Android Partie 3

Interfaces Avancés

Les unités

⇨ dp, sp

- Ce sont des tailles « logiques ».
  - dp (ou dip) : Density independent Pixels
  - sp : Scale independent Pixels
- Ce sont celles-ci qu'il faut utiliser.
- dp est relatif par rapport à un écran de 160dpi.
  - sur un écran 160dpi => 1dp = 1px
  - sur un écran 320dpi => 1dp = 2px

- C'est le système qui convertit cette unité en pixels.

  => une image aura toujours la même taille

Les styles

⇨ introduction

- C’est un moyen de regrouper un ensemble de propriétés.
- Un style est défini dans un fichier xml.
- Comme pour les CSS, les styles sont hiérarchisés.
- Pratique quand par exemple plusieurs TextView doivent avoir le même rendu que cela soit sur la même page (activité) ou pas.

Les styles

⇨ exemple

Soit le TextView

```
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="@string/creerPel_nom"
        android:id="@+id/textView_nom" />
```

On peut faire :

```
<TextView
      android:style="@style/monStyle"
      android:text="@string/creerPel_nom"
      android:id="@+id/textView_nom" />
```

Avec le fichier styles.xml dans le dossier res/values

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
      <style name="monStyle"
            <item name="android:layout_width">wrap_content</item>
            <item name="android:layout_height">wrap_content</item>
            <item name="android:layout_marginTop">20dp</item>
       </style>
</resources>
```

Les styles

⇨ héritage

- Un style peut être hérité :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="monStyle" parent="@style/Text">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

```
<EditText
    android:style="@style/monStyle"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello, World!" />
```

Composants graphiques Avancés

⇨ les AdapterView et Adapter (1)

- Les AdapterView sont capables d'afficher une liste d'items.
- Très utiles pour afficher des données dynamiques.
- On utilise un Adapter pour fournir les items à l'AdapterView.
- AdapterView est une classe abstraite générique.

```
AdapterView<T extends android.widget.Adapter>
```

- android.widget.Adapter est une interface

Composants graphiques avancés

⇨ les AdapterView et Adapter (2)

- Un **Adapter** est une passerelle entre les données (tableau, liste ou encore curseur) et les objets graphiques.
- ArrayAdapter<T>, BaseAdapter, SpinnerAdapter et ListAdapter sont des interfaces qui héritent d'Adapter.
- Spinner, GridView, ListView sont des sous classes concrètes d'AdapterView.
- Pour relier l'Adapter à l'AdapterView utiliser la méthode :

```
public void setAdapter(T adapter)
```

Les Views

⇨le spinner (1)

- Permet de sélectionner une donnée dans une liste (combobox).
- Plusieurs solutions s'offrent à nous pour peupler un spinner :
  - par un tableau de String dans les ressources
    - Utilisation de l'attribut *android:entries* sur le spinner
    - Création d'un Adapter à partir de la ressource par la méthode *createFromResource()*
  - par un tableau de String dans le code et création d'un Adapter
    - Les données du tableau pourront provenir de sources extérieures (base de données, web service)

- Exemple :

Peupler un spinner par un

tableau de strings

```
String [] tabJours = {"Lundi","Mardi","Mercredi","Jeudi"};

Spinner spinJour = (Spinner) findViewById(R.id.spin_jour);

// Création de l'ArrayAdapter utilisant le tableau de string
ArrayAdapter<String> adapterJour =
            new ArrayAdapter<String>(this, tabJours,
                        android.R.layout.simple_spinner_item);
// relier l'Adapter à notre Spinner
spinJour.setAdapter(adapterJour);
```

- Exemple :

Peupler un spinner par un tableau ressource

(inspiré de la doc)

```
<string-array name="tab_jours">
        <item>Lundi</item>
        <item>Mardi</item>
        <item>Mercredi</item>
        <item>Jeudi</item>
        <item>Vendredi</item>
    </string-array>
```

