Application liste

Table des matières

Objectifs

A la fin de cette leçon, l'étudiant sera capable de :

- Décrire le Stockage une liste d'items;
- Afficher une liste, adaptateurs ;
- Consulter et éditer un item.

Application liste

I

1. Stockage une liste d'items

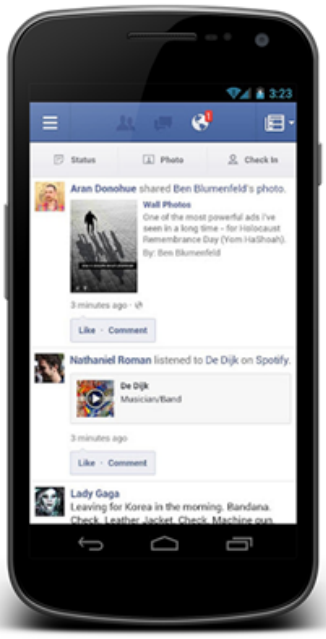
Définition

Les listes permettent l'affichage d'une grande quantité de données, plusieurs dizaines/centaines d'éléments sans ralentissement.

- La liste intègre une gestion de scroll afin de pouvoir faire défiler facilement.
- La liste limite la consommation mémoire et processeur.
- La liste permet d'afficher des vues complexes et variées.

Exemple : Applications utilisant des listes

Cas *Twitter* et *Facebook*



Les listes ont été créées afin de permettre aux développeurs de pouvoir charger dynamiquement des éléments sans ralentissement.

Elles reposent sur un composant dédié.

- *ListView*

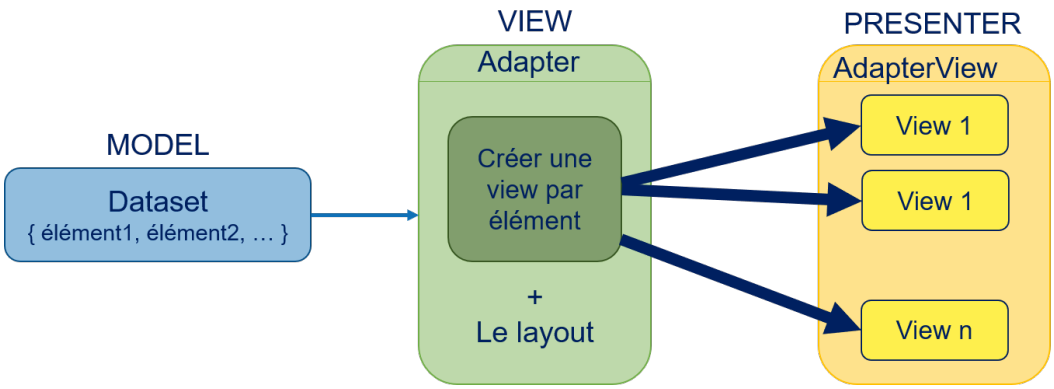
Un nouveau composant est venu améliorer *ListView* dans les versions récentes d'Android (API > 20) nommé *RecyclerView*.

Son implémentation étant plus complexe, il est préférable de commencer la découverte des listes avec *ListView*.

Une liste peut être vue comme trois objets distincts :

- *Le Dataset (modèle):* représente les données brutes à afficher
- *L'Adapter (vue):* représente une vue par élément de la liste
- *L'AdapterView (présentateur):* représente la somme des vues.

Il s'agit du modèle *MVP* (Model-View-Presenter)



Complément

Le Modèle – vue – présentateur (*MVP*) est une dérivation du modèle architectural, modèle – vue – contrôleur (*MVC*) principalement utilisé pour la création d'interfaces utilisateur.

