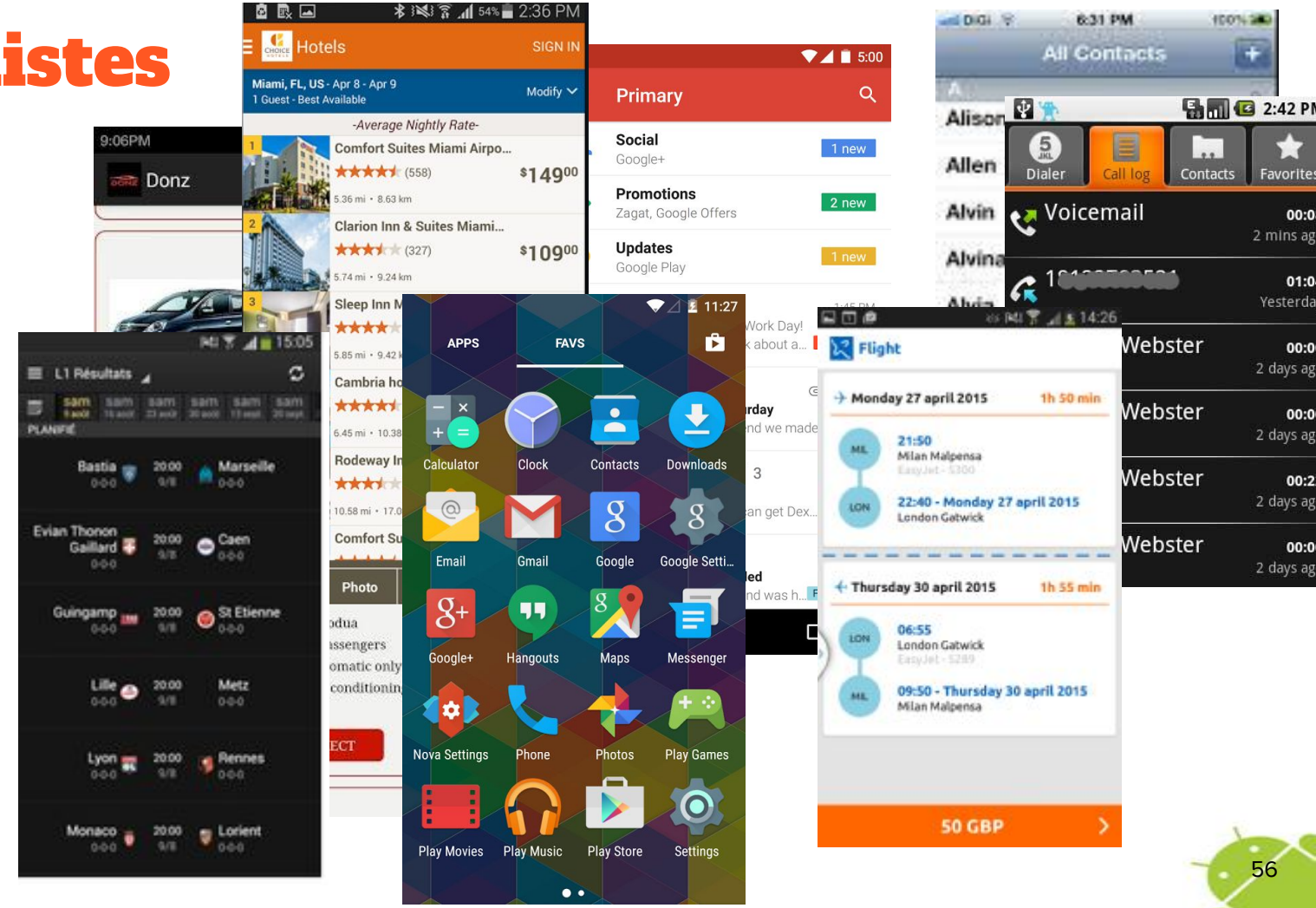
3.4 Les listes

Un composant indispensable dans la majorité des applications mobiles. L'espace est réduit, la solution est donc de défiler le contenu.



