Les listes et les adaptateurs

ListView

ListView place les éléments en liste avec un ascenseur vertical si nécessaire. ListView est normalement utilisé pour afficher des éléments textuels éventuellement accompagnés d'une case à cocher lorsqu'il s'agit d'une liste à choix multiples.

Il est toutefois possible d'y afficher des éléments plus complexes en utilisant un gestionnaire de contenu.

Type de choix

android:choiceMode="c" (où c peut prendre les valeurs : none, singlechoice, multipleChoice) pour indiquer le mode de choix dans la liste (aucun, un seul, plusieurs).

Méthodes de la classe ListView : android.widget.ListView

Construction

ListView(Context) le paramètre est généralement l'activité elle-même

Contenu de la liste

Le contenu d'une ListView peut être défini de façon statique par la propriété android:entries dans le fichier xml. Lorsque l'on souhaite avoir un contenu dynamique on utilise un ArrayAdapter (collection) que l'on remplit (méthodes add ou insert du ArrayAdapter) et que l'on associe à la ListView par la méthode setAdapter(ArrayAdapter).

Remarque

Pour gérer un contenu autre que du texte il existe un gestionnaire (classe SimpleAdapter) pour lequel chaque élément de la liste est une petite interface définie dans un fichier xml.
On peut également écrire son propre gestionnaire de contenu en définissant une classe héritant de BaseAdapter.

Aspect de la liste

- setDivider(Drawable) définit l'image qui sera dessinée entre chaque élément de la liste
- getDivider() retourne l'image dessinée entre chaque élément de la liste
- setDividerHeight(int) définit la hauteur de la séparation entre chaque élément de la liste
- getDividerHeight() renvoie la hauteur de la séparation entre chaque élément de la liste
- addFooter(View) ajoute une vue en bas de la liste
- addHeader(View) ajoute une vue en haut de la liste
- setFooterDividersEnabled(boolean) autorise ou interdit le dessin d'un séparateur pour le bas de liste
- setHeaderDividersEnabled(boolean) autorise ou interdit la dessin d'un séparateur pour le haut de liste
- invalidateViews() provoque le rafraichissement des tous le éléments internes

Selection deséléments

- setItemChecked(int, boolean) sélectionne ou non un élément de liste par son rang (à partir de 0).
- isItemChecked(int) indique si l'élément de liste désigné par son rang (à partir de 0) est selectionné.
- clearChoices() invalide les choix actuels
- setChoiceMode(int) définit le mode de choix, le paramètre peut prendre les valeurs : none, singleChoice ou multipleChoice
- getChoiceMode() renvoie le mode de choix
- clearTextFilter() supprime le filtrage des éléments au fur et à mesure de la saisie
- getTextFilter() renvoie la chaîne de caractères utilisée pour le filtrage des éléments au fur et à mesure de la saisie
- isTextFilterEnabled()indique si le filtrage des éléments au fur et à mesure de la saisie est actif ou pas
- setFilterText(String) définit la chaîne de caractères utilisée pour le filtrage des éléments au fur et à mesure de la saisie
- setTextFilterEnabled(boolean) active ou désactive le filtrage des éléments au fur et à mesure de la saisie

Evénements

− setOnItemClickListener(AdapterView.OnItemClickListener) associe un écouteur d'événements au clic sur un élément. La méthode onItemClick(AdapterView<?>, View, int, long) de l'interface AdapterView.OnItemClickListener est implémentée pour traiter l'événement. Le 3ème paramètre indique le rang de l'élément et le dernier son identifiant.

− setOnItemLongClickListener(AdapterView.OnItemLongClickListener) associe un écouteur d'événements au clic long sur un élément. La méthode onItemLongClick(AdapterView<?>, View, int, long) de l'interface AdapterView.OnItemLongClickListener est utilisée pour traiter l'événement.

− setOnItemSelectedListener(AdapterView.OnItemSelectedListener) associe un écouteur d'événements à la sélection d'un élément (pour les listes à choix multiples). La méthode onItemSelected (AdapterView<?>, View, int, long) de l'interface AdapterView.OnItemSelectListener est utilisée pour traiter l'événement de sélection d'un élément.

I) Introduction

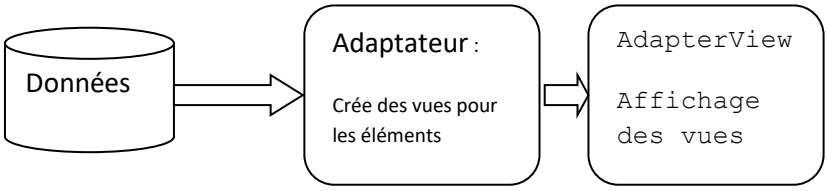
La gestion des listes se divise en deux parties, Les `Adapter` (**adaptateurs**) et les `AdapterView`.

- Les `Adapter` (**adaptateurs**), sont des objets qui gèrent les données, mais pas leur affichage ou leur comportement en cas d'interaction avec l'utilisateur.

On peut considérer un adaptateur comme un intermédiaire entre les données et la vue qui représente ces données.

