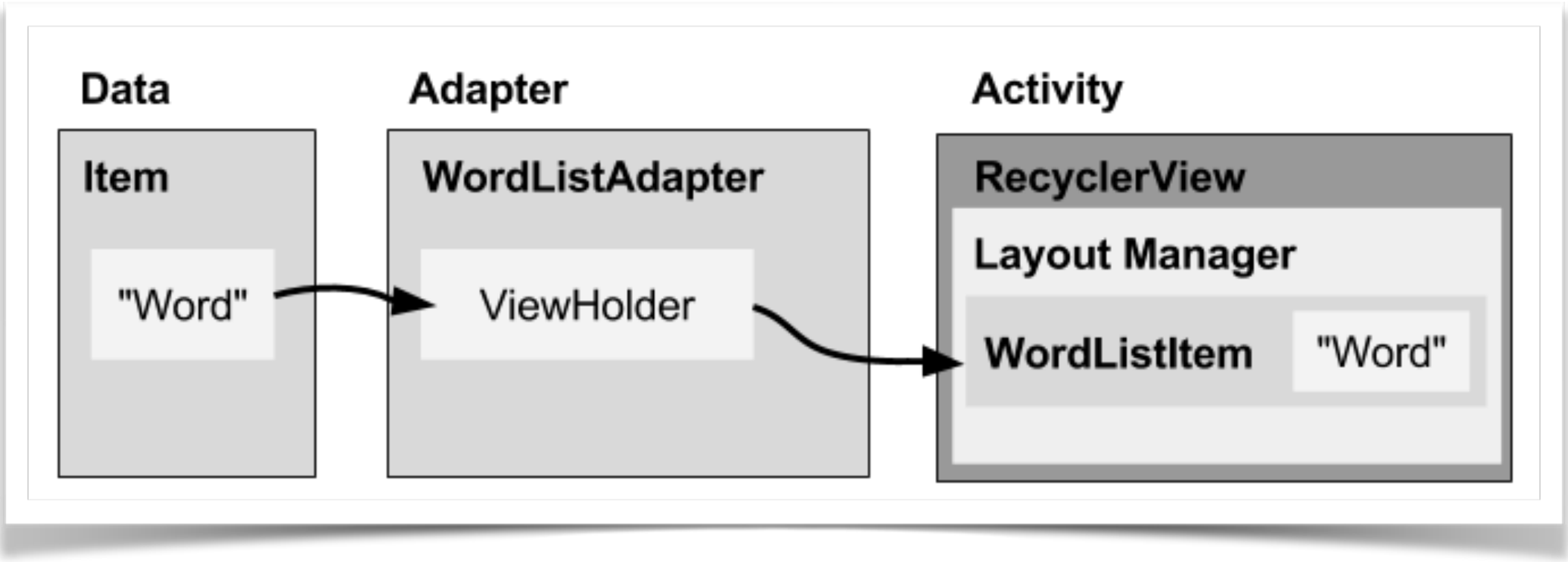
- Dans une `RecyclerView`, le système ne va charger que les vues qui seront visible à l'écran ! Lors du scroll, elle réutilisera les vues qui disparaissent pour charger les éléments suivants.

RECYCLER VIEW

- C'est un container « scrollable », adéquat pour les données volumineuses.
- Permet d'utiliser et de réutiliser un très grand nombre de Views.
- Met à jour les données très rapidement.

RECYCLER VIEW

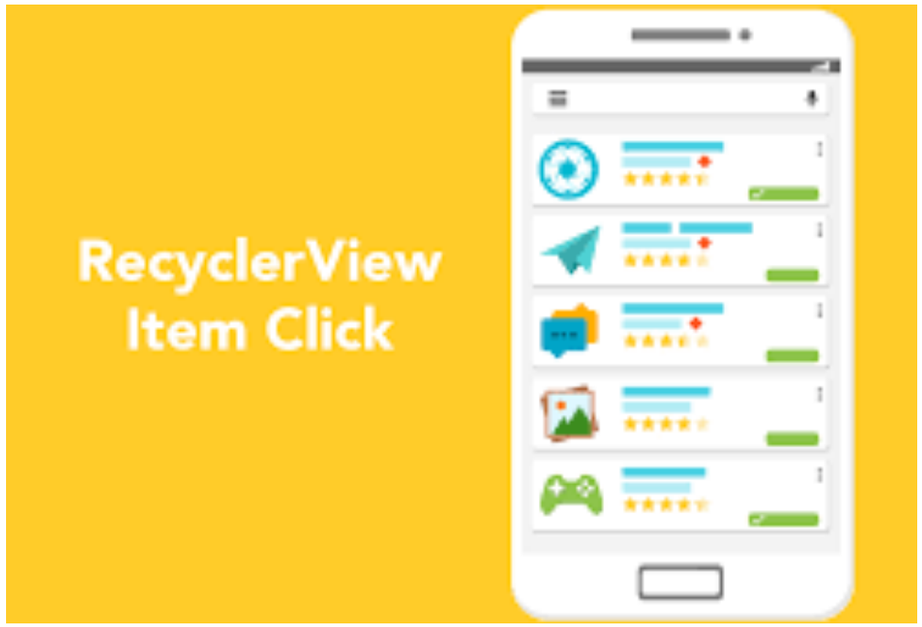
- Pour charger les données dans un RecyclerView, nous avons impérativement besoin d'un `Adapter`.