Dans MVP, le présentateur assume la fonctionnalité de «l'*intermédiaire*». Dans MVP, toute la logique de présentation est poussée au présentateur. MVP préconise de séparer la logique métier et la logique de persistance de l'activité et du fragment

Éléments	Description
Modèle (Model)	Il représente les informations que l'utilisateur souhaite visualiser et ou manipuler dans les vues de l'application.C'est un acteur passif du modèle MVP. *Quelques exemples de Modèle :* - les informations saisies dans une commande, - les informations saisies dans un moteur de recherche, - les informations affichées comme résultat d'une recherche...
Vue (View)	Elle affiche à l'écran les informations nécessaires au cas d'utilisation dont elle est liée et répond également aux actions de l'utilisateur. Elle travaille en coopération avec le Présenteur. *Quelques exemples de Vue :* - un écran de connexion, - un écran qui affiche des produits, - un menu vertical dans le coin gauche de l'écran affichant des catégories de produit...
Présentateur (Presenter)	- Partie communicant avec les deux autres pour traduire et transmettre les commandes de l'utilisateur envoyée de la vue vers le modèle et pour formater et afficher les données du modèle dans la vue. - Le principe est de découpler la vue et le modèle, en utilisant le présentateur comme intermédiaire. - il permet d'avoir plusieurs vues d'un même modèle (exemple: une table de données sous la forme d'un tableau modifiable et sous la forme d'un graphique). Et une même vue peut présenter les données de plusieurs modèles (vue combinée ou synthèse).

2. Exercice

Exercice

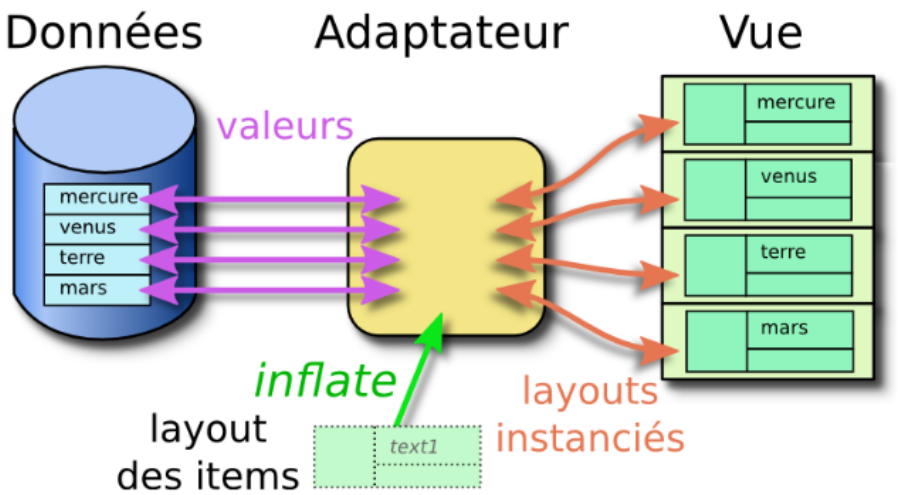
Donnez la définition de MVP

- Model-View-Presenter

- Model-View-Presenteur
- Model-Vue-Presenter
- Modèle-Vue-Presenter

3. liste, adaptateurs

Rôle d'un adaptateur / adapter



Adaptateur entre les données et la vue

Le rôle de l'adaptateur est de convertir une donnée ou un groupe de données en une vue.

Il fait le lien entre un *layout* et les *données* permettant de remplir le *layout*.

L'adaptateur répond à la question que pose le ListView: «*que dois-je afficher à tel endroit dans la liste ?*». Il va chercher les données et instancie le layout d'item avec les valeurs.

C'est une classe qui :

- accède aux données à l'aide de méthodes telles que getItem(int position), getCount(), isEmpty() quelque soit le type de stockage des éléments : tableau, BDD.
- crée les vues d'affichage des items : getView(...)à l'aide du layout des items. Cela consiste à instancier le layout — on dit expanser le layout, inflate en anglais

```
//--- Adapter
ArrayAdapter<String> adapter =
    new ArrayAdapter<String>( //--- Adapter type ArrayAdapter<String>
        context: MainActivity.this,  //--- Context
        android.R.layout.simple_list_item_1, //--- Layout
        names    //--- Dataset
    );
```

L'utilisation des *layout* et *adapter* de base permet de réaliser des listes rapides mais simplistes (layout avec peu d'éléments).

Lorsque l'on souhaite réaliser des vues avec un *layout* plus compliqué (plusieurs éléments, alignement des éléments personnalisés, ...) il faut créer une nouvelle classe adapter héritant de *BaseAdapter*.