- Les `AdapterView` gèrent l'affichage et l'interaction avec l'utilisateur, mais sur lesquels on ne peut pas effectuer les opérations de modification des données.

- Pour afficher une liste depuis un ensemble de données :
  1. On donne à l'adaptateur une liste d'éléments à traiter.
  2. l'adaptateur va créer une vue pour chaque élément en fonction des informations fournies avec le layout à respecter.
  3. Puis, les vues sont fournies à un `AdapterView` où elles seront affichées dans l'ordre fourni et avec le layout correspondant. L'`AdapterView` possède lui aussi un layout afin de le personnaliser.



Les adaptateurs

Les adaptateurs d'Android se chargent de fournir la liste des données d'un widget de sélection et de convertir les différents éléments en vues spécifiques pour qu'elles s'affichent dans ce widget de sélection.

Pour des widgets simples, on fait appel à trois adaptateurs principaux :

1. `ArrayAdapter` : permet d'afficher les informations simples ;
2. `SimpleAdapter` : utile dès qu'il s'agit d'écrire plusieurs informations pour chaque élément (s'il y a deux textes (nom, prénom) dans l'élément par exemple) ;

3. `CursorAdapter`, pour adapter le contenu qui provient d'une base de données.

Les AdapterView (Les vues responsables de l'affichage des listes)

On trouve la classe `AdapterView` dans le package `android.widget.AdapterView`.
L'adaptateur se charge de construire les sous-éléments.
Il lie ces sous-éléments et les affiche en une liste. De plus, c'est l'`AdapterView` qui gère les interactions avec les utilisateurs.

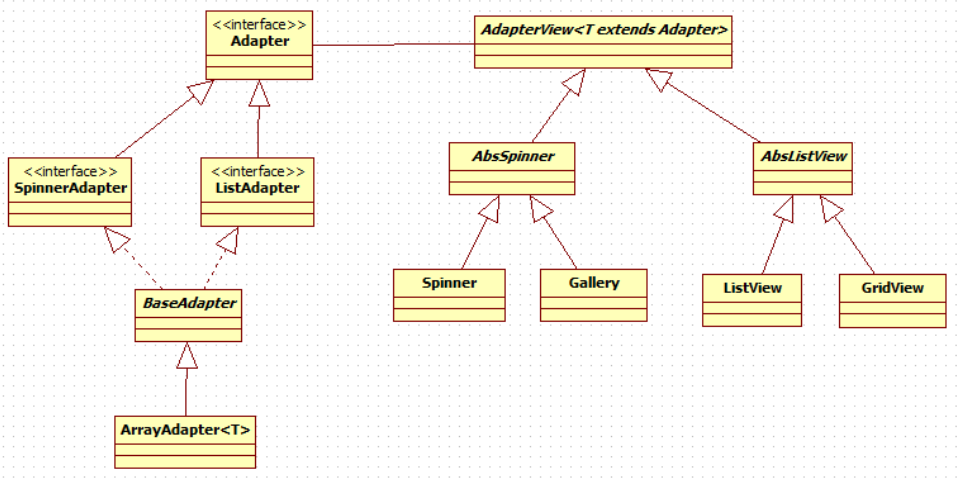
L'adaptateur s'occupe des éléments en tant que données, alors que **l'`AdapterView`** s'occupe de leur affichage et veille aux interactions avec un utilisateur.

On observe trois principaux `AdapterView` :

1. `ListView`, pour simplement afficher des éléments les uns après les autres ;
2. `GridView`, afin d'organiser les éléments sous la forme d'une grille ;
3. `Spinner`, qui est une liste défilante.

`android.widget.Adapter` est une interface. Beaucoup de classes implémentent cette interface comme `ArrayAdapter` et `BaseAdapter`.
`SpinnerAdapter` et `ListAdapter` sont des interfaces qui héritent de `Adapter`

Pour associer un `adapter` à un `AdapterView`, on utilise la méthode `void setAdapter (Adapter adapter)`.



II) Les listes simples : ArrayAdapter

La classe `ArrayAdapter` se trouve dans le package `android.widget.ArrayAdapter`.

Les constructeurs :

```
public ArrayAdapter (Context contexte, int id, T[] objects)
public ArrayAdapter (Context contexte, int id, List<T> objects).
```

- `contexte` : dans lequel l'adapter va fonctionner (généralement, l'instance de l'activité).
- `id` : l'identificateur du fichier xml décrivant l'aspect de chaque item de la ListView (cet aspect peut être ensuite enrichi par la programmation), il peut être identifiant d'une ressource système prédéfinie, comme : **`android.R.layout.simple_list_item_1`).**
- `objects` : la liste ou le tableau des éléments à afficher.

Le composant `ListView`

Il permet d'afficher une liste d'items, accessible par défilement mais aussi par filtre.

Lorsqu'on construit une classe qui hérite de ListActivity, on peut utiliser plusieurs méthodes de la classe ListActivity comme :

- public ListView **getListView**() qui retourne le composant graphique ListView associé à cette activité.
- public void **setListAdapter**(ListAdapter adapter) qui positionne le ListAdapter associé à la ListView de cette activité (ListActivity).

