



# **Chapitre**

## **Les interfaces utilisateurs avancés avec Android**

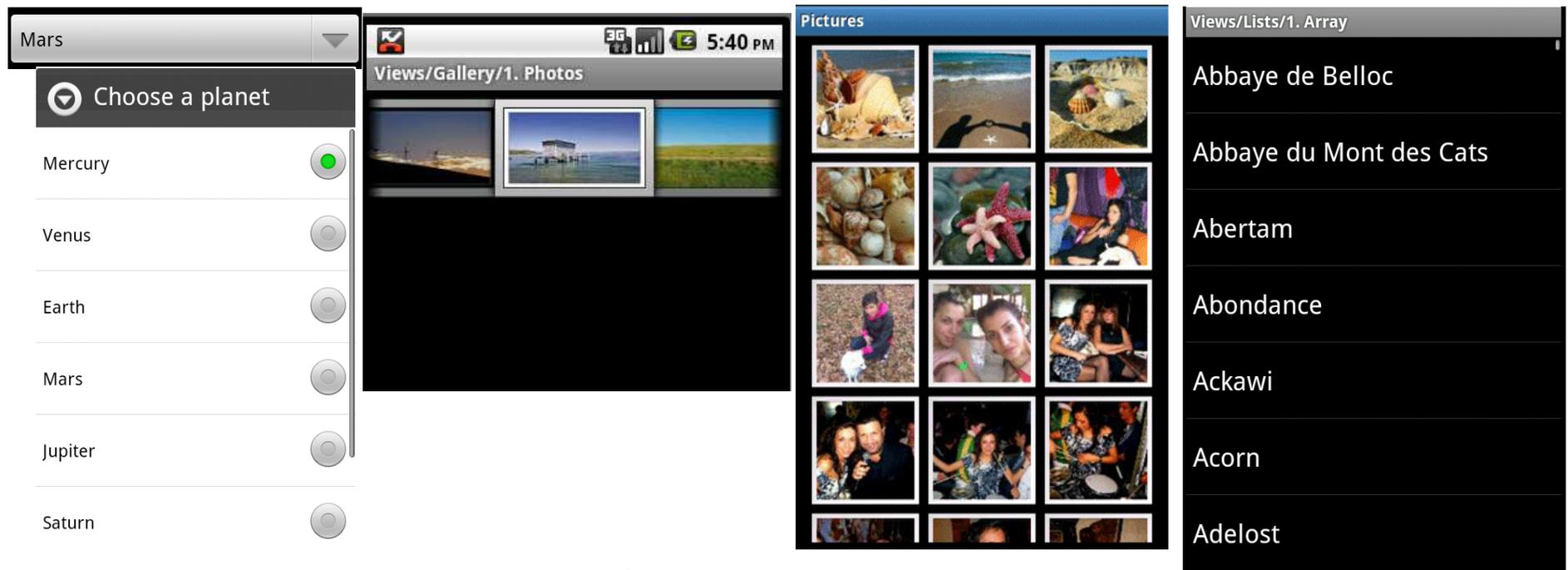
# Plan du chapitre IHM avancé

- ❑ Les composants avancés. Les Adapter et AdapterView. Exemple de la ListView
- ❑ Les View "standards" pour les items de ListView
- ❑ Afficher des objets dans une ListView, l'IHM de chaque item, la méthode getView()
- ❑ La UI Thread
- ❑ La classe AsyncTask
- ❑ Les fragments

# Composants IHM avancés

- ❑ Certains composants graphiques permettent d'afficher beaucoup d'items par des mécanismes de défilement. Ce sont :

❑ les Spinner, les Gallery, les GridView, les ListView



- ❑ Il y en a d'autres. Voir à Apis Demos | Views

# AdapterView **et** Adapter

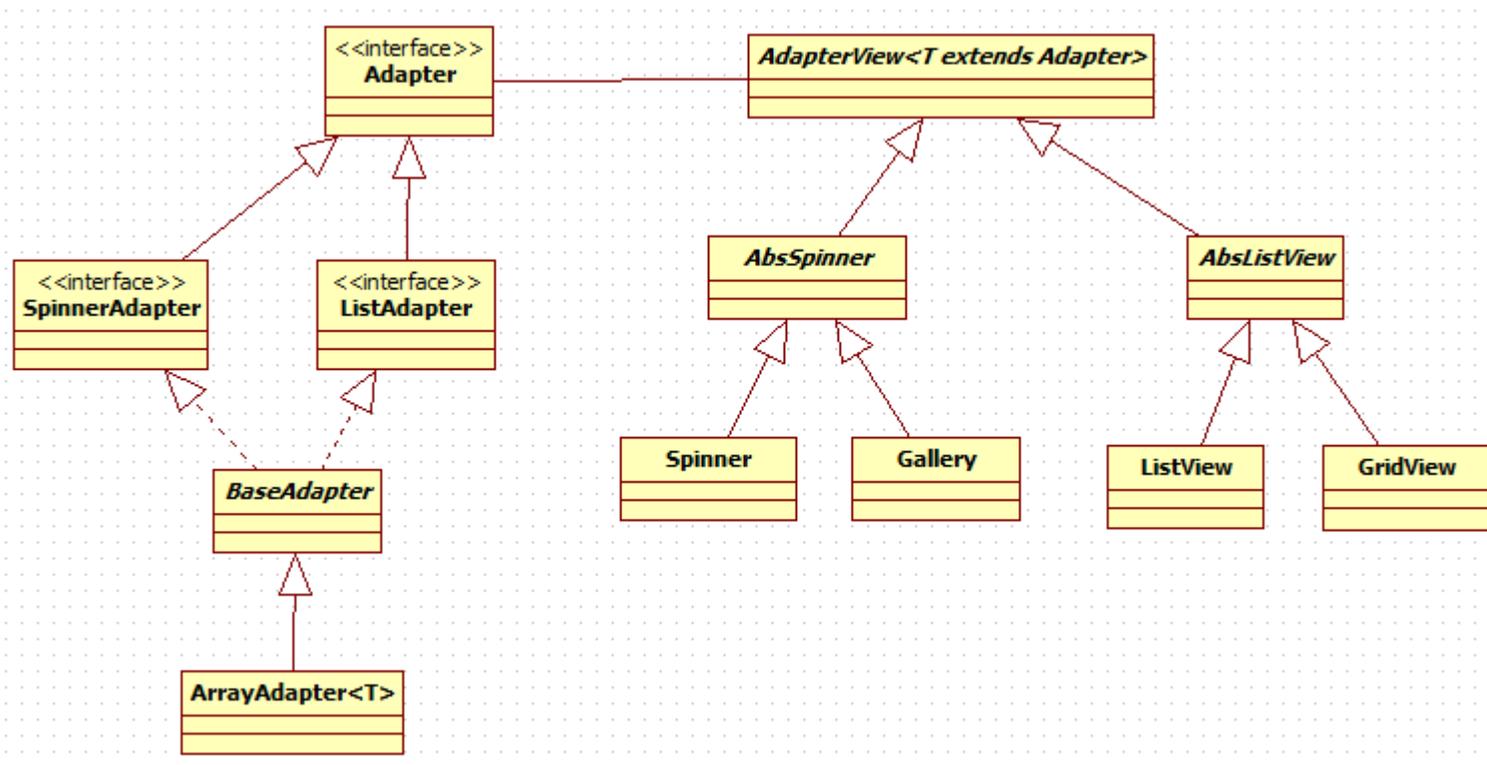
- ❑ Les composants graphiques qui affichent un ensemble d'items sont des `AdapterViews`
- ❑ Ces composants sont très utiles si les données qu'ils affichent ne sont pas connues à la création du programme mais seront chargées dynamiquement
- ❑ Pour fournir les items à ces composants graphiques, on utilise des `Adapter`
- ❑ Comme `AdapterView` est en fait `AdapterView<T extends android.widget.Adapter>`, un `AdapterView` est toujours associé à un `Adapter`
- ❑ L'`Adapter` sert de lien avec les données à afficher et l'`AdapterView`. Bref c'est le design pattern adapter
- ❑ Les `AdapterView` gèrent l'affichage de ces items
- ❑ "An Adapter object acts as a bridge between an AdapterView and the underlying data for that view. The Adapter provides access to the data items. The Adapter is also responsible for making a View for each item in the data set."  
(<http://developer.android.com/reference/android/widget/Adapter.html>)