En parallèle de cette nouvelle *classe*, il faut créer un nouveau *layout* qui servira de base pour la génération des vues.

Complément

Une classe héritant de *BaseAdapter* doit redéfinir les quatre méthodes suivantes :

1. public int *getCount() :*
   Cette méthode permet de connaître le nom de d'occurrence de données (le nombre d'éléments dans la liste).
2. public Object *getItem(*int position*) :*
   Cette méthode permet de retourner une occurrence précise des données en fonction du paramètre position.
3. public long *getItemId(*int position*)* :
   Cette méthode permet de retourner un Id unique pour chaque occurrence des données
4. public View *getView(*int position, View convertView, ViewGroup parent*)*
   Cette méthode permet de retourner pour chaque occurrence de données une view, remplie comme voulue, en utilisant le layout de l'adapter.

En plus de ces quatre *méthodes*, il faut absolument définir un *constructeur* permettant de stocker le *context* ainsi que les *données* (Dataset).

```
@Override
public int getCount() { return m_listP.size(); }
@Override
public Object getItem(int position) { return m_listP.get(position); }
@Override
public long getItemId(int position) { return position; }
@Override
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) m_inflater.inflate(R.layout.person_view, parent,  attachToRoot: false);
    } else {
        layoutItem = (LinearLayout) convertView;
    }

    //(2) : Récupération des TextView de notre layout
    TextView tv_lastName = (TextView)layoutItem.findViewById(R.id.textView_lastName);
    TextView tv_firstName = (TextView)layoutItem.findViewById(R.id.textView_firstName);
    TextView tv_age = (TextView)layoutItem.findViewById(R.id.textView_age);
    ImageView   tv_img = (ImageView)layoutItem.findViewById(R.id.imageView);

    //(3) : Renseignement des valeurs
    tv_lastName.setText("Nom :" +" "+ m_listP.get(position).getLastName());
    tv_firstName.setText("Prénom :" +" "+ m_listP.get(position).getFirstName());
    tv_age.setText("Age :" +" "+ String.valueOf(m_listP.get(position).getAge()));
    tv_img.setImageResource(m_imgId[m_listP.get(position).getImg()]);
    //---
    if(position%2 == 0)
        layoutItem.setBackgroundColor(Color.rgb( red: 250, green: 250, blue: 250));
    //On retourne l'item créé.
    return layoutItem;
}
```

Rappel : Un layout

layout est une représentation visuelle de l'application Android. Il peut contenir du texte, des images, des formulaires et peut intéragir avec l'utilisateur. Les layouts peuvent être construits à partir d'un fichier XML ou directement dans le code Java, au moment d'exécution de l'application.

Adaptateurs prédéfinis

Android propose quelques classes d'adaptateurs prédéfinis, dont :

- *ArrayAdapter* pour un ArrayList simple,
- *SimpleCursorAdapter* pour accéder à une base de données SQLite (on ne verra pas).

En général, dans une application innovante, il faut définir son propre adaptateur.