Pour associer un AdapterView à un Adapter pour la classe ListActivity on peut utiliser :

```
this.getListView().setAdapter(arrAdapter);
```

ou

```
this.setListAdapter(arrAdapter)
```

- ### Affichage avec une ListActivity

Exemple 1

Il faut ajouter un ListView avec l'identifiant : "@android:id/list".

Fichier main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout

<ListView
android:id="@android:id/list"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:drawSelectorOnTop="false" />
</LinearLayout>
```

Code Java

```
public class ListViewDemo extends ListActivity {
String[] prenoms={"ali", "aziz", "lina", "siham", "ahmed"};

@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
setContentView(R.layout.main);

ArrayAdapter<String> aa =new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, prenoms));
setListAdapter(aa);
//this.getListView().setAdapter(aa);
} }
```

Exemple2

Dans le programme précédent, on peut remplacer android.R.layout.simple_list_item_1 (une des IHM standard d'android) par R.layout.layout1 (notre propre fichier XML).

Pour définir un `ListView`, on commence par indiquer comment sera affiché chacun de ses items. Par exemple sous forme de `TextView`.
On construit donc un fichier `res/layout/layout1.xml` contenant cette `TextView`

Créer le fichier suivant layout1.xml

```
<?xml version="1.0" encoding="utf-8"?>

<TextView
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:padding="10dp"
android:textSize="16sp"
android:textColor=ff00ff>
</TextView>
```

Code java

```
public class MainActivity extends ListActivity   {
static String[] prenoms={"alia", "aziz", "lina", "siham", "ahmed"};
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
       // setContentView(R.layout.activity_main);
ArrayAdapter<String> aa =new ArrayAdapter<String>(this, R.layout.layout1,
prenoms);
setListAdapter(aa);
//this.getListView().setAdapter(aa);
}}
```

L'avantage d'utiliser une ListActivity par rapport à Activity est qu'on n'a pas besoin de récupérer l'instance du composant ListView dans l'IHM.

- ## Affichage sans ListActivity

Code XML

Créer une ListView dans le fichier xml :

```
<ListView
        android:id="@+id/list1"
        android:layout_height="wrap_content"
        android:layout_width="match_parent">
</ListView>
```

Code java

`ArrayAdapter(Context, id, type)` : le troisième paramètre est un type prédéfini :

- **android.R.layout.simple_list_item_1** pour une liste à choix unique
- **android.R.layout.simple_list_item_multiple_choice** pour une liste à choix multiple, (une case à cocher apparaît à coté de chaque élément de la liste)

```
public class ListViewAndroidExample extends Activity {
        ListView L;
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
```

```
            setContentView(R.layout.activity_main);
            L = (ListView) findViewById(R.id.list1);

         String[] donnees = new String[] { "Android List View",
         "Adapter", "Create List View", "Android Example", "List View
Source", "List View Array", "Android Example List" };

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

  // Le deuxième paramètre "android.R.layout.simple_list_item_1" permet d'indiquer la
//présentation utilisée pour les items de notre liste. Ici il s'agit d'une présentation simple pour les
//chaines de caractères incluse dans le SDK.
            L.setAdapter(adapter);
        }
    }
```

Séparateurs avec xml

- `android:divider` : définit la couleur des séparateurs ou utilise une image comme séparateur.
- `android:dividerHeight="unité"` : définit la hauteur des séparateurs

Type de choix `avec xml`

- `android:choiceMode="c"` (où c peut avoir les valeurs : none, singlechoice, multipleChoice) pour indiquer le mode de choix dans la liste (aucun, un seul, plusieurs)

Remarques

Pour remplir la liste, on peut utiliser la méthode ArrayAdapter.add(Object) ou obtenir les éléments de la liste à partir d'un fichier de ressource : **res/values/string.xml**.

```
<resources>
<string-array name="maliste">
<item>premier élément</item>
<item>deuxième élément</item>
…
<item>dernier élément</item>
</string-array>
</resources>
```

- Dans le fichier main.xml on remplit la propriété android:entries par la liste.

```
<ListView
   android:id="@+id/list1"
    android:entries="@array/maliste"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
</ListView>
```

- Dans le code java :

```
ListView L = (ListView) findViewById(R.id.List1);
```

```
//String[] elements = getResources().getStringArray(R.array.maliste);

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_multiple_choice);
//for (int i=0; i<elements.length; i++) adapter.add(elements[i]);
L.setAdapter(adapter);
```

Liste de choix : Spinner

Le Spinner est l'équivalent des listes déroulantes. Il affiche le choix actuel et présente un RadioGroup quand on clique dessus pour le changer.
La classe `Spinner` se trouve dans le package `android.widget.Spinner`.
On utilise deux vues, une pour l'élément sélectionné qui est affiché, et une pour la liste d'éléments sélectionnables.

Comme pour ListView, on fournit l'adaptateur pour les données et les vues filles via setAdapter().

Propriétés :

android:prompt : définit le titre de la fenêtre qui s'ouvre lorsque l'on fait un choix
android:entries="@array/maliste" définit le contenu de la liste à partir du contenu d'un fichier xml placé dans **res/values/** qui a la forme suivante :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string-array name="maliste">
<item>Mercure</item>
…
<item>Neptune</item>
</string-array>
</resources>
```

Si on souhaite personnaliser la vue d'affichage de la boîte déroulante, il faut configurer l'adaptateur non pas le Spinner. Pour ce faire, on a besoin de la méthode setDropDownViewResource() afin de fournir l'identifiant de la vue concernée.