# AdapterView, Adapter **et** **compagnie ! (1/2)**

- ❑ `android.widget.AdapterView<T extends android.widget.Adapter>` est une classe abstraite générique
- ❑ `android.widget.Adapter` est une interface. Beaucoup de classes implémentent cette interface : `ArrayAdapter<T>`, `BaseAdapter`
- ❑ `SpinnerAdapter` et `ListAdapter` sont des interfaces qui héritent de `Adapter`
- ❑ `Spinner`, `Gallery`, `GridView`, `ListView` sont des sous classes concrètes d'`AdapterView`
- ❑ L'association entre un `AdapterView` et un `Adapter` est faite à l'aide de la méthode  
`public void setAdapter (T adapter)`
- ❑ Question : Où est indiqué que le paramètre `T` est un type implémentant `Adapter` ?
- ❑ Réponse : Dans la définition de la classe `AdapterView`

# AdapterView, Adapter et compagnie ! (2/2)

□ On a donc :



# Tutoriaux pour les AdapterView

- ❑ Beaucoup de documentation existe pour les AdapterView
- ❑ Dans la documentation "officielle", on a des tutoriaux sur ces composants à partir de la définition de leur classe :
  - ❑ Spinner :  
<http://developer.android.com/guide/topics/ui/controls/spinner.html>
  - ❑ Gallery : est, en fait, dépréciée à partir de l'API 16. Il faut utiliser des `HorizontalScrollView` et des `ViewPager`
  - ❑ GridView :  
<http://developer.android.com/guide/topics/ui/layout/gridview.html>
  - ❑ ListView :  
<http://developer.android.com/guide/topics/ui/layout/listview.html>

# Le composant ListView

□ "A (List)view (that) shows items in a vertically scrolling list. The items come from the `ListAdapter` associated with this view." (\*)

□ Parfois les items proviennent d'un accès distant

□ Héritage de `ListView` :

```
java.lang.Object
└─ android.view.View
    └─ android.view.ViewGroup
        └─ android.widget.AdapterView<T extends android.widget.Adapter>
            └─ android.widget.AbsListView
                └─ android.widget.ListView
```

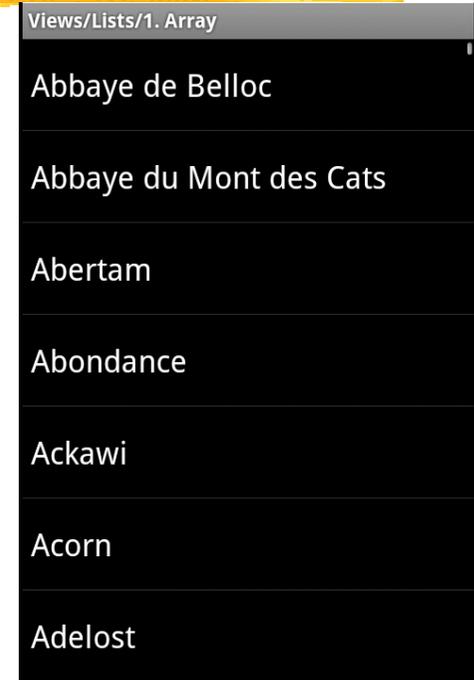
□ Une `ListView` est un `AdapterView` qui est "a view whose children are determined by an `Adapter`." (\*\*)

□ sources : (\*)

<http://developer.android.com/reference/android/widget/ListView.html>

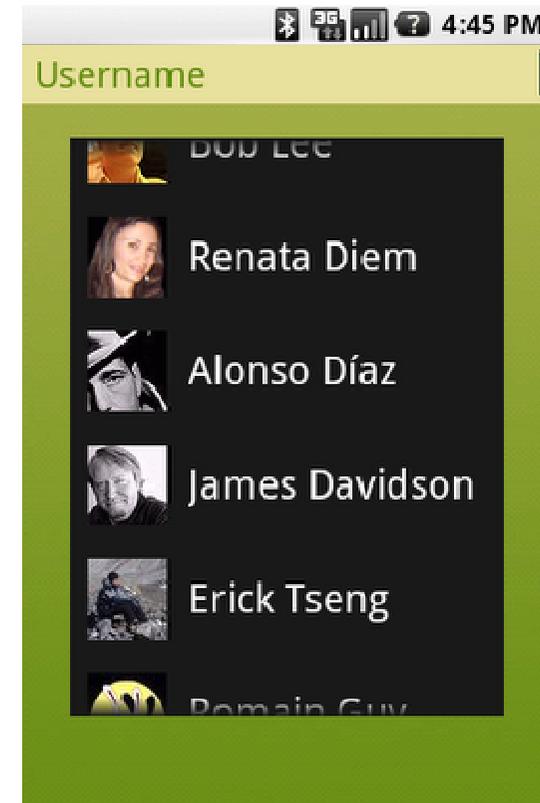
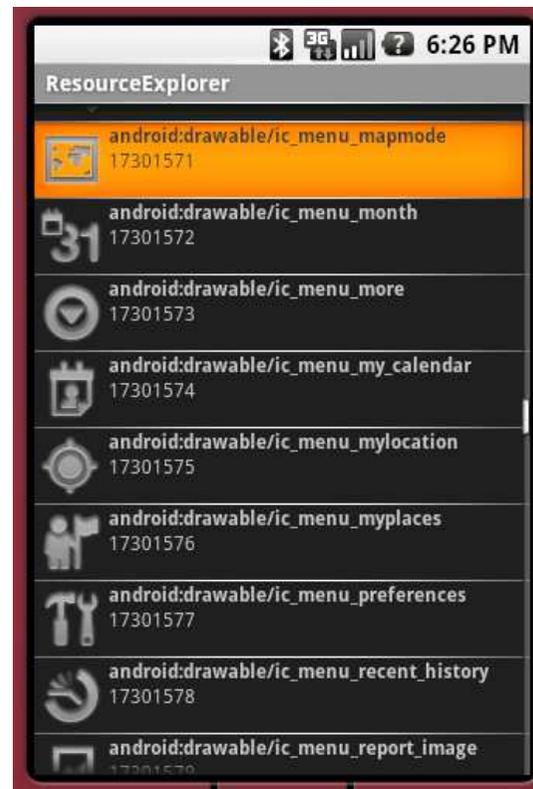
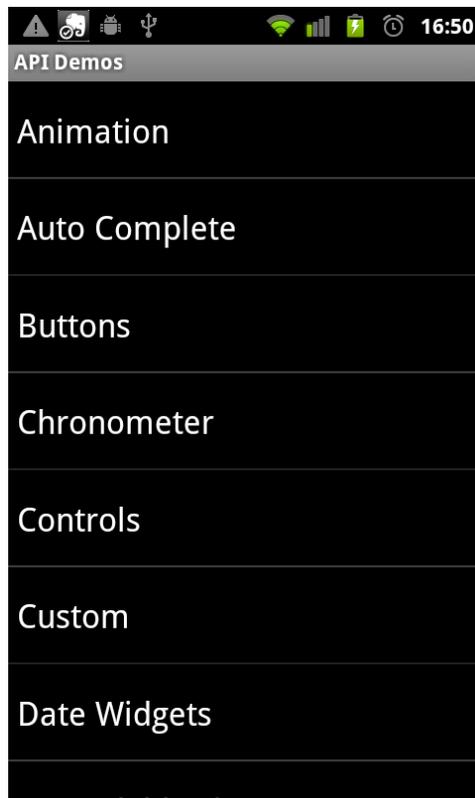
(\*\*)