RECYCLER VIEW

- Un adapter permet à deux interfaces incompatibles de communiquer.
- Il joue le donc le rôle d'intermédiaire entre les données (Data) et la vue (View).

- Permet de gérer la création, l'ajout, la suppression et la modification des éléments Views.
- Hérite de `RecyclerView.Adapter`

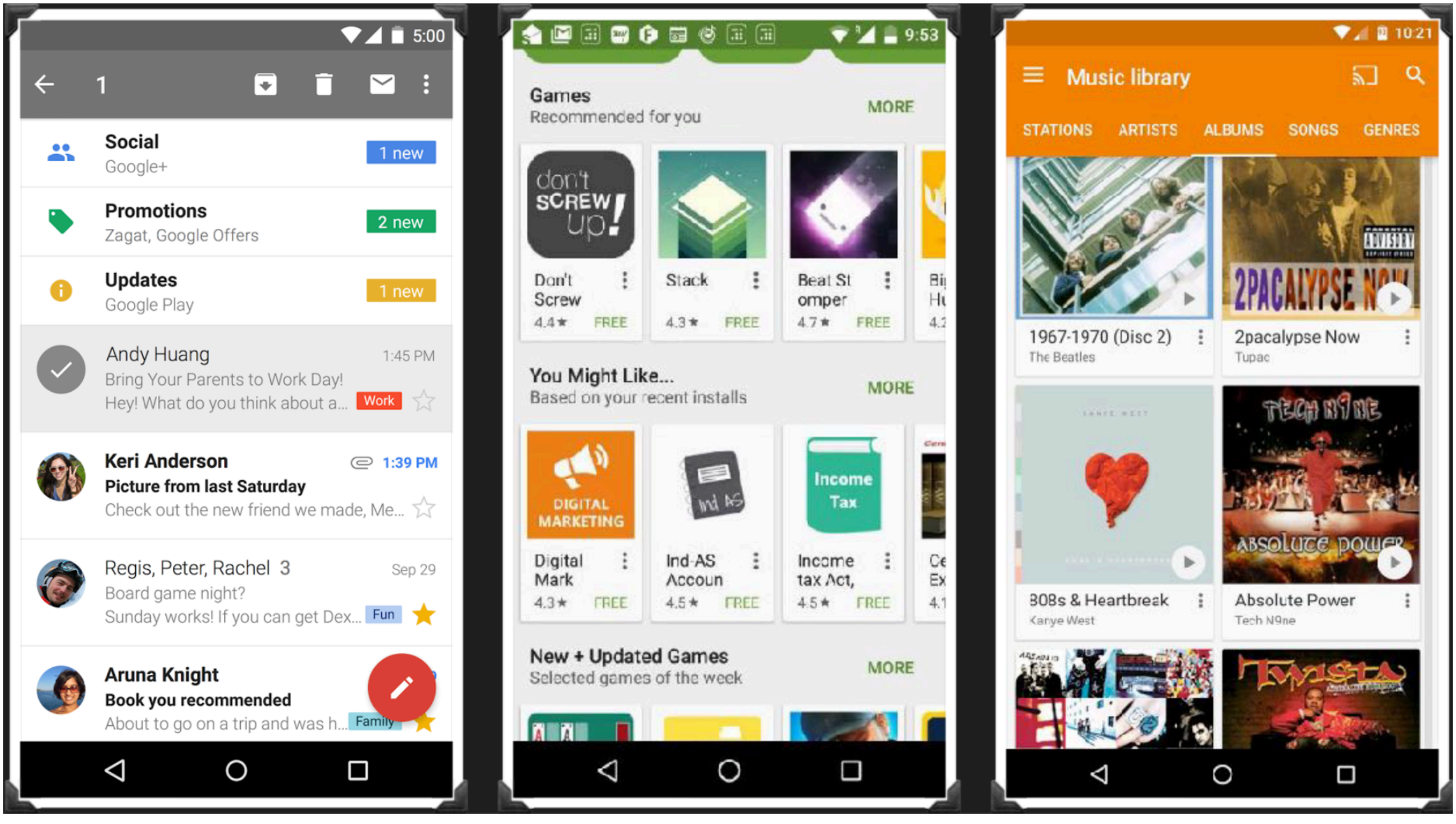


RECYCLER VIEW : LAYOUT MANAGER

- Nous avons aussi besoin d'un `LayoutManager`.
- Chaque ViewGroup a un `LayoutManager`.
- On en a besoin pour positionner un View dans un `RecyclerView`.