Exemple

```
<TextView
android:id="@+id/selection"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>

<Spinner android:id="@+id/spinner"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:drawSelectorOnTop="true"
/>
```

La propriété **android:drawSelectorOnTop** indique que la flèche permettant de dérouler la sélection se trouvera à droite du Spinner.

Code java

```
public class SpinnerDemo extends Activity  implements AdapterView.OnItemSelectedListener {
TextView selection;
String[] items={"donnee1", " donnee2", " donnee3", " donnee4", " donnee5", " donnee6"};
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
setContentView(R.layout.main);
selection=(TextView)findViewById(R.id.selection);
Spinner spin=(Spinner)findViewById(R.id.spinner);
spin.setOnItemSelectedListener(this);

ArrayAdapter<String> aa=new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item,
items);
aa.setDropDownViewResource( android.R.layout.simple_spinner_dropdown_item);
spin.setAdapter(aa);
}
public void onItemSelected(AdapterView<?> p, View v, int position, long id) {
//selection.setText(items[position]);
  selection.setText(p.getItemAtPosition(position).toString());
}
public void onNothingSelected(AdapterView<?> p) {
selection.setText("");
} }
```

GridView

Ce type de liste fonctionne presque comme ListView.
Il met les éléments dans une grille dont il détermine automatiquement le nombre d'éléments par ligne.
Il est cependant possible de fixer ce nombre d'éléments par ligne à l'aide des propriétés :

- **android:numColumns** en XML ou **setNumColumns (int column)** en Java (si sa valeur est auto_fit, Android calcule ce nombre en fonction de l'espace disponible et de la valeur d'autres propriétés.
- **android:verticalSpacing** et **android:horizontalSpacing** précisent l'espace qui sépare les éléments de la grille.
- android:columnWidth indique la largeur de chaque colonne en pixels.

- android:stretchMode : (si android:numColumns =auto_fit) indique ce qui devra se passer lorsqu'il reste de l'espace non occupé par des colonnes ou des espaces de séparation.

Exemple

```
<GridView
android:id="@+id/liste_de_donnees"
android:entries ="@array/maliste"
android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent"
android:numColumns="2"
android:stretchMode="columnWidth"
android:columnWidth="60dp"
android:gravity="fill_horizontal"
android:choiceMode="multipleChoice"
/>
```

–code java

```
GridView table = (GridView) findViewById(R.id.liste_de_donnees);
String[] elements = getResources().getStringArray(R.array.maliste);

ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_multiple_choice);
for (int i=0; i<elements.length; i++) aa.add(elements[i]);
table.setAdapter(aa);
```

III) Des listes plus complexes : `SimpleAdapter`

Rappel

Différence entre **android.R.layout.simple_list_item_1 et android.R.layout.simple_list_item_2**

android.R.layout.simple_list_item_1 est représenté par le fichier suivant :

```
<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"
/>
```

android.R.layout.simple_list_item_2 est représenté par le fichier suivant :

```
<TwoLineListItem xmlns:android="http://schemas.android.com/apk/res/android"
android:paddingTop="2dip"
android:paddingBottom="2dip"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:minHeight="?android:attr/listPreferredItemHeight"
android:mode="twoLine">

<TextView android:id="@android:id/text1"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginLeft="6dip"
android:layout_marginTop="6dip"
android:textAppearance="?android:attr/textAppearanceLarge"/>
```

```
<TextView android:id="@android:id/text2"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_below="@android:id/text1"
android:layout_alignLeft="@android:id/text1"
android:textAppearance="?android:attr/textAppearanceSmall" />
</TwoLineListItem>
```

Dans ce cas, il y a deux TextViews (identifié par text1 et text2). On ne peut donc indiquer simplement comme précédemment :
ArrayAdapter<String>(this, android.R.layout.simple_list_item_1); qui provoque d'ailleurs une exception.
On doit indiquer ce que devra contenir text1 et text2 avec un **SimpleAdapter**.

SimpleAdapter

`SimpleAdapter` est utile pour afficher (un objet complexe) plusieurs informations par élément. Pour chaque information de l'élément on aura une vue dédiée qui affichera l'information voulue.

- Un tel objet est défini dans une HashMap<String, ?>. Les clés String sont les "noms de champs", la valeur associée est celle qui devra être représentée.

- On définit également deux tables parallèles qui permettront de faire la correspondance entre les "champs" de l'objet et les Views dans lesquelles ils seront affichés:
  - String[] contenant les noms des champs (clés d'accès à la HashMap).
  - int[] contenant les identifiants des vues (R.id).

- Finalement, on construit une List<HashMap<String, ?>> qu'on fournit au constructeur du SimpleAdapter associé à la ListView.