<http://developer.android.com/reference/android/widget/AdapterView.html>



# Exemple de ListView

☐ Ce peut être :



(merci jean-michel)

# Le composant `Listview`

- ❑ Il permet d'afficher une (grande) liste d'items, accessible par défilement mais aussi par filtre : l'utilisateur tape les premières lettres, le `Listview` n'affiche plus que les items qui commencent par ces lettres
- ❑ Pour définir un `Listview`, on commence par indiquer comment sera affiché chaque item du `Listview`. Par exemple sous forme de `TextView`. On construit donc le fichier `res/layout/list_item.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp"
    android:textSize="16sp" >
</TextView>
```

- ❑ Une démo : projet `Ch3bisHelloListView`

# L'activité `ListActivity`

- ❑ Une `ListView` peut occuper tout l'écran. Dans ce cas, elle peut être construite à partir d'une `android.app.ListActivity` (qui hérite de `android.app.Activity`)
- ❑ Lorsqu'on construit une classe qui hérite de `ListActivity`, on peut alors utiliser plusieurs méthodes de la classe `ListActivity` :
  - ❑ `public ListView getListView()` qui retourne le composant graphique `ListView` associé à cette activité
  - ❑ `public void setListAdapter(ListAdapter adapter)` positionne le `ListAdapter` associé à la `ListView` de cette activité `ListActivity`
- ❑ Et on spécialise la méthode `onCreate()` (comme d'hab) et on a une `Activity` qui présente une `ListView`

# ListView : 1er exemple à l'aide de ListActivity

```
public class HelloListViewActivity extends ListActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setListAdapter(new ArrayAdapter<String>(this, R.layout.list_item, COUNTRIES));

        ListView lv = getListView();
        lv.setTextFilterEnabled(true);

        lv.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id) {
                // le code lancé lors de la sélection d'un item
            }
        });
    }
    static final String[] COUNTRIES = new String[] { ... };
}
```

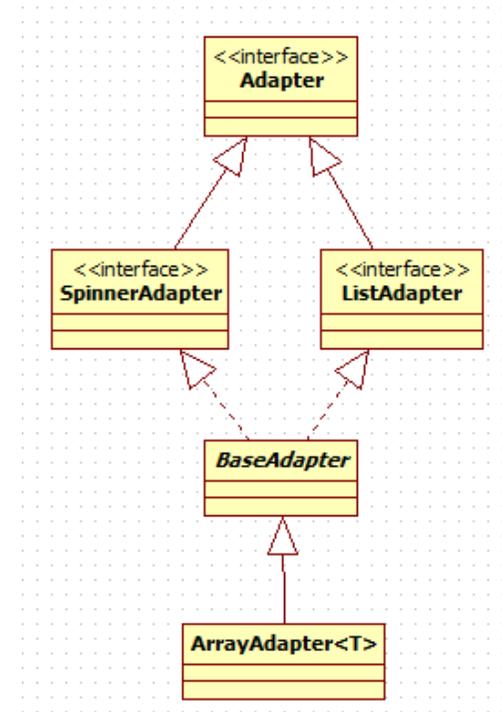


# ArrayAdapter<T> pour ListView

- ❑ On construit le `ListAdapter` à partir d'un `ArrayAdapter<String>`. En fait on a l'arborescence (d'interfaces, de classe abstraite et de classe générique !) suivante :
- ❑ On a utilisé le constructeur :

```
public ArrayAdapter (Context context,  
    int textViewResourceId,  
    T[] objects)
```

car :
  - Activity dérive de Context :
  - textViewResourceId repère l'IHM qui modélise un item
  - objects est le tableau de données à afficher



# Sélection dans une ListView

- ❑ La `ListView` construite à partir de l'activité et récupérée par `getListView()`, est sensible à la sélection par préfixe grâce à l'appel `setTextFilterEnabled(true)` ;
- ❑ Euh cela fonctionne dans un AVD, mais sur un vrai smartphone ?
- ❑ Il suffit d'ajouter une zone de texte qui filtre
- ❑ Demo `Ch3bisListViewAvecFiltreProjet`
- ❑ source :  
<http://www.androidhive.info/2012/09/android-adding-search-functionality-to-listview/>



# Sélection dans une ListView :

## activity\_main.xml

- ❑ Le fichier IHM layout\activity\_main.xml de l'activité principale est :

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

    <!-- EditText pour la recherche dans la ListView -->
    <EditText android:id="@+id/inputSearch"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Search products.." />

    <!-- la ListView -->
    <ListView
        android:id="@+id/list_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

# Sélection dans une ListView : l'IHM d'un item

- ❑ Ici le fichier d'IHM d'un item de la `ListView`  
`layout\list_item.xml` est :

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

    <TextView android:id="@+id/product_name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:padding="10dip"
        android:textSize="16dip"
        android:textStyle="bold"/>

</LinearLayout>
```

- ❑ Il va falloir indiquer ce que doit contenir le `TextView` d'id `product_name`. D'où l'utilisation d'un constructeur d'`ArrayAdapter` à 4 arguments diapo suivante

# Sélection dans une ListView : l'activité principale

- Dans le `onCreate()` de l'activité principale on a :

```
String products[] = {
    "Dell Inspiron", "HTC One X", "HTC Wildfire S", "HTC Sense",
    "HTC Sensation XE", "iPhone 4S", "Samsung Galaxy Note 800",
    "Samsung Galaxy S3", "MacBook Air", "Mac Mini", "MacBook Pro"};

inputSearch = (EditText) findViewById(R.id.inputSearch);

lv = (ListView) findViewById(R.id.list_view);
adapter = new ArrayAdapter<String>(this, R.layout.list_item,
    R.id.product_name, products);
lv.setAdapter(adapter);
```

- On a alors la bonne interface graphique

# Sélection dans une `ListView` : le constructeur `ArrayAdapter`

- ❑ Dans le `onCreate()` de l'activité principale on a :

```
String products[] = { ... };  
...  
adapter = new ArrayAdapter<String>(this, R.layout.list_item,  
    R.id.product_name, products);  
lv.setAdapter(adapter);
```

- ❑ On a utilisé le constructeur générique (T)

```
public ArrayAdapter (Context context, int  
resource, int textViewResourceId, T[] objects)
```

avec

- ❑ `context` : le contexte
- ❑ `resource` : l'ID du fichier XML d'IHM pour chaque item de la liste. Dans notre cas, chaque item de la `ListView` est un `LinearLayout` contenant un `TextView` (cf. fichier `list_item.xml`)
- ❑ `textViewResourceId` : l'ID du composant graphique, dans le fichier `list_item.xml`, à alimenter pour chaque item
- ❑ `objects` : le tableau des objets qui alimentent la `ListView`

- ❑ Et le filtre ?

# Sélection dans une ListView : le filtre

- ❑ Il suffit d'ajouter dans le `onCreate()` de l'activité principale :

```
inputSearch.addTextChangedListener(new TextWatcher() {
    @Override
    // appelé pour indiquer que count caractères commençant à start dans cs
    // viennent de remplacer un ancien texte de longueur before
    public void onTextChanged(CharSequence cs, int start, int before, int count) {
        MainActivity.this.adapter.getFilter().filter(cs);
    }

    @Override
    // appelé pour indiquer que count caractères commençant à start dans s
    // vont être remplacés par un texte de longueur after
    public void beforeTextChanged(CharSequence s, int start, int count,
        int after) { }

    @Override
    // appelé pour indiquer que dans s le texte a changé
    public void afterTextChanged(Editable s) { }
});
```

- ❑ Quand le texte change dans le `TextView`, immédiatement, le filtre de l'adaptateur est changé en conséquence