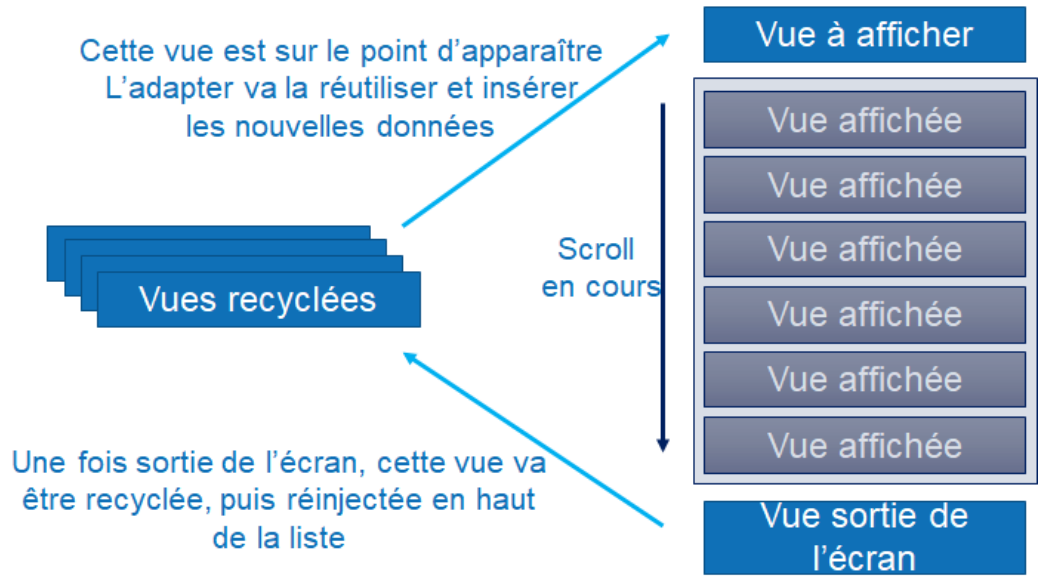
ArrayAdapter<Type>pour les listes

Il permet d'afficher les données d'un *ArrayList*, mais il est limité à une seule chaîne par item, par exemple le nom d'une planète, fournie par sa méthodetoString(). Son constructeur : ArrayAdapter(Context *context*, int *item_layout_id*, int *textview_id*, List<T> *données*)

- *context* : c'est l'activité qui crée cet adaptateur, mettre *this*
- *item_layout_id* : identifiant du layout des items, par exemple *android.R.layout.simple_list_item_1* ou *R.layout.item*
- *textview_id* : identifiant du *TextView* dans ce layout, par exemple *android.R.id.text1*ou *R.id.item_nom*
- *données* : c'est la liste contenant les données (List est une surclasse de *ArrayList*)

AdapterView

Les AdapterView contiennent toutes les view, mais charge seulement celles visibles afin d'économiser des ressources.



- La gestion des interactions utilisateur se fait entièrement sur l'AdapterView.

- Pour ajouter une gestion du clic il faut utiliser la méthode :

void setOnItemClickListener(AdapterView.OnItemClickListener listener)

- La fonction de callback associée est :

1. *void* onItemClick(AdapterView<?> adapter, View view, int position, long id)
2. *adapter* -> l'AdapterView qui contient la vue sur laquelle le clic a été effectué.
3. *view* -> la vue que laquelle le clic a été effectué.
4. *position* -> la position de la vue dans la liste.
5. *id* -> l'identifiant de la vue.

Exemple : Exemple d'utilisation d'un clic sur un élément d'une liste

Lors d'un clic sur un élément de la liste, faire apparaître le numéro de l'élément ainsi que l'attribut « *lastName* » de l'élément dans un Toast.

```
m_listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        //---
        Toast.makeText(getApplicationContext(),
                text: "Position : "+position+", lastName : "+m_listP.get(position).getLastName(),
                Toast.LENGTH_SHORT).show();

    }
});
```

4. Exercice

Exercice

Répondre par Vrai ou Faux :

getCount() est une méthode qui permet de retourner un Id unique.

- ○ Vrai
- ○ Faux

5. Affichage de la liste

Au sein d'un gabarit, on peut implanter une liste que l'on pourra dérouler si le nombre d'éléments est important. Si l'on souhaite faire une liste plein écran, il suffit juste de poser un layout linéaire et d'y implanter une ListView. Le XML du gabarit est donc:

```
<LinearLayout ...>
<ListView android:id="@+id/listView1" ...>
</ListView></LinearLayout>
```

Etant donné qu'une liste peut contenir des éléments graphiques divers et variés, les éléments de la liste doivent être insérés dans un *ListAdapter* et il faut aussi définir le gabarit qui sera utilisé pour afficher chaque élément du *ListAdapter*. Prenons un exemple simple: une liste de chaine de caractères. Dans ce cas, on créé un nouveau gabarit montexte et on ajoute dynamiquement un *ArrayAdapter* à la liste *listView1*. Le gabarit suivant doit être placé dans *montexte.xml*:

```
<TextView ...> </TextView>
```

6. Consulter et éditer un item

Les *Intents* permettent de gérer l'envoi et la réception de messages afin de faire coopérer les applications. Le but des Intents est de déléguer une action à un autre composant, une autre application ou une autre activité de l'application courante.