SimpleAdapter(Context context, List<? **extends** Map<String, ? >> data, int ressource, String[] from, int[] to)

Exemple

```
public class ListesActivity extends Activity {
ListView vue;
 public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    vue = (ListView) findViewById(R.id.listView);
       String[][] repertoire = new String[][]{
       {"Bill Gates", "06 06 06 06 06"},
       {"Niels Bohr", "05 05 05 05 05"},
       {"Alexandre ", "04 04 04 04 04"}};

List<HashMap<String, String>> liste=new ArrayList<HashMap<String,
String>>();
    HashMap<String, String> element;

    for(int i = 0;i<repertoire.length;i++){
      element = new HashMap<String, String>();
```

```
        element.put("text1", repertoire[i][0]);
        element.put("text2", repertoire[i][1]);
        liste.add(element);
       }

ListAdapter adapter = new SimpleAdapter(this, liste,
android.R.layout.simple_list_item_2, new String[]{"text1",
"text2"},new int[] {android.R.id.text1, android.R.id.text2});
vue.setAdapter(adapter);
  }}
```

Quelques méthodes communes à tous les adaptateurs

- `void add (T object)` : ajouter un objet à un adaptateur
- `void insert (T object, int position)`: insère un objet à une position particulière.
- `getCount()` renvoie le nombre d'éléments
- `T getItem (int position)` : récupérer un objet dont on connaît la position
- `int getPosition (T object)`: récupérer la position d'un objet
- `void remove (T object)`
- `void clear()`.

Les listes standards

Par défaut, ListView est configurée pour recevoir les clics sur les entrées de la liste. Cependant, on a parfois besoin qu'une liste mémorise un ou plusieurs choix de l'utilisateur. Il faut alors quelques modifications :

- Dans le code Java, appelez la méthode setChoiceMode() de l'objet ListView pour configurer le mode de sélection avec les paramètre CHOICE_MODE_NONE, CHOICE_MODE_SINGLE ou CHOICE_MODE_MULTIPLE

- Au lieu de passer en paramètre android.R.layout.simple_list_item_1 au constructeur de ArrayAdapter on utilise :
  - soit android.R.layout.simple_list_item_single_choice,
  - soit android.R.layout.simple_list_item_multiple_choice

Pour récupérer le rang de l'élément sélectionné
- pour une sélection unique : `int getCheckedItemPosition()`
- une sélection multiple:0
- `SparseBooleanArray getCheckedItemPositions()`.

Un `SparseBooleanArray` est un tableau associatif dans lequel on associe un entier à un booléen.
Pour vérifier la valeur associée à une clé on utilise la méthode `boolean get(int key)`.
`liste.getCheckedItemPositions().get(3)` : pour savoir si le troisième élément de la liste est sélectionné. Si le résultat vaut `true`, alors l'élément est bien sélectionné dans la liste.

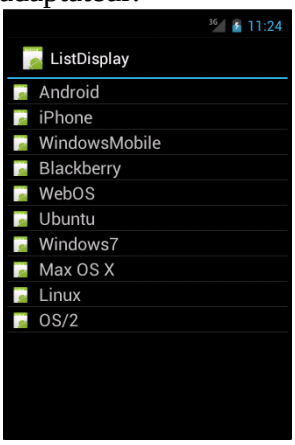
La gestion des événements

La gestion d'événements est assurée par `AdapterView.OnItemClickListener` lancée lorsque l'utilisateur sélectionne un item de la `ListView`
Cette interface demande d'implémenter la méthode
`public void onItemClick(AdapterView<?> parent, View view, int position, long id)`

- `parent` est l'`AdapterView` qui a été utilisé lors de l'événement
- `view` est la `vue` (c'est à dire l'item à l'intérieur de la `ListView`) qui a été utilisée.
- `position` est le numéro de l'item dans cette `ListView`. La valeur `0` est celle du premier item
- `id` est l'identificateur de l'item sélectionné

Les listes accompagnées des images

Pour avoir une liste avec des images à côté, il faut créer son propre Layout et l'utiliser comme adaptateur.



Pour cela créer un fichier xml (layout1.xml par exemple) comme suit :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
      <ImageView
             android:id="@+id/img"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:layout_gravity="center_vertical"
             android:padding="10px"
             android:src="@drawable/ic_launcher"
             />

       <LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:paddingLeft="10px"
```

```
            android:layout_weight="1">

        <TextView android:id="@+id/label"
                  android:layout_width="fill_parent"
                  android:layout_height="fill_parent"
                  android:textSize="16px"
                  android:textStyle="bold"
                  />
    </LinearLayout>
</LinearLayout>
```

Code java

public class MainActivity extends ListActivity {

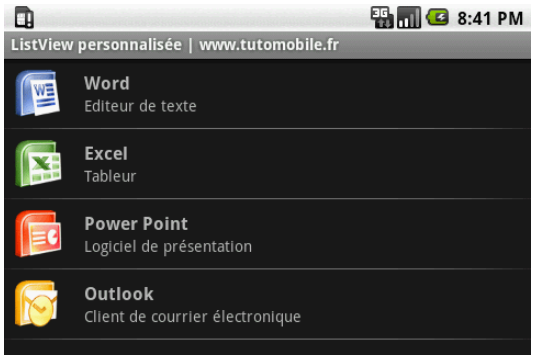
```
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    String[] values = new String[] { "Android", "iPhone", "WindowsMobile",
        "Blackberry", "WebOS", "Ubuntu", "Windows7", "Max OS X",
        "Linux", "OS/2" };