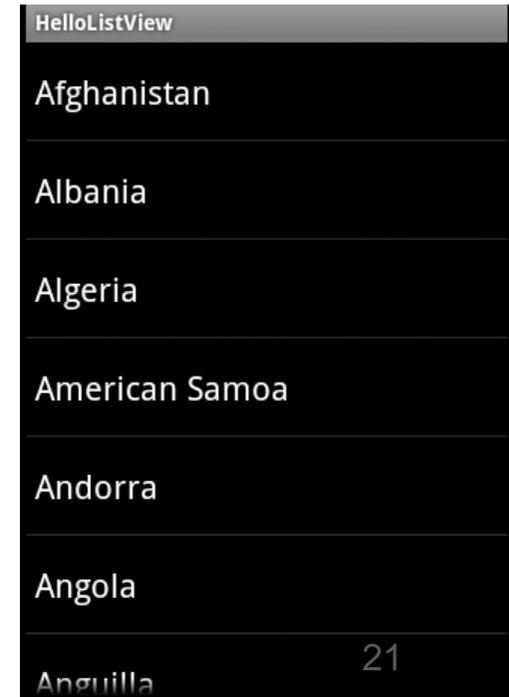
# Sélection dans une ListView :

## l'AndroidManifest.xml

- ❑ Lorsqu'on lance l'application, le focus est mis sur le premier composant graphique, ici l'EditText
- ❑ Dans ce cas, le clavier logiciel est immédiatement affiché
- ❑ Pour éviter cet affichage immédiat du clavier logiciel, il est bon d'ajouter l'attribut `android:windowSoftInputMode="stateHidden"` dans l'élément `activity` de l'activité principale

# Views "standards" pour les items d'une ListView

- ❑ Le second argument du constructeur `ArrayAdapter` est l'identificateur du fichier xml décrivant l'aspect de chaque item de la `ListView` (aspect qui peut être ensuite enrichi par la programmation)
- ❑ Certains de ces aspects sont donnés par android lui même ce sont : `android.R.layout.simple_list_item_1`
- ❑ Si, dans le programme précédent, on remplace par `R.layout.list_item` (notre propre fichier XML) par `android.R.layout.simple_list_item_1` (une des IHM standard d'android), on obtient :



# L'aspect

`android.R.layout.simple_list_item_1`

❑ Il y a peu de différence avec notre `list_item.xml`

❑ En fait le fichier associé à l'identificateur

`android.R.layout.simple_list_item_1` est :

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:gravity="center_vertical"
    android:paddingLeft="6dip"
    android:minHeight="?android:attr/listPreferredItemHeight"
/>
```

❑ Voir à `REP_INSTALL_ANDROID\android-sdk\platforms\android-XXX\data\res\layout\simple_list_item_1.xml`

❑ Ce fichier est encodé dans `android.jar`

❑ Bref les items sont des `TextView`

# Utiliser une ListView dans une Activity

- La ListView est indiquée dans un fichier XML d'IHM

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

- Dans le code Java, on récupère cette ListView (par la méthode `findViewById()`) et on l'alimente d'items par :

```
public class MainActivity extends Activity ... {
    public void onCreate(Bundle savedInstanceState) {
        ...
        ListView laListe = (ListView) findViewById(R.id.listMenu);
        Resources res = getResources();
        String [] lesItems = new String[] {
            res.getString(R.string.seeAllStations),
            res.getString(R.string.aboutUs)
        };
        laListe.setAdapter(new ArrayAdapter<String>(this, R.layout.list_item, lesItems));
    }
}
```

- `seeAllStations` et `aboutUs`, ... ont été définies dans `res/values/strings.xml`

# ListView : la gestion des événements

- ❑ La gestion des événements est assurée par l'auditeur de la `ListView`  
`public void setOnItemClickListener`  
(`AdapterView.OnItemClickListener listener`)

- ❑ Exemple :

```
ListView laListe = ...  
laListe.setOnItemClickListener(...);
```

- ❑ `AdapterView.OnItemClickListener` est une interface qui demande d'implémenter la méthode  
`public void onItemClick(AdapterView<?> parent, View view, int position, long id)`  
qui est lancée lorsque l'utilisateur sélectionne un item de la `ListView`
  - ❑ `parent` est l'`AdapterView` qui a été utilisé lors de l'événement
  - ❑ `view` est la `View` (c'est à dire l'item à l'intérieur de la `ListView`) qui a été utilisé
  - ❑ `position` est le numéro de l'item affiché dans la `ListView`. La valeur 0 est celle du premier item affiché
  - ❑ `id` est l'identificateur de l'item sélectionné. Dans ce cas il faut définir la méthode  
`public abstract long getItemId (int position)` provenant de l'interface `Adapter` pour l'adapteur associé à la `ListView` (cf. plus loin)

# Une question Java

- ❑ Pourquoi la méthode

```
public void setOnItemClickListener  
(AdapterView.OnItemClickListener listener)
```

qui demande comme argument un

`AdapterView.OnItemClickListener` peut être utilisé par le code :

```
lv.setOnItemClickListener(new OnItemClickListener() {  
... });
```

?

- ❑ Réponse : Car l'environnement de développement a ajouté le bon import  
`import android.widget.AdapterView.OnItemClickListener;`

- ❑ Merci Fabrice !

# Exercice



- ❑ Construire une activité présentant un `ListView` (1ère étape)
- ❑ Gérer les affichages de l'écran suivant le choix de l'utilisateur des différents items de cette `ListView` (2ième étape)

# Afficher des objets dans une ListView

- ❑ On a une collection d'objets, on voudrait les afficher "joliment" dans une ListView
- ❑ Par exemple une liste d'UE constituant un diplôme
- ❑ Une démo : Projet Ch3bisCertifCompet, onglet "Unités d'enseignement"
- ❑ Il s'agit de faire afficher une `ArrayList<UE>`

avec

```
public class UE {  
    private String nom, lieu, periode;  
    // accesseurs set/get pour ces champs  
    ...  
}
```

constituant un diplôme

```
public class DiplomeCNAM {  
    private ArrayList<UE> lesUE;  
    public DiplomeCNAM() {  
        this.lesUE = new ArrayList<UE>();  
        ...  
    }  
    public ArrayList<UE> getLesUE() {return lesUE; }  
}
```

© JMF (Tous droits réservés)



# Adapteur d'objets pour une ListView

- ❑ On construit un Adapter pour la ListView en dérivant de BaseAdapter
- ❑ Il faut donc donner un corps aux méthodes abstraites des classes ancêtres :

```
public class LeBelAdapter extends BaseAdapter {
    private DiplomeCNAM diplome;
    ...

    public int getCount() {
        return diplome.getLesUE().size();
    }

    public Object getItem(int position) {
        return diplome.getLesUE().get(position);
    }

    public long getItemId(int position) {
        return position;
    }

    public View getView(int position, View convertView, ViewGroup parent) {
        // Détaillé plus loin
    }
}
```

# L'IHM de chaque item (1/3)

- Comme d'habitude, l'IHM de chaque item est décrit par le fichier `res/layout/uecnam.xml` :

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

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:text="TextView" />
```

- D'abord un `TextView` de taille 16sp, puis ...