- Hérites de RecyclerView.LayoutManager
- Exemples :
  - LinearLayoutManager
  - GridLayoutManager, etc…

RECYCLER VIEW : LAYOUT MANAGER



RECYCLER VIEW : VIEW HOLDER

- On a également besoin d'un ViewHolder.
- Un ViewHolder est utilisé par l'Adapter pour préparer une Vue avec ses données pour chaque élément de la liste.
- Pour cela, il faut créer un Layout spécifique pour cet élément.
- Peut évidemment contenir des éléments cliquables.
- Il est placé par le Layout Manager.
- Hérite de RecyclerView.ViewHolder

IMPLÉMENTATION

Etapes d'implémentation :

1. Ajouter la dépendance RecyclerView dans le build.gradle (si nécessaire)

```
implementation 'com.android.support:recyclerview-v7:28.0.0-alpha3'
```

3. Ajouter un élément RecyclerView au Layout de l'activité.

4. Créer un Layout pour un élément de la liste.

5. Créer un Adapter et un ViewHolder.

6. Dans onCreate() de l'activité concernée, créer un RecyclerView avec l'Adapter et le ViewHolder créés précédemment.

LAYOUT

- Exemple de Layout pour un élément de la liste :

```
<LinearLayout …>
  <TextView
    android:id="@+id/word"
    style="@style/word_title" />
</LinearLayout>
```

ADAPTER

- Implémentation de l'Adapter

```
public class WordListAdapter
    extends RecyclerView.Adapter<WordListAdapter.WordViewHolder> {


    public WordListAdapter(Context context,
                           LinkedList<String> wordList) {
        mInflater = LayoutInflater.from(context);
        this.mWordList = wordList;
    }
}
```

- L'Adapter doit implémenter 3 méthodes : onCreateViewHolder(), onBindViewHolder() et getItemCount().

N.B : LayoutInflater est utilisé pour créer un nouvel object View à partir de l'une de vos XML.

ADAPTER

- **onCreateViewHolder**() qui, comme son nom l'indique, est exécutée lors de la création de notre ViewHolder, cette méthode n'est exécutée qu'une seule fois, c'est pourquoi nous allons charger la vue .xml de notre cellule ici.

- **onBindViewHolder**() qui est exécutée lorsque que le ViewHolder est de nouveau lié à l'adapter, c'est à dire lors du recyclage d'une cellule, nous recevons d'ailleurs comme paramètre sa position et l'instance du ViewHolder. On se contente dans cette méthode de mettre à jour notre cellule courante pour rafraichir notre TextView.

- **getItemCount**() qui permet de renvoyer le nombre d'items de notre liste, comme nous ne savons pas forcément combien d'éléments il y aura dans notre liste, j'utilise la méthode .size().

ADAPTER

- Commençons par la méthode onCreateViewHolder() :

```
@Override
public WordViewHolder onCreateViewHolder(
        ViewGroup parent, int viewType) {
        // Create view from layout
        View mItemView = mInflater.inflate(
             R.layout.wordlist_item, parent, false);
        return new WordViewHolder(mItemView, this);
}
```

ADAPTER

- Ensuite, la méthode onBindViewHolder() :

```
@Override
public void onBindViewHolder(
        WordViewHolder holder, int position) {
        // Retrieve the data for that position
        String mCurrent = mWordList.get(position);
        // Add the data to the view
        holder.wordItemView.setText(mCurrent);
}
```

ADAPTER

- Enfin, la méthode getItemCount() :

```
@Override
public int getItemCount() {
  // Return the number of data items to display
  return mWordList.size();
}
```

VIEWHOLDER

- A présent, on va créer un ViewHolder dans notre classe de l'Adapter.

```
class WordViewHolder extends RecyclerView.ViewHolder { //.. }
```

- Si on désire gérer les événements sur les éléments de la liste, on doit implémenter l'interface onClickListener.

```
class WordViewHolder extends RecyclerView.ViewHolder
                     implements View.OnClickListener { //.. }
```

VIEWHOLDER

- Implémentons maintenant le constructeur de notre ViewHolder :

```
public WordViewHolder(View itemView, WordListAdapter
adapter) {

  super(itemView);

  wordItemView = itemView.findViewById(R.id.word);

  this.mAdapter = adapter;

  itemView.setOnClickListener(this);
}
// Implement onClick()
```