    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        R.layout.layout1, R.id.label, values);
    setListAdapter(adapter);
  }
  @Override
  protected void onListItemClick(ListView l, View v, int position, long id) {
    String item = (String) getListAdapter().getItem(position);
    Toast.makeText(this, item + " selected", Toast.LENGTH_LONG).show();
  }
}
```

Ajouter plusieurs images.

Pour cela il nous faut créer dans le projet un répertoire **drawable** dans les ressources puis mettre dedans les images voulues.

Android utilise des adaptateurs pour remplir les listes. Nous allons créer ici un adaptateur sous forme d'une classe Java. Ce dernier retournera à la liste d'activité chaque ligne à afficher.

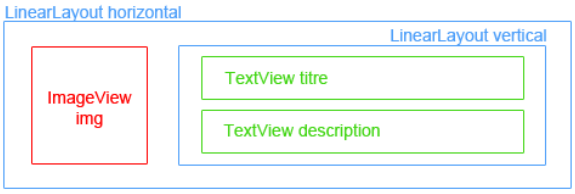
Exemple



Créer une *ListView* dans le fichier Activity*main.xml*,

```
<ListView
        android:id="@+id/listviewperso"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
      />
```

Créer un nouveau fichier XML, *affichageitem.xml*, qui va permettre de créer la vue qui affichera l'item dans la *ListView*.




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

    <ImageView
                android:id="@+id/img"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center_vertical"
                android:padding="10px"
                />

        <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
                android:layout_gravity="center_vertical"
                android:paddingLeft="10px"
            android:layout_weight="1"
            >
            <TextView android:id="@+id/titre"
                  android:layout_width="fill_parent"
                  android:layout_height="fill_parent"
                  android:textSize="16px"
                  android:textStyle="bold"
                  />
            <TextView android:id="@+id/description"
                  android:layout_width="fill_parent"
                  android:layout_height="fill_parent"
                  />
    </LinearLayout>
</LinearLayout>
```

Code JAVA

```
public class Tutoriel5_Android extends Activity {
        private ListView maListViewPerso;

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

        maListViewPerso = (ListView) findViewById(R.id.listviewperso);
        ArrayList<HashMap<String, String>> listItem = new ArrayList<HashMap
<String, String>>();
```

```
        //Création d'une HashMap pour insérer les informations du premier item
de notre listView
        HashMap<String, String> map = new HashMap<String, String>();

        //on insère un élément titre que l'on récupérera dans le textView titre
créé dans le fichier affichageitem.xml
        map.put("titre", "Word");
        //on insère un élément description que l'on récupérera dans le textView
description créé dans le fichier affichageitem.xml
        map.put("description", "Editeur de texte");

        //on insère la référence à l'image (convertit en String car normalement
c'est un int) que l'on récupérera dans l'imageView créé dans le fichier
affichageitem.xml
        map.put("img", String.valueOf(R.drawable.word));

//enfin on ajoute cette hashMap dans arrayList
        listItem.add(map);

        //On refait la manip plusieurs fois avec des données différentes

        map = new HashMap<String, String>();
        map.put("titre", "Excel");
        map.put("description", "Tableur");
        map.put("img", String.valueOf(R.drawable.excel));
        listItem.add(map);

        map = new HashMap<String, String>();
        map.put("titre", "Power Point");
        map.put("description", "Logiciel de présentation");
        map.put("img", String.valueOf(R.drawable.powerpoint));
        listItem.add(map);

        map = new HashMap<String, String>();
        map.put("titre", "Outlook");
        map.put("description", "Client de courrier électronique");
        map.put("img", String.valueOf(R.drawable.outlook));
        listItem.add(map);

        //Création d'un SimpleAdapter qui se chargera de mettre les items
présent dans notre list (listItem) dans la vue affichageitem
        SimpleAdapter aa = new SimpleAdapter (this.getBaseContext(), listItem,
R.layout.affichageitem, new String[] {"img", "titre", "description"}, new int[]
{R.id.img, R.id.titre, R.id.description});

        //On attribue à notre listView l'adapter que l'on vient de créer
        maListViewPerso.setAdapter(aa);

        //Enfin on met un écouteur d'évènement sur notre listView
        maListViewPerso.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> a, View v, int position, long id) {

//on récupère la HashMap contenant les infos du item (titre, description, img)
HashMap<String, String> map = (HashMap<String, String>)
maListViewPerso.getItemAtPosition(position);
    //on crée une boite de dialogue
AlertDialog.Builder adb = new AlertDialog.Builder(this);

 adb.setTitle("Sélection Item");
//on insère un message à notre boite de dialogue, ici le titre de l'item cliqué
```

```
adb.setMessage("Votre choix : "+map.get("titre"));

adb.setPositiveButton("Ok", null);

adb.show();
}
});
}}
```