# L'IHM de chaque item (2/3)

```
<TableRow
  android:layout_width="match_parent"
  android:layout_height="wrap_content" >

  <ImageView
    android:id="@+id/imageId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/orange01" >
  >
</ImageView>

  <TextView
    android:id="@+id/lieuId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TextView"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textSize="12sp" >
  </TextView>
</TableRow>

</LinearLayout>
```

# L'IHM de chaque item (3/3)

- ❑ Le TableRow contient une ImageView et un TextView
- ❑ L'ImageView contient l'image orange01 comme valeur par défaut
- ❑ Le TextView contient TextView comme texte par défaut
- ❑ L'apparence du texte est l'apparence standard de l'API Android, textAppearanceMedium qui est défini par le fichier platforms\android-X\data\res\values\themes.xml du sdk qui contient :

```
<item name="textAppearanceLarge">@android:style/TextAppearance.Large</item>
<item name="textAppearanceMedium">@android:style/TextAppearance.Medium</item>
<item name="textAppearanceSmall">@android:style/TextAppearance.Small</item>
```

qui redirige en fait sur \platforms\android-X\data\res\values\styles.xml qui contient :

```
<style name="TextAppearance.Medium">
  <item name="android:textSize">18sp</item>
  <item name="android:textStyle">normal</item>
  <item name="android:textColor">?textColorPrimary</item>
</style>
```

- ❑ source : <http://stackoverflow.com/questions/11590538/dpi-value-of-default-large-medium-and-small-text-views-android>

# La méthode `getView()` de l'adaptateur

- ❑ C'est cette méthode qui construit la `View` de chaque item de la `ListView`
- ❑ Elle provient de l'interface `Adapter`. Elle est d'ailleurs signée dans cette interface  

```
public abstract View getView (int position, View convertView, ViewGroup parent)
```
- ❑ Elle retourne la `View` associé au `(position + 1)` item de la `ListView`. `convertView` est une référence sur une `View` à éventuellement réutiliser (voir plus loin). `parent` est la `View` parente à laquelle la `View` créé est attachée

# Notre méthode getView( ) (1/2)

- ❑ C'est cette méthode qui construit la View de chaque item de la ListView
- ❑ On peut l'écrire ainsi :

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

    View itemView = inflater.inflate(R.layout.uecnam, parent, false);
    TextView nomView = (TextView)itemView.findViewById(R.id.textView1);
    ImageView imageView = (ImageView)itemView.findViewById(R.id.imageId);
    TextView lieuView = (TextView)itemView.findViewById(R.id.lieuId);

    nomView.setText(diplome.getLesUE().get(position).getNom());
    lieuView.setText(diplome.getLesUE().get(position).getLieu());
    String periode = diplome.getLesUE().get(position).getPeriode();
    if (periode.equals("1")) {
        imageView.setImageResource(R.drawable.orange01);
    } else {
        imageView.setImageResource(R.drawable.green02);
    }
    return itemView;
}
```

# Notre méthode `getView()`

## (2/2)

- ❑ Cette méthode récupère l'analyseur XML (un `LayoutInflater`) de fichier d'IHM, puis chacun des composants graphiques et les positionne avec des valeurs précises, fonction du numéro de l'item de la `ListView` associée à cet adaptateur
- ❑ Remarques (bidouilles ?) sur `public View inflate (int resource, ViewGroup root, boolean attachToRoot)` du `LayoutInflater`
  - ❑ la `View` retournée est la racine de l'IHM fabriquée
  - ❑ `root` est la `View` parent de la hiérarchie d'IHM générée si `attachToRoot` est `true`. Dans ce cas c'est la valeur retournée. Si `attachToRoot` est `false` (= on n'attache pas), c'est (simplement) un objet qui passe ses paramètres `LayoutParams` pour la racine de la hiérarchie d'IHM générée
- ❑ On a ici un exemple de construction graphique en partie statique (à l'aide d'un fichier XML) pour les valeurs générales, et en partie dynamique (à l'aide de la programmation) pour les parties variables

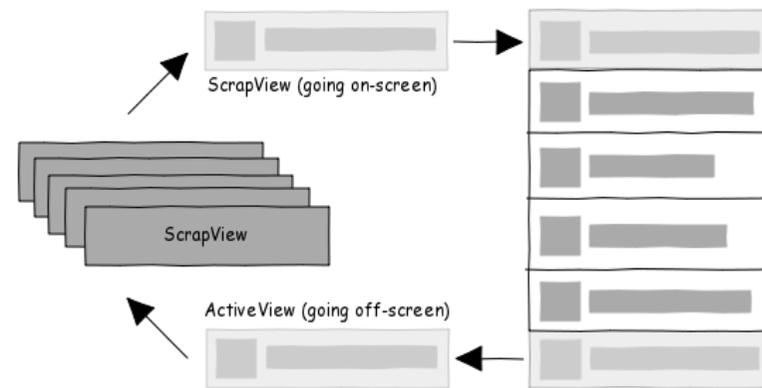
# Un cache pour la construction des items

- ❑ Le problème avec l'écriture de `getView()` précédente est que cette méthode est lancée pour chaque item
- ❑ Et donc l'analyse XML du fichier d'IHM est faite pour chaque item
- ...
- ❑ alors que cela donne toujours le même résultat !
- ❑ Les parties de programmation communes ne doivent être faites qu'une seule fois
- ❑ On utilise un cache :

```
class CacheView {  
    private TextView nomView;  
    private ImageView imageView;  
    private TextView lieuView;  
  
    // accesseurs set/get pour ces champs  
    ...  
}
```

# Le fonctionnement d'une ListView

- Une `ListView` garde trace des `View` de chaque item déjà construit. C'est une réserve de "ScrapView" :



"ScrapViews are old views that could potentially be used by the adapter to avoid allocating views unnecessarily"

([https://github.com/android/platform\\_frameworks\\_base/blob/43ee0ab8777632cf171b598153fc2c427586d332/core/java/android/widget/AbsListView.java#L5764](https://github.com/android/platform_frameworks_base/blob/43ee0ab8777632cf171b598153fc2c427586d332/core/java/android/widget/AbsListView.java#L5764))

- **source** : <http://lucasr.org/2012/04/05/performance-tips-for-androids-listview/>

# Association de données à une View

- ❑ On peut donc réutiliser si possible les ScrapViews
- ❑ Lorsque `getView()` est lancé, l'argument `convertView` repère éventuellement une ScrapView qui correspond à l'item à afficher. Si c'est le cas, inutile de refaire l'analyse XML. Si ce n'est pas le cas, on construit la vue de cet item et on lui associe des données particulières
- ❑ On associe des données supplémentaires à une View à l'aide de sa méthode `public void setTag (Object tag)` (de la classe View)
- ❑ On récupère des données supplémentaires à une View à l'aide de sa méthode `public Object getTag()` (de la classe View)
- ❑ Il suffit alors, la fois suivante, si on a récupéré une ScrapView pour l'item, de rechercher ces données

# Réécriture de getView()

□ D'où le code "optimisé" de getView() (et merci jean-michel)

```
public View getView(int position, View convertView, ViewGroup parent) {
    View itemView = convertView;
    if (convertView == null) {
        LayoutInflater inflater =
(LayoutInflater)ctx.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        itemView = inflater.inflate(R.layout.uecnam, parent, false);
        CacheView cache = new CacheView();
        cache.setNomView((TextView)itemView.findViewById(R.id.textView1));
        cache.setImageView((ImageView)itemView.findViewById(R.id.imageId));
        cache.setLieuView((TextView)itemView.findViewById(R.id.lieuId));
        itemView.setTag(cache);
    }
    // on récupère l'IHM déjà construite par analyse du fichier XML d'IHM
    CacheView cache = (CacheView)itemView.getTag();
    // on positionne les valeurs des composants graphiques
    // (au cas où ces valeurs auraient changées)
    cache.getNomView().setText(diplome.getLesUE().get(position).getNom());
    cache.getLieuView().setText(diplome.getLesUE().get(position).getLieu());
    String periode = diplome.getLesUE().get(position).getPeriode();
    if (periode.equals("1")) {
        cache.getImageView().setImageResource(R.drawable.orange01);
    } else {
        cache.getImageView().setImageResource(R.drawable.green02);
    }
    return itemView;
}
```

# La "UI Thread" (1/2)

- ❑ Lorsqu'une application Android est lancée, un seul processus est créé qui contient une seule thread pour l'application
- ❑ Cette thread est dite la thread principale
- ❑ Elle s'occupe, entre autre, de l'affichage et de l'interaction sur les divers écrans
- ❑ Voilà pourquoi cette thread principale est appelée la UI Thread (User Interface Thread) : "As such, the main thread is also sometimes called the UI thread."
- ❑ source :  
`http://developer.android.com/guide/components/processes-and-threads.html`

# La "UI Thread" (2/2)

- ❑ De plus, dans une application Android, il existe une et une seule thread qui gère l'interface graphique : la UI Thread (User Interface Thread)
- ❑ Tout ce qui concerne l'affichage est (et doit être) géré par cette thread. Si une autre thread s'occupe de faire de l'affichage graphique, il y a erreur à l'exécution
- ❑ Lorsqu'un travail demandant du temps est lancé, il faut le faire dans une thread autre que la UI Thread. Au besoin en crée une !
- ❑ Mais lorsque cette autre thread demande à afficher dans l'IHM, cette autre thread doit contacter l'UI Thread !

# Gérer la UI Thread

❑ Pour lancer du code dans la UI Thread à partir d'une autre thread, il existe plusieurs techniques :

❑ `Activity.runOnUiThread(Runnable)`

❑ `View.post(Runnable)`

❑ `View.postDelayed(Runnable, long)`

❑ Les Handler

❑ La classe `AsyncTask`

❑ Par exemple on peut écrire :

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            // a potentially time consuming task
            final Bitmap bitmap =
                processBitMap("image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

❑ source :

<https://developer.android.com/guide/components/processes-and-threads.html#Processes> © JMF (Tous droits réservés)

# La classe