```
Spinner spinJour = (Spinner) findViewById(R.id.spin_jour);
// Création de l'ArrayAdapter utilisant la ressource string
ArrayAdapter<CharSequence> adapterJour =
            ArrayAdapter.createFromResource(this,
            R.array.tab_jours, android.R.layout.simple_spinner_item);
// on peut spécifier le  layout de la liste
adapterJour.setDropDownViewResource(
                        android.R.layout.simple_spinner_dropdown_item);
// relier l'Adapter à notre Spinner
spinJour.setAdapter(adapterJour);
```

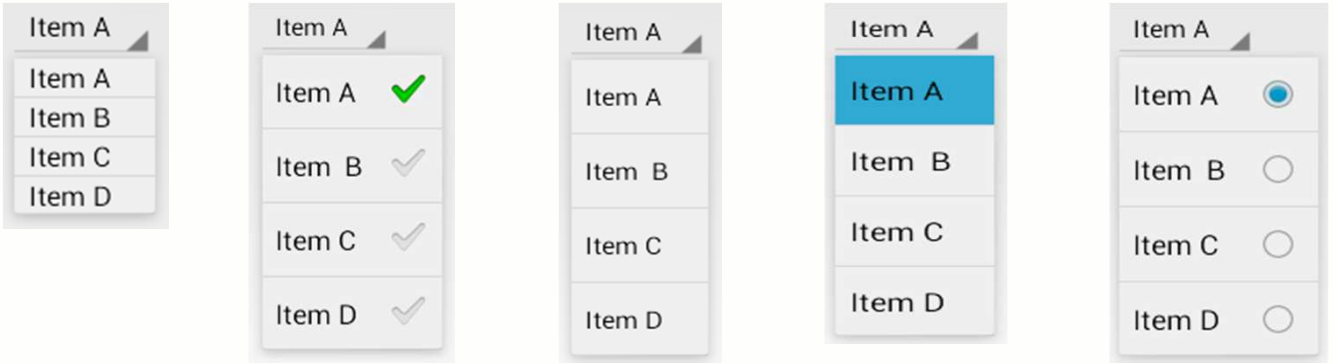
(par défaut le layout du spinner est *simple_spinner_item*)

Les Views

⇨le spinner (4)

- Android propose plusieurs layouts pour les spinner :
  - A : simple_spinner_item
  - B : simple_list_item_checked
  - C : simple_spinner_dropdown_item
  - D : simple_list_item_activated_1
  - E : simple_list_item_single_choice

A B C D E



Les Views

⇨la ListView (1)

- Le composant ListView permet d'afficher une liste d'items que l'utilisateur peut faire défiler.
- Les items de la liste sont cliquable.
- En l'associant à un EditText, on peut faire du filtrage (l'utilisateur tape une lettre, la liste s'adapte).
- Il faut définir le layout d'affichage des items de la liste (des layouts prédéfinis existent).
- Il y a plusieurs manières de construire une ListView :

1. Comme pour les Spinner, on défini un élément <ListView> dans le layout de l'activité.

- On relie la liste à un adapter par la méthode :

public void setAdapter(ListAdapter adapter)

avec

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,

android:R.layout.simple_list_item_1, données)

(android:R.layout.simple_list_item_1 est un layout prédéfini pour la représentation de nos items)

- Nous pourrions définir notre propre Layout :
- Dans ce cas, il faut utiliser un constructeur d'adapter à 4 paramètres :

```
ArrayAdapter<String> adapter = new
    ArrayAdapter<String>(this,
            R.layout.nomDuLayout,
            R.id.nomTextView, données)
```

nomDuLayout : nom du fichier xml représentant notre item

nomTextView : l'id du textView qui doit afficher le texte dans notre layout

- Exemple :
  - le fichier itemDevise.xml

```
<LinearLayout
        android:layout_width="matchParent"
        android:layout_height="matchParent"
        android:orientation="vertical"  >
        <TextView
                android:id="@+id/devise"
                android:layout_width="matchParent"
                android:layout_height="matchParent«  />
</LinearLayout>
```

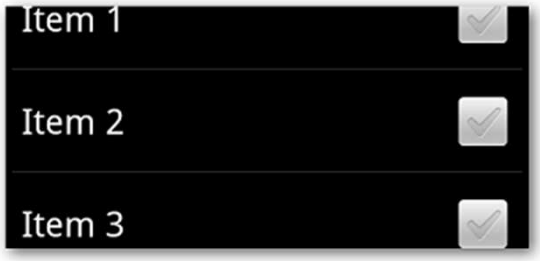
  - dans le onCreate()

```
adapter  = new ArrayAdapter<String>(this,
                                    R.layout.itemDevise,
                                    R.id.devise, devises)
```

- on peut lui ajouter un attribut :

android:choiceMode="le_mode"

avec le_mode : singleMode ou multipleMode



Par le code utiliser :

Void setChoiseMode(int mode) avec mode prenant soit ListView.CHOISE_MODE_SINGLE ou ListView.CHOISE_MODE_MULTIPLE

- Pour récupérer un (ou les) élément sélectionné :
  - sélection unique

    Int getCheckedItemPosition()
  - sélection multiple

    SparseBooleanArray getCheckedItemPositions()

    (SparseBooleanArray est équivalent à Hashmap<integer, Boolean>)

- Au niveau événementiel, pour gérer le clic sur un élément :

  void setOnItemClickListener(

  AdapterView.OnItemClickListener)

  Et redéfinir :

  void onItemClick (AdapterView<?> adapter,

  View v, int position, long id)

  - adapter est l'AdapterView
  - v est la View cliquée
  - position est la position de l'item dans la liste
  - id est l'identifiant de la Vie

2. la liste prend tout l'écran alors on peut construire une activité qui étendra une ListActivity.

- On peut relier la liste à un adapter par la méthode :

  public void setListApadter(ListAdapter adapter)

- On pourra utiliser la méthode :

  public ListView getListView()

  pour récupérer la listView si besoin.

- Exemple :