Création d'un adaptateur personnalisé

Code java

```
public class MainActivity extends ListActivity {
  public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
    String[] values = new String[] { "Android", "iPhone", "WindowsMobile" };

      // use your custom layout
   MonAdaptateurDeListe   adapter = new MonAdaptateurDeListe(this, values);
   setListAdapter(adapter);
    }}}

class MonAdaptateurDeListe extends ArrayAdapter<String> {
   //récupération des images
private Integer[] tab_images_pour_la_liste = {
     R.drawable.ic_launcher ,
     R.drawable.sym_keyboard_ok,
     R.drawable.ico_ok,
  };

   @Override
   public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater)getContext().getSystemService
(Context.LAYOUT_INFLATER_SERVICE);
       View rowView = inflater.inflate(R.layout.rowlayout, parent, false);

      TextView textView = (TextView) rowView.findViewById(R.id.label);
      ImageView imageView = (ImageView) rowView.findViewById(R.id.img);

      textView.setText(getItem(position));/à_
*;:

      if(convertView == null )
        imageView.setImageResource(tab_images_pour_la_liste[position]);
      else
        rowView = (View)convertView;

      return rowView;
   }

   public MonAdaptateurDeListe(Context context, String[] values) {
      super(context, R.layout.rowlayout, values);
```

```
    }
}
```

Personnalisation des items

Considérons une liste de personnes avec un nom, prénom et genre (Masculin / Féminin). Un `SimpleAdapter` ne peut pas gérer ces données. Il faut créer notre propre adaptateur, il faut partir de `BaseAdapter`.

```
Dè SimpleAdapter(Context context, List<? extends Map<String, ? >> data, int ressource, String[] from, int[] to=)// cvb,,nb
```

qu'une classe hérite de `BaseAdapter`, il faut implémenter obligatoirement trois méthodes :

- `public Object getItem(int position)` pour récupérer un item de la liste en fonction de sa position
- `public long getItemId(int position)` : pour récupérer l'identifiant d'un item de la liste en fonction de sa position (utilisé dans le cas d'une base de données)
- `public View getView(int position, View convertView, ViewGroup parent)` : doit renvoyer un objet Viewreprésentant la ligne située à la position fournie par l'adaptateur. cette méthode est appelée à chaque fois qu'un item est affiché à l'écran.

En ce qui concerne les paramètres de cette méthode :

- `position` : la position de l'item dans la liste (et donc dans l'adaptateur).
- `parent` : layout auquel rattacher la vue.
- `convertView` vaut `null`… ou pas

`convertView` vaut `null` uniquement les premières fois qu'on affiche la liste.
Dans notre exemple, `convertView` vaudra `null` aux sept premiers appels de `getView` (donc les sept premières créations de vues), c'est-à-dire pour tous les éléments affichés à l'écran au démarrage.
Toutefois, dès qu'on fait défiler la liste jusqu'à afficher un élément qui n'était pas à l'écran à l'instant d'avant, `convertView` ne vaut plus `null`, mais plutôt la valeur de la vue qui vient de disparaître de l'écran.

Il nous faut alors un moyen d'inflater une vue, mais sans l'associer à notre activité. Il existe au moins trois méthodes pour cela :

- `LayoutInflater getSystemService (LAYOUT_INFLATER_SERVICE)` sur une activité.
- `LayoutInflater getLayoutInflater ()` sur une activité.
- `LayoutInflater LayoutInflater.from(Context contexte)`, sachant que `Activity` dérive de `Context`.

Puis vous pouvez inflater une vue à partir de ce `LayoutInflater` à l'aide de la méthode `View inflate (int id, ViewGroup root)`, avec `root` la racine à laquelle attacher la hiérarchie désérialisée. Si vous indiquez `null`, c'est la racine actuelle de la hiérarchie qui sera renvoyée, sinon la hiérarchie s'attachera à la racine indiquée.

Pourquoi ce mécanisme? C'est une question d'optimisation. En effet, si vous avez un layout personnalisé pour votre liste, à chaque appel de `getView` vous allez remplir la rangée avec le layout à inflater depuis son fichier XML :

Exemple

```
LayoutInflater mInflater;
String[] mListe;

public View getView(int position, View convertView, ViewGroup parent) {
  TextView vue = (TextView) mInflater.inflate(R.layout.ligne, null);
  vue.setText(mListe[position]);
  return vue;
}
```

Cependant, la désérialisation est un processus lent!. C'est pourquoi il *faut* utiliser `convertView` pour vérifier si cette vue n'est pas déjà peuplée et ainsi ne pas désérialiser à chaque construction d'une vue :

```
LayoutInflater mInflater;
String[] mListe;

public View getView(int position, View convertView, ViewGroup parent) {
  TextView vue = null;

  // Si la vue est recyclée, elle contient déjà le bon layout
  if(convertView != null)
    vue  = (TextView) convertView; // On n'a plus qu'à la récupérer

 else     // Sinon, il faut utiliser le LayoutInflater
    vue = mInflater.inflate(R.layout.ligne, null);
    vue.setText(mListe[position]);
    return vue;
}
```

Exemple

Considérons une liste de personnes avec un nom, prénom et genre (Masculin / Féminin). Un `SimpleAdapter` ne peut pas gérer ces données.