```
android.os.AsyncTask<Params, Progress, Result>
```

- ❑ Au lieu de créer de telles threads et de les gérer, Android propose une classe qui gère correctement cela. C'est la classe (générique et abstraite !)

```
android.os.AsyncTask<Params, Progress, Result>
```

- ❑ Elle est faite pour gérer correctement la gestion entre une thread (de traitement long) et la UI Thread
- ❑ Elle permet de lancer un traitement en arrière plan et de mettre à jour (éventuellement de temps en temps) l'interface graphique de l'application
- ❑ Les paramètres génériques sont utilisés pour :
  - ❑ `Params` : le type des paramètres nécessaires à la tâche en arrière plan
  - ❑ `Progress` : le type des paramètres reçus par la méthode de mise à jour
  - ❑ `Result` : le type du résultat retourné par la tâche en arrière plan

# Les méthodes de AsyncTask<Params, Progress, Result> (1/2)

- ❑ Cette classe contient 4 méthodes dont trois d'entre elles sont exécutées dans la UI Thread. Ce sont :

```
protected void onPreExecute ()
```

```
protected void onProgressUpdate (Progress... values)
```

```
protected void onPostExecute (Result result)
```

qui sont exécutées dans la UI Thread et

```
protected abstract Result doInBackground (Params... params) lancée dans une autre thread
```

- ❑ `onPreExecute()` est exécutée avant le lancement de la tâche d'arrière plan

## Les méthodes de `AsyncTask<Params, Progress, Result>` (2/2)

- ❑ `doInBackground(Params...)` est lancée après `onPreExecute()` dans une thread autre que la UI Thread : c'est le traitement en arrière plan. Les paramètres de type `Params` sont passés à cette méthode. Cette méthode retourne un résultat de type `Result` (qui sera exploité par `onPostExecute(Result)`)
- ❑ Cette méthode peut lancer `publishProgress(Progress...)` qui permet de lancer `onProgressUpdate(Progress...)` dans la UI Thread et mettre ainsi à jour l'IHM de l'application, tout en continuant la tâche en arrière plan
- ❑ `onPostExecute(Result)` est lancée après la fin de la tâche en arrière plan. Cette méthode reçoit le paramètre de type `Result` qu'a renvoyé `doInBackground(Params...)`

# Utilisation de

`AsyncTask<Params, Progress, Result>`

- ❑ Pour utiliser cette classe, il faut la sous classer, construire une instance de cette sous-classe et lancer la méthode `execute(Params ...)` sur cette instance
- ❑ Il faut au moins redéfinir la méthode `doInBackground(Params...)` (bon sens !)
- ❑ La sous classe créée est souvent une sous-classe interne à une méthode
- ❑ La création de l'instance de cette sous classe doit être faite dans la UI Thread et `execute(Params ...)` doit être lancée sur cette instance une seule fois dans la UI Thread
- ❑ Les méthodes `onPreExecute()`, `doInBackground(Params...)`, `onProgressUpdate(Progress...)`, `onPostExecute(Result)` ne doivent pas être lancées explicitement : c'est la méthode `execute(Params ...)` qui déclenche le tout

# Les paramètres génériques de

`AsyncTask<Params, Progress, Result>`

- ❑ On lance la méthode `execute(Params ...)` qui passe la valeur de type `Params` à
- ❑ `Result doInBackground(Params...)`. La valeur de type `Result` retournée est passé à
- ❑ `onPostExecute(Result)`
- ❑ Rappel : éventuellement `doInBackground()` appelle `publishProgress(Progress...)` qui lance `onProgressUpdate(Progress...)` dans la UI Thread

# Exemple de code de

`AsyncTask<Params, Progress, Result>`

```
class ChargementDesStationsTache extends AsyncTask<Void, Void, Void> {
    protected void onPreExecute() {
        progress = ProgressDialog.show(ListingDesStationsActivity.this,
            getResources().getString(R.string.app_name),
            getResources().getString(R.string.chargement_message),
            true);
    }

    protected Void doInBackground(Void ... bidon) {
        // charger les stations
        try {
            sp = new StationsParser();
        } catch (Exception e) { e.printStackTrace(); }
        leStationsAdapter = new StationsAdapter(getBaseContext(),
            sp.getArrList() );

        return null;
    }

    protected void onProgressUpdate(Void... aAfficher) { }

    protected void onPostExecute(Void result) {
        // arrêter le progressDialog
        progress.dismiss();
        // mettre à jour la ListView des stations
        listing.setAdapter(leStationsAdapter);
    }
}

new ChargementDesStationsTache().execute();
```

# getBaseContext ( ) **vs.** getApplicationContext ( )

- ❑ On a utilisé `getBaseContext ( )` dans la diapo précédente
- ❑ `getBaseContext ( )` (de la classe `android.content.ContextWrapper`) retourne le contexte de l'activité courante. C'est ici utile pour avoir le thème de cette activité est donc bien afficher les résultats dans la `ListView`
- ❑ `getApplicationContext ( )` (déclarée abstraite dans la classe abstraite `android.content.Context`), souvent utilisée, retourne le contexte de l'application. C'est toujours le même contexte retourné pendant toute l'exécution de l'application
- ❑ En fait, "getBaseContext ( ) should not be used. It use the context which is associated with the activity and could possible be destroyed when the activity is destroyed"
- ❑ source :  
<http://stackoverflow.com/questions/6854265/getapplicationcontext-getbasecontext-getapplication-getparent>

# Configuration de l'AndroidManifest.xml

- ❑ Comme l'application demande à utiliser l'internet, il faut l'indiquer dans le fichier `AndroidManifest.xml` par :

```
<uses-permission android:name="android.permission.INTERNET" />
```

fil de l'élément `manifest`

- ❑ Comme l'application va utiliser une nouvelle Activity, il faut l'indiquer dans le fichier `AndroidManifest.xml` par :

```
<activity android:name=".ListingDesStationsActivity" />
```

fil de l'élément `application`

- ❑ Par exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="..." >
...
    <uses-permission android:name="android.permission.INTERNET" />

    <application ...
        ...
        <activity android:name=".ListingDesStationsActivity" />
    </application>

</manifest>
```