```
public class DeviseListViewActivity extends ListActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

      ArrayAdapter<String> adapter = new
                        ArrayAdapter<String>(this,
                        android.R.layout.simple_list_item_1,
                        devises);

        setListAdapter(adapter);
```

Cet argument pourrait être notre Layout de tout à l'heure R.layout.itemDevise

- Afficher plusieurs informations par item dans une listView :
- On peut utiliser le SimpleAdapter

SimpleAdapter(Context context,

List<? extends Map<String, ?>> data,

int resource, String[] from, int[] to)

  - resource : l'id du layout
  - from : tableau des clés de la Map
  - to : les id des Views du layout

Afficher des objets dans une listView :

- Construire un Adapter en dérivant la classe BaseAdapter et implémenter les méthodes :
  - public int getCount()
  - public Object getItem(int position)
  - public long getItemId(int position)
  - public View getView(int position, View convertView, ViewGroup parent)
    - position : position de l'item dans la liste
    - convertView : voir diapos suivantes
    - parent : la listeView

- la méthode getView() est appelée à chaque fois que la vue d'un item doit être créée.
- Il faut un moyen de créer cette vue
  1. On fait appel à un service du système que l'on récupère dans une activité par :

     LayoutInflater Inflater getSystemInflater(
     LAYOUT_INFLATER_SERVICE)

2. on appel la méthode inflate() de cet inflater qui va créer notre vue item :

   View Inflate (int id, ViewGroup parent)

   id : l'identificateur de notre layout d'items

   parent : la racine sur laquelle se rattache la vue

   (si null, racine courante)

3. On remplit notre vue item (ex setText() …)
4. la méthode getView() retourne cette vue

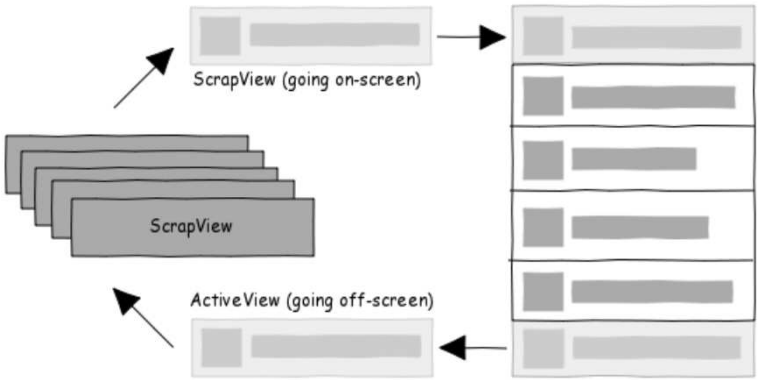
- Exemple de getView() :

```
public View getView(int position, View convertView, ViewGroup parent) {
    LayoutInflater inflater = null;
    inflater = (LayoutInflater) getSystemService(
                                        Context.LAYOUT_INFLATER_SERVICE);

    View itemView = inflater.inflate(R.layout.monItemLayout, parent);
    TextView txtDev = (TextView)itemView.findViewById(R.id.devise);

    txtDev.setText(devises.get(position).getLibelle();

    retrurn itemView;
}
```

Et le paramètre convertView ??

- En fait une ListeView garde trace des vues de chaque item déjà construit.

C'est une réserve de ScrapView.



- Il faut donc tester dans la méthode getView() si le paramètre convertView est nul ou pas :
  - si null alors construction de la vue item
  - si non null alors on le récupère
  - Dans les duex cas, on le réhydrate avec les données

- Notre getView() :

```
public View getView(int position, View convertView, ViewGroup parent) {
    View itemView = null;
    if (convertView == null) {
        LayoutInflater inflater = (LayoutInflater) getSystemService(
                                        Context.LAYOUT_INFLATER_SERVICE);
        itemView = inflater.inflate(R.layout.monItemLayout, parent);
    } else {
        itemView = convertView;
    }
    TextView txtDev = (TextView)itemView.findViewById(R.id.devise);

    txtDev.setText(devises.get(position).getLibelle();

    retrurn itemView;
}
```

Optimisation : utilisation d'un cache

- On créer une classe
- On utilise setTag() et getTag()