1. Créer la classe personne :

```
public class Personne {
       public final static int MASCULIN = 1;
       public final static int FEMININ = 2;

       public String nom;
       public String prenom;
       public int genre;

       public Personne(String aNom, String aPrenom, int aGenre) {
              nom = aNom;
              prenom = aPrenom;
              genre = aGenre;
       }
       public static ArrayList<Personne> getAListOfPersonne(){
        ArrayList<Personne> listPers = new ArrayList<Personne>();
```

```
        listPers.add(new Personne("Nom1", "Prenom1", FEMININ));
        listPers.add(new Personne("Nom2", "Prenom2", MASCULIN));
        listPers.add(new Personne("Nom3", "Prenom3", MASCULIN));
        listPers.add(new Personne("Nom4", "Prenom4", FEMININ));
      return listPers;
      }
}
```

2. Ajoutez un fichier XML.

Les personnes possèdent trois propriétés. Il n'est donc plus possible de passer par le layout standard pour les afficher. Nous allons donc créer un nouveau layout, et l'utiliser pour afficher les items de la liste. Ajoutez un nouveau fichier XML.

Dans ce nouveau layout, ajouter deux TextView pour décrire les nom et prénom de chaque personne. Le genre de la personne sera décrit par la couleur de fond de notre item : bleu pour les garçons, rose pour les filles :).

```
<TextView  android:id="@+id/TV_Nom"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Nom">
      </TextView>

<TextView android:id="@+id/TV_Prenom"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Prénom">
      </TextView>
```

3. Classe pour gérer le mapping entre les données et le layout des items.

Pour afficher la liste des personnes en utilisant ce layout, créer un objet qui se chargera de gérer le mapping entre nos données et le layout des items.

Ce composant sera basé sur un Adapter. Dans Eclipse, créez une nouvelle classe : "PersonneAdapter". Faites-la hériter de "BaseAdapter".

L'Adapter va gérer une liste de personnes et va s'occuper également de l'affichage. On va donc lui ajouter trois propriétés.

```
public class PersonneAdapter extends BaseAdapter {
private List<Personne> mListP; // Une liste de personnes
private Context mContext; //Le contexte dans lequel est présent notre
adapter
private LayoutInflater mInflater; //Un mécanisme pour gérer l'affichage
graphique depuis un layout XML

//constructeur
public PersonneAdapter(Context context, List<Personne> aListP) {
  mContext = context;
  mListP = aListP;
  mInflater = LayoutInflater.from(mContext);
}
```

Le LayoutInflater permet de parser un layout XML et de te transcoder en IHM Android.

Pour respecter l'interface BaseAdapter, il nous faut spécifier la méthode "count()". Cette méthode permet de connaître le nombre d'items présent dans la liste. Dans notre cas, il faut donc renvoyer le nombre de personnes contenus dans "mListP".

```
public int getCount() {
  return mListP.size();
}
```

Ensuite, il y a deux méthodes pour identifier les items de la liste. Une pour connaitre l'item situé à une certaine position et l'autre pour connaitre l'identifiant d'un item en fonction de sa position.

```
public Object getItem(int position) {
  return mListP.get(position);
}

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

Maintenant il faut surcharger la méthode pour renvoyer une "View" en fonction d'une position donnée.
Cette view contiendra donc une occurrence du layout "personne_layout.xml" convenablement renseignée (avec le nom et prénom au bon endroit).

```
public View getView(int position, View convertView, ViewGroup parent) {
  LinearLayout layoutItem;

 //(1) : Réutilisation des layouts
  if (convertView == null) {
  //Initialisation de notre item à partir du  layout XML
"personne_layout.xml"
layoutItem = (LinearLayout) mInflater.inflate(R.layout.personne_layout,
parent, false);

  } else {
       layoutItem = (LinearLayout) convertView;
  }
  //(2) : Récupération des TextView de notre layout
  TextView tv_Nom = (TextView)layoutItem.findViewById(R.id.TV_Nom);
  TextView tv_Prenom =
(TextView)layoutItem.findViewById(R.id.TV_Prenom);

  //(3) : Renseignement des valeurs
  tv_Nom.setText(mListP.get(position).nom);
  tv_Prenom.setText(mListP.get(position).prenom);

  //(4) Changement de la couleur du fond de notre item
  if (mListP.get(position).genre == Personne.MASCULIN) {
    layoutItem.setBackgroundColor(Color.BLUE);
  } else {
       layoutItem.setBackgroundColor(Color.MAGENTA);
  }
```

```
  return layoutItem;
}
```

4. Pour afficher la liste, modifiez l'Activity principale :

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    //Récupération de la liste des personnes
    ArrayList<Personne> listP = Personne.getAListOfPersonne();

//Création et initialisation de l'Adapter pour les personnes
    PersonneAdapter adapter = new PersonneAdapter(this, listP);

    //Récupération du composant ListView
    ListView list = (ListView)findViewById(R.id.ListView01);
    //Initialisation de la liste avec les données
    list.setAdapter(adapter);
}
```