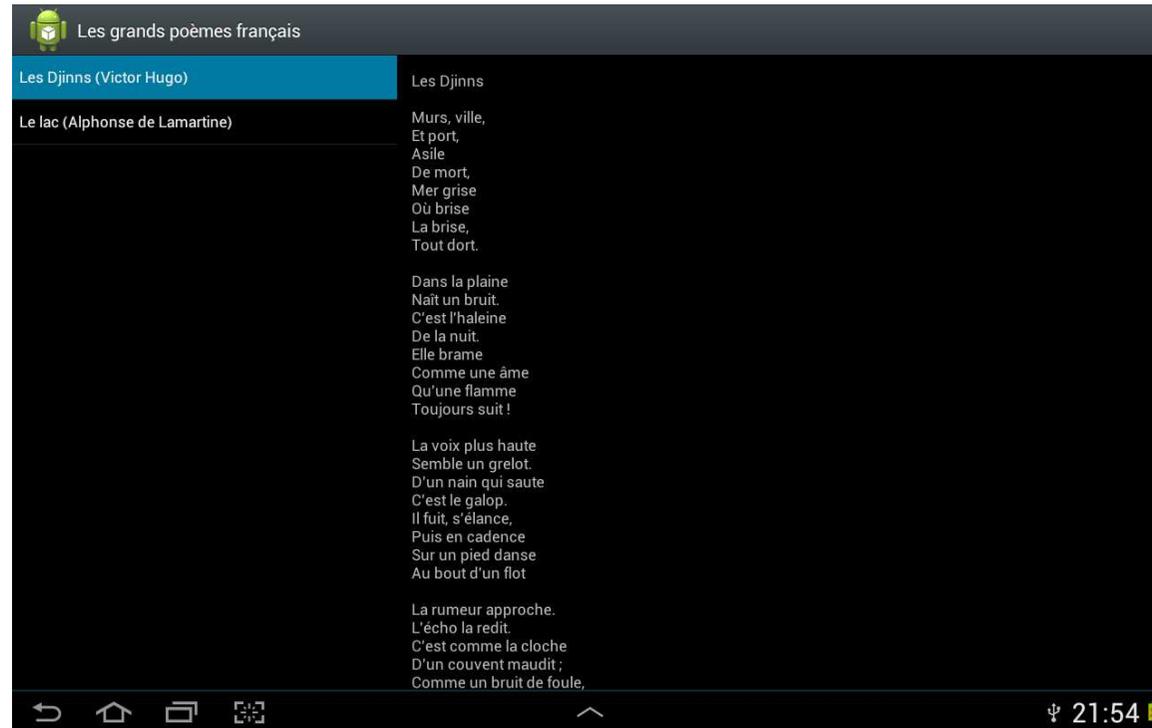
# Exercice



- ❑ Ecrire une activité contenant une `ListView` alimentée par des données se trouvant à une URL

# Les fragments c'est pour quoi ?

□ C'est pour ça :



□ Démo dans ...DevMobiles\Android\Travail, projet  
FragmentsBasisJMFProjet fortement inspiré de FragmentsBasis  
à  
<https://developer.android.com/training/basics/fragments/index.html>

# Plus précisément

- ❑ Ici, l'activité gère deux parties, deux portions (= deux fragments) d'écran
- ❑ Une activité peut contenir de multiples fragments
- ❑ Les fragments peuvent être utilisés dans plusieurs activités



# Fichier IHM de l'activité principale

- Le fichier d'IHM `news_articles.xml` chargé par l'activité principale `MainActiviy` est :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:name="com.example.android.fragments.TitresFragment"
        android:id="@+id/headlines_fragment"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

    <fragment android:name="com.example.android.fragments.PoemeFragment"
        android:id="@+id/article_fragment"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

</LinearLayout>
```

- Les fragments sont des instances des classes `TitresFragment` et `PoemeFragment`

# La classe TitresFragment : IHM

- Pour commencer, elle est :

```
public class TitresFragment extends ListFragment {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setListAdapter(new ArrayAdapter<String>(getActivity(), layout,
            TitreEtPoemes.TitresDesPoemes));
    }
    ...
}
```

- Bref c'est un fragment qui est une sorte de `ListView`. D'où son adapter fournissant l'IHM des items (`layout`) et les données `TitreEtPoemes.TitresDesPoemes`

- Dans cette classe, on a :

```
public class TitreEtPoemes {
    static String[] TitresDesPoemes = {
        "Les Djinns (Victor Hugo)",
        "Le lac (Alphonse de Lamartine)"
    };
    ...
}
```

# La classe TitresFragment : la gestion des événements

- ❑ Lorsqu'un item de cette liste est sélectionné par l'utilisateur (c'est un titre de poème), le poème doit être affiché dans le fragment 2
- ❑ La classe TitresFragment est complétée par :

```
private SelectionDUnTitreListener mCallback;
// L'activité contenant les deux fragments devra implémenter cette interface.
public interface SelectionDUnTitreListener {
    /** Devra mettre à jour PoemeFragment quand un titre est choisi */
    public void lorsDeLaSelectionDUnTitre(int position);
}

@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    mCallback = (SelectionDUnTitreListener) activity;
}

@Override
public void onItemClick(ListView l, View v, int position, long id) {
    mCallback.lorsDeLaSelectionDUnTitre(position);
}
}
```

- ❑ La méthode `onAttach()` est lancée lorsque le fragment est mis dans l'activité par l'environnement d'exécution

# L'activité principale

## MainActivity

❑ Finalement c'est :

```
public class MainActivity extends FragmentActivity
    implements TitresFragment.SelectionDUnTitreListener {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);
    }

    public void lorsDeLaSelectionDUnTitre(int position) {
        PoemeFragment articleFrag =
(PoemeFragment)getSupportFragmentManager().findFragmentById(R.id.article_fragment);
        articleFrag.updateArticleView(position);
    }
}
```

- ❑ On comprend alors que, lorsque l'utilisateur choisi un titre de poème, celui-ci est affiché dans le second fragment
- ❑ N'est ce pas ? ;-). OK, diapos suivantes !

# Explication (1/2)

- ❑ L'activité principale implémente `SelectionDUnTitreListener`
- ❑ Elle est repérée dans le fragment `TitresFragment` par

```
private SelectionDUnTitreListener mCallback;
```

qui a été initialisé par :

```
public void onAttach(Activity activity) {  
    super.onAttach(activity);  
    mCallback = (SelectionDUnTitreListener) activity;  
}
```

- ❑ Lorsque l'utilisateur sélectionne un item de liste la méthode

```
public void onItemClick(ListView l, View v, int position, long id) {  
    mCallback.lorsDeLaSelectionDUnTitre(position);  
}
```

est lancée (cf. les `ListView`)

- ❑ Le code de la méthode `lorsDeLaSelectionDUnTitre(int position)` de l'activité est essentiellement :

```
PoemeFragment lePoemeFragment =  
(PoemeFragment) getSupportFragmentManager().findFragmentById(R.id.article_fragment);  
lePoemeFragment.updateArticleView(position);
```

# Explication (2/2)

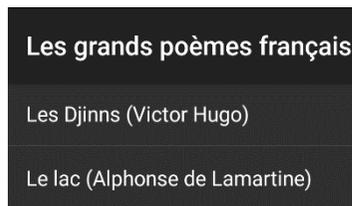
- ❑ La classe `PoemeFragment` contient :

```
public void updateArticleView(int position) {  
    TextView article = (TextView) getActivity().findViewById(R.id.article);  
    article.setText(TitreEtPoemes.Poemes[position]);  
}
```

- ❑ Rappel : `public View findViewById(int id)` est une méthode de la classe `Activity` qui "Finds a view that was identified by the id attribute from the XML that was processed in `onCreate(Bundle)`."

# Smartphone, tablette et fragment

- Et si on veut faire une seule application qui présente l'interface précédente sur une tablette et l'interface



donne

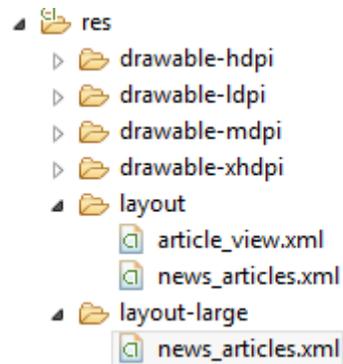


sur un smartphone ?

- C'est évidemment possible !

# Smartphone, tablette et fragment : architecture

- ❑ On a tout d'abord, sous `res`, deux fichiers `news_articles.xml`



- ❑ Le fichier `news_articles.xml` sous `layout-large` est celui chargé pour les tablettes, l'autre sous `layout`, est celui chargé pour les smartphones (fichier par défaut)
- ❑ cf. internationalisation
- ❑ On comprend qu'on fait alors la différence entre smartphone et tablette

# Les fichiers

## news\_articles.xml

- ❑ Celui sous `layout-large` est celui déjà indiqué diapo "Fichier IHM de l'activité principale"
- ❑ Celui sous `layout` est :

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

- ❑ C'est un conteneur `FrameLayout` destiné à recevoir dynamiquement un composant graphique

# onCreate() de l'activité principale

- ❑ Elle commence par :

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.news_articles);

    if (findViewById(R.id.fragment_container) != null) {
        TitresFragment firstFragment = new TitresFragment();
        getSupportFragmentManager().beginTransaction()
            .add(R.id.fragment_container, firstFragment).commit();
    }
}
```