```
class CacheItem {
    private TextView txtDev;

    getTxtDev() …
    setTxtDev() …
}
```

```
public View getView(int position, View convertView, ViewGroup parent) {
    CacheItem item = null;
    if (convertView == null) {
        item = new CacheItem();
        LayoutInflater inflater = (LayoutInflater) getSystemService(
                                        Context.LAYOUT_INFLATER_SERVICE);
        convertView = inflater.inflate(R.layout.monItemLayout, parent);
        item.setTxtDev((TextView)convertView.findViewById(R.id.devise));
        convertView.setTag(item);
    } else {
        item = (CacheItem)convertView.getTag();
    }
    item.getTxtDev().setText(devises.get(position).getLibelle());

    retrurn convertView;
}
```

Les boites de dialogue

Dialog (1)

- C'est une petite fenêtre modale qui passe au premier plan. Elle est liée à l'activité.
- Pour la créer, il faut appeler la méthode :

  showDialog(int id)

  - id est un entier représentant la boite
  - Mettre des id différents si on a plusieurs boite dans la même activité.
- On utilise généralement des sous classe de la classe Dialog :
  - AlertDialog, ProgressDialog, DatePickerDialog, TimePickerDialog

Les boites de dialogue

Dialog (2)

- Android pour optimiser, ne créer la boite de dialogue qu'une seule fois :
  - La classe activity possède la méthode de callback onCreateDialog(int id) qui sera appelée à la toute première utilisation de la boite.
    - Cette méthode est obligatoire, c'est ici que l'on va créer la boite.
  - Aux autres utilisations de la boite, c'est la méthode onPrepareDialog(int id, Dialog box) qui sera appelée.
    - C'est ici que l'on modifiera la boite si elle doit être modifiée à chaque utilistion.

Les boites de dialogue

Dialog (3)

- Pour construire la boite, on fait appel à un builder :

```
AlertDialog.Builder monBuilder = new
                                 AlertDialog.Builder(this)
```

- Tester le paramètre id afin de traiter la bonne boite (si on en a plusieurs)
- Paramètrer la boite en utilisant différentes méthode de l'objet Builder :

  setMessage(), setPositiveButton()

- Enfin, on appelle la méthode show() sur l'objet Dialog (ici monBuilder.show()).

Les boites de dialogue

Dialog (4)

- Exemple :

```
private static final int DIALOG_QUIT = 10;

public void onClick(View view) {
    showDialog(DIALOG_QUIT);
}
```

```
@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
        case DIALOG_QUIT:
            AlertDialog.Builder adb = new AlertDialog.Builder(this);
            //on attribut un titre à notre boite de dialogue
            adb.setTitle("Confirmation de sortie");
            //on insère un message à notre boite de dialogue
            adb.setMessage("Est-vous sur de vouloir quitter ? ");
            //on attache un listenner sur le bouton Oui
            adb.setPositiveButton("Oui", new DialogInterface.OnClickListener() {
                  public void onClick(DialogInterface dialog, int id) {
                     // ToDo !   });
            //on attache un listenner sur le bouton Non
            adb.setNegativeButton("Non", new DialogInterface.OnClickListener() {
                  public void onClick(DialogInterface dialog, int id) {
                     // ToDo !   });
            //on affiche la boite de dialogue
            adb.show();
    }
    return super.onCreateDialog(id);
}
```

Les boites de dialogue

Dialog (5)

- Depuis la version 3 d'Android, vous devez créer vos boites de dialog en sous classant la classe :

  android.support.v4.app.DialogFragment

- Il faut redéfinir la méthode :

  public Dialog onCreateDialog(Bundle savedInstanceState)

- Dans cette méthode :
  - Créer un Builder (AlertDialogBuilder) comme décrit dans les précédentes diapos.
  - On obtient l'objet Dialog en appelant ma méthode create() sur le builder.
  - Retourner l'objet ainsi créé.

Dialog (6)

- Dans l'activié :
- Créer une instance de notre dialogue
- Appeler sur cet objet la méthode :

  show(FragmentManager fm, String tag)
  - On récupère le FragmentManager par :

    getFragmentManager()
  - Le tag sert au système pour gérer la vie du fragment

```
public void clicQuitter(View v){
    DialogFragment newFragment = new QuitDialogFragment();
    newFragment.show(getSupportFragmentManager(), "quit");
}
```

- Exemple :

```
import android.app.AlertDialog;
import android.app.Dialog;
import android.support.v4.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;

public class QuitDialogFragment extends DialogFragment {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_quit)
               .setPositiveButton(R.string.oui, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       getActivity().finish();  }
               })
               .setNegativeButton(R.string.non, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) { }
               });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

Les boites de dialogue

Dialog (5)

- Une boite de dialogue comporte 3 zones :
  - Le titre : setTitle() *(optionnel)*
  - La zone de contenu:
    - Un message : setMessage()
    - Une liste : setItems()
    - Un Layout perso : setView()
  - Les boutons d'action : voir ci-dessous
- il ne peut y avoir que 3 boutons paramétrés par :
  - builder.setPositiveButton()
  - builder.setNeutralButton()
  - builder.setNegativeButton()