Un objet Intent contient les information suivantes:

- le nom du composant ciblé (facultatif)
- l'action à réaliser, sous forme de chaine de caractères
- les données: contenu MIME et URI
- des données supplémentaires sous forme de paires clef/valeur
- une catégorie pour cibler un type d'application
- des drapeaux (informations supplémentaires)

On peut envoyer des Intents informatifs pour faire passer des messages. Mais on peut aussi envoyer des Intents servant à lancer une nouvelle activité.

Intents pour une nouvelle activité

Il y a plusieurs façons de créer l'objet de type Intent qui permettra de lancer une nouvelle activité. Si l'on passe la main à une activité interne à l'application, on peut créer l'Intent et passer la classe de l'activité ciblée par l'Intent:

```
Intent login = new Intent(getApplicationContext(), GiveLogin.class);
startActivity(login);
```

Le premier paramètre de construction de l'Intent est en fait le contexte de l'application. Dans certain cas, il ne faut pas mettre *this* mais faire appel à *getApplicationContext()* si l'objet manipulant l'Intent n'hériste pas de *Context*.

S'il s'agit de passer la main à une autre application, on donne au constructeur de l'Intent les données et l'URI cible: l'OS est chargé de trouver une application pouvant répondre à l'Intent.

```
Button b = (Button)findViewById(R.id.Button01);
b.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
  Uri telnumber = Uri.parse("tel:0248484000");
  Intent call = new Intent(Intent.ACTION_DIAL, telnumber);
  startActivity(call);
}
});
```

Sans oublier de déclarer la nouvelle activité dans le *Manifest*.

Retour d'une activité

Lorsque le bouton retour est pressé, l'activité courante prend fin et revient à l'activité précédente. Cela permet par exemple de terminer son appel téléphonique et de revenir à l'activité ayant initié l'appel.

Au sein d'une application, une activité peut vouloir récupérer un code de retour de l'activité "enfant". On utilise pour cela la méthode *startActivityForResult* qui envoie un code de retour à l'activité enfant. Lorsque l'activité parent reprend la main, il devient possible de filtrer le code de retour dans la méthode *onActivityResult* pour savoir si l'on revient ou pas de l'activité enfant.

L'appel d'un *Intent* devient donc:

```
public void onCreate(Bundle savedInstanceState) {
  Intent login = new Intent(getApplicationContext(), GivePhoneNumber.class);
  startActivityForResult(login,48);
... }
```

Le filtrage dans la classe parente permet de savoir qui avait appelé cette activité enfant:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
  if (requestCode == 48)
    Toast.makeText(this, "Code de requête récupéré (je sais d'ou je viens)",
                   Toast.LENGTH_LONG).show();
}
```

Résultat d'une activité

Il est aussi possible de définir un résultat d'activité, avant d'appeler explicitement la fin d'une activité avec la méthode *finish()*. Dans ce cas, la méthode *setResult* permet d'enregistrer un code de retour qu'il sera aussi possible de filtrer dans l'activité parente.

Dans l'activité enfant, on met donc:

```
Button finish = (Button)findViewById(R.id.finish);
  finish.setOnClickListener(new OnClickListener() {

  @Override
  public void onClick(View v) {
    setResult(50);
    finish();
  }});
```

Ajouter des informations

Les Intent permettent de transporter des informations à destination de l'activité cible. On appelle ces informations des Extra : les méthodes permettant de les manipuler sont *getExtra* et *putExtra*. Lorsqu'on prépare un Intent et que l'on souhaite ajouter une information de type "clef -> valeur" on procède ainsi:

```
Intent callactivity2 = new Intent(getApplicationContext(), Activity2.class);
callactivity2.putExtra("login", "jfl");
startActivity(callactivity2);
```

Du côté de l'activité recevant l'Intent, on récupère l'information de la manière suivante:

```
Bundle extras = getIntent().getExtras();
String s = new String(extras.getString("login"));
```