ACTIVITY

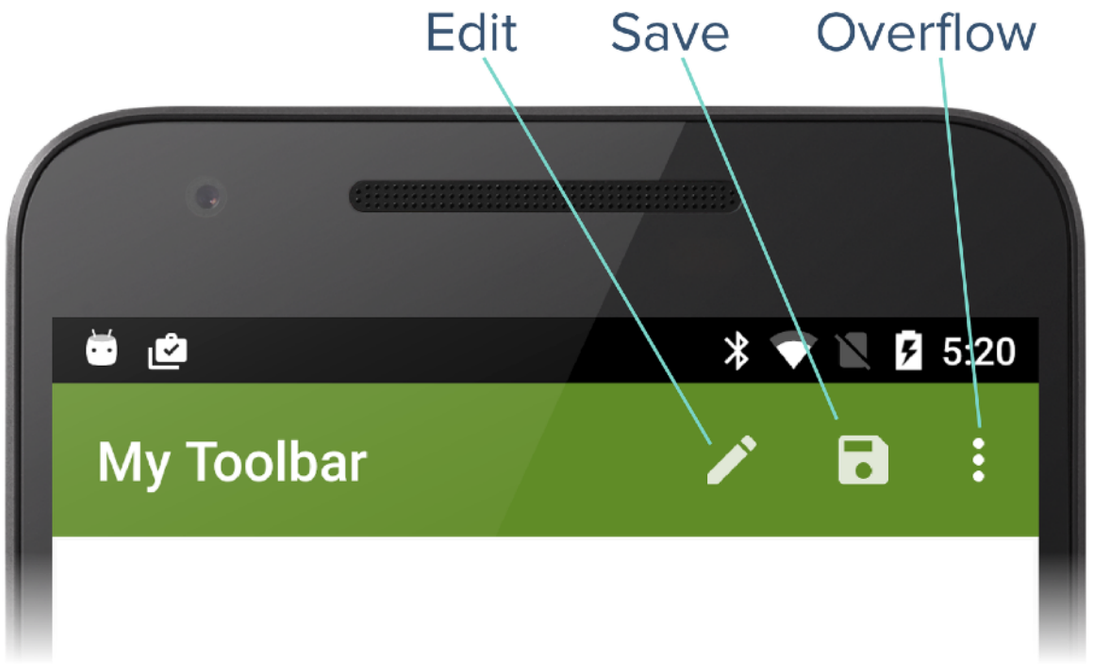
- Tout est presque prêt. On passe alors à la création du RecyclerView dans la méthode onCreate() de notre activité.

```
mRecyclerView = findViewById(R.id.recyclerview);
mAdapter = new WordListAdapter(this, mWordList);
mRecyclerView.setAdapter(mAdapter);
mRecyclerView.setLayoutManager(new
                        LinearLayoutManager(this));
```

ACTIONBAR

ACTIONBAR

- La barre d'actions (`ActionBar`) se situe en haut d'une activité.
- Elle peut afficher :
  - le titre de l'activité,
  - des icônes,
  - des actions qui peuvent être lancées
  - des vues additionnelles,
  - etc.



- Elle peut aussi être utilisée pour naviguer dans une application.

ACTIONBAR

- Une activité renseigne sa barre d'actions dans sa méthode onCreateOptionsMenu(). Ces entrées sont appelées des actions.
- Les actions pour une barre d'action sont définies dans un fichier de ressources XML.
- La classe MenuInflater permet d'enregistrer les actions définies dans le XML en les ajoutant dans la barre d'actions.
- L'attribut showAsAction permet de définir comment l'action est affichée.
- Par exemple, l'attribut ifRoom indique que l'action est affichée uniquement s'il reste suffisamment de place sur l'ActionBar.

ACTIONS

- Créer des Actions dans l'ActionBar

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
      <item
            android:id="@+id/action_refresh"
            android:orderInCategory="100"
            android:showAsAction="always"
            android:icon="@drawable/ic_action_search"
            android:title="Refresh"/>
      <item
            android:id="@+id/action_settings"
            android:title="Settings" >
      </item>
</menu>
```

MENUINFLATER

- Une instance de type MenuInflater peut être accédé avec la méthode getMenuInflater() de l'activité.
- L'exemple suivant montre la création d'actions dans la barre d'actions :

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.mainmenu, menu);
        return true;
}
```

PROPRIÉTÉS

- Visibilité :

```
ActionBar actionBar = getActionBar();
actionBar.hide();
actionBar.show();
```

- Texte :

```
ActionBar actionBar = getActionBar();
actionBar.setSubtitle("mytest");
actionBar.setTitle("Titre");
```

JELASSI.NIDHAL@GMAIL.COM

NIDHAL JELASSI