- ❑ Le composant graphique d'id `R.id.fragment_container` est présent si on utilise un smartphone, pas si on utilise une tablette
- ❑ Pourquoi ?
- ❑ Le voir à partir des fichiers `news_articles.xml` chargés
- ❑ Donc si on est sur une tablette on a l'IHM du début, sinon, sur un smartphone, l'application crée une instance de `TitresFragment` et la met dans le `FrameLayout` d'id `R.id.fragment_container`

# Réaction aux événements (1/3)

- ❑ Rappel : lorsque l'utilisateur clique sur un item de la liste, la méthode `public void lorsDeLaSelectionDUnTitre(int position)` de l'activité principale est lancée
- ❑ L'essentiel du code de cette méthode est, en fait :

```
public void lorsDeLaSelectionDUnTitre(int position) {
    PoemeFragment lePoemeFragment =
(PoemeFragment)getSupportFragmentManager().findFragmentById(R.id.article_fragment);

    if (lePoemeFragment != null) {
        lePoemeFragment.updateArticleView(position);
    } else {
        PoemeFragment newFragment = new PoemeFragment();
        Bundle args = new Bundle();
        args.putInt(PoemeFragment.ARG_POSITION, position);
        newFragment.setArguments(args);
        FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();

        transaction.replace(R.id.fragment_container, newFragment);
        transaction.addToBackStack(null);
        transaction.commit();
    }
}
```

# Réaction aux événements

## (2/3)

- ❑ La référence `lePoemeFragment` est différente de `null` si un composant graphique (un fragment) d'id `R.id.article_fragment` a été trouvé
- ❑ C'est le cas pour une tablette, n'est ce pas !
- ❑ Le voir sur le fichier `res/layout-large/new_articles.xml`
- ❑ Il désigne le fragment de droite
- ❑ Sur ce composant graphique est lancée la méthode `updateArticleView(position)` qui met à jour ce fragment

# Réaction aux événements

## (3/3)

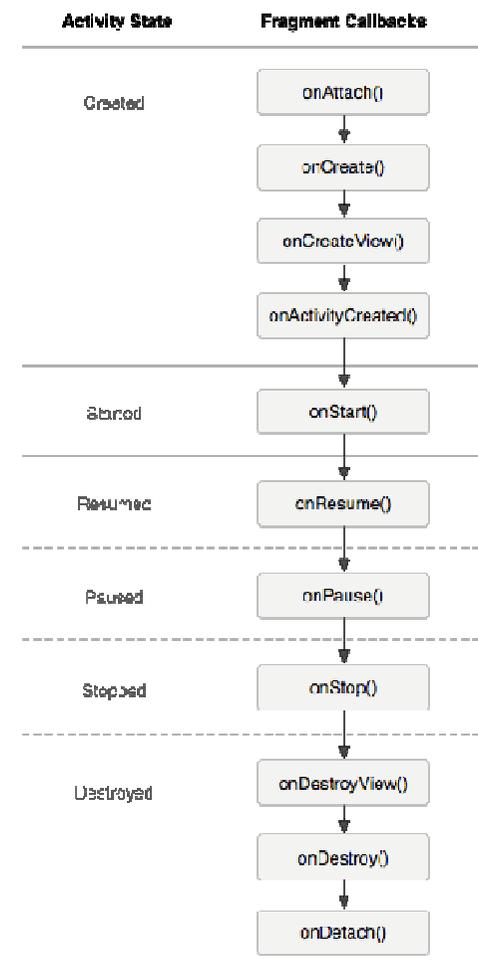
- ❑ Il n'y a pas de composant graphique d'id `R.id.article_fragment` sur un smartphone. L'IHM est alors celui du `PoemeFragment` a qui on a passé l'indice position pour afficher le bon poème
- ❑ D'ailleurs la classe `PoemeFragment` a la méthode :

```
public void onStart() {
    super.onStart();
    // On vérifie alors si des arguments ont été passés pour afficher ce fragment
    // Si oui, celui repéré par ARG_POSITION indique le numéro du poème
    // à afficher
    Bundle args = getArguments();
    if (args != null) {
        updateArticleView(args.getInt(ARG_POSITION));
    }
    ...
}
```

- ❑ Il faut aussi mettre dans la "back stack" cette manipulation pour pouvoir revenir à la situation précédente en cas d'appui du bouton back par l'utilisateur

# Le cycle de vie d'un fragment

- ❑ La vie d'un fragment est lié à celle de l'activité qui le contient
- ❑ biblio : <http://developer.android.com/guide/components/fragments.html#Lifecycle>
- ❑ Le cycle de vie ci-contre montre les méthodes (callbacks) lancées dans un fragment suivant les états de son activité (contenante)



# Etats fondamentaux d'un fragment

- ❑ Essentiellement un fragment a trois états
- ❑ Dans l'état Resumed, le fragment est visible dans son activité
- ❑ Dans l'état Paused, une activité en premier plan et ayant le focus masque en partie l'activité du fragment
- ❑ Dans l'état Stopped, le fragment n'est pas visible (mais existe encore)

# Le fragment avec son activité

- ❑ Comme indiqué dans le diagramme, sont lancées dans cet ordre
- ❑ 1°) la méthode `onAttach()` du fragment est lancée lorsque le fragment est associé à l'activité
- ❑ 2°) la méthode `onCreate()` du fragment est lancée lorsque le fragment est créé
- ❑ 3°) la méthode `onCreateView()` du fragment est lancée pour construire l'IHM du fragment
- ❑ 4°) après l'exécution de la méthode `onCreate()` de l'activité, la méthode `onActivityCreated()` du fragment est lancée
- ❑ De manière similaire, lorsque l'IHM du fragment est détruit, la méthode `onDestroyView()` du fragment est lancée
- ❑ Lorsque le fragment n'est plus associé à l'activité, la méthode `onDetach()` du fragment est lancée
- ❑ biblio :  
<http://developer.android.com/reference/android/app/Fragment.html#Lifecycle>

# Bibliographie pour les fragments

- <http://developer.android.com/guide/components/fragments.html>
- <http://developer.android.com/training/basics/fragments/index.html>  
pour FragmentsBasicsProjet
- <http://www.vogella.com/tutorials/AndroidFragments/article.html>

# Bibliographie pour ce chapitre

- [1] : Pour ce chapitre

<http://developer.android.com/guide/topics/ui/index.html>

- <http://www.vogella.com/articles/AndroidListView/article.html> : article de Lars Vogel sur les ListView

- Pour l'optimisation de ListView : polycopié de cours  
ListView\_Introduction de Jean-Michel Douin et

<http://lucasr.org/2012/04/05/performance-tips-for-androids-listview/>

# Résumé du chapitre IHM avancé

- ❑ Les composants graphiques qui affichent des ensembles (comme une `ListView`) sont des `AdapterView` gérés par des `Adapter`
- ❑ L'affichage des éléments peut être indiqué par des `View` "standards" ou en codant la méthode `getView()` de l'adapter
- ❑ La thread principale est celle qui gère l'affichage et est la `UI Thread`
- ❑ On ne peut faire des affichages graphiques que dans la `UI Thread`. La classe `AsyncTask` aide pour cela
- ❑ Pour gérer des parties d'écran, surtout utile pour les tablettes, on utilise les fragments



**Fin**