Listes : le principe

Au sein d'un gabarit (layout), 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/maliste" ...>
    </ListView>
</LinearLayout>
```

Étant donné qu'une liste peut contenir des éléments graphiques divers et variés, les éléments de la liste doivent être insérés dans un Adapter et il faut aussi définir le gabarit qui sera utilisé pour afficher chaque élément de l' Adapter.

Liste simple

Prenons un exemple simple: une liste de chaîne de caractères. Dans ce cas, on crée un nouveau gabarit **ligne** et on attache un **ArrayAdapter** à la liste **maliste**. Le gabarit suivant doit être placé dans *ligne.xml*:

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

Le code de l'activité qui crée la liste peut être :

```
ListView list = (ListView)findViewById(R.id.maliste);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(list.getContext(),
                                        R.layout.ligne);
for (int i=0; i<40; i++) {
  adapter.add("Etudiant " + i); }
list.setAdapter(adapter);
```

Démonstration

Video : www.t-t.ma/docs/AndroidList1.avi

Liste avec Layout avancé

Lorsque les listes contiennent un layout plus complexe qu'un simple texte, il faut utiliser un autre constructeur de **ArrayAdapter** (ci-dessous) où **resource** est l'id du layout à appliquer à chaque ligne et **textViewResourceId** est l'id de la zone de texte inclue dans ce layout complexe. A chaque entrée de la liste, la vue générée utilisera le layout complexe et la zone de texte contiendra la *string* passée en argument à la méthode **add**.

```
ArrayAdapter (Context context, int resource, int textViewResourceId)
```

Liste avec Layout avancé

Le code de l'exemple précédent doit être adapté comme ceci:

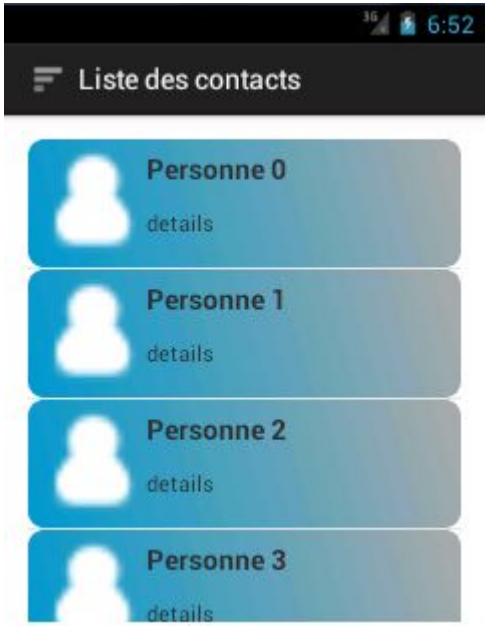
```
ListView list = (ListView)findViewById(R.id.maliste);
ArrayAdapter<String> tableau = new
ArrayAdapter<String>(list.getContext(), R.layout.ligne,
R.id.monTexte);
for (int i=0; i<40; i++) {
  tableau.add("Personne " + i); }
list.setAdapter(tableau);
```

Avec le layout de ligne suivant (ligne.xml):

```
<LinearLayout …>
  <ImageView
android:id="@+id/monIcone"/>
  <LinearLayout…>
    <TextView …
android:id="@+id/monTexte"/>
    <TextView …
android:id="@+id/autreTexte"/>

  </LinearLayout>
</LinearLayout>
```

Interface résultat

www.t-t.ma/docs/AndroidList2.avi



Listes avancées

Une autre solution consiste à hériter de la classe **ArrayAdapter** ou **BaseAdapter**, ce qui permet de re-coder la méthode **getView()** et permet d'afficher des objets complexes autres que les simples Strings.

getView() permet de faire le Mappage entre les données d'un objet complexe et le Layout de la ligne.

Listes avancées : données complexes

Supposons qu'on a la classe suivante :

```
class Personne{
  private String nom, prenom, description;
  private boolean online=false;
  public Personne(String fn, String ln, boolean
o){...}
  public String getFstName(){return nom;}
  public String getLstName(){return prenom;}
  public boolean isOnline(){return online;}
  public void setFstName(String fstName){....}
……
……
…
}
```

Dans notre programme principal (e.g. Activity qui contient la ListView)

```
ArrayList<Personne> data=.....;

ListView list =
(ListView)findViewById(R.id.maliste);
MyAdapter adapter= new
MyAdapter(list.getContext(), data);

list.setAdapter(adapter);
```

Listes Avancées

Et finalement, notre Adapter personnalisé :

```
class MyAdapter extends ArrayAdapter {
.....
.....
.....
@Override
public View getView(int position, View convertView, ViewGroup parent) {
      LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
      View rowView = inflater.inflate(R.layout.ligne, parent, false);
      TextView textView1 = (TextView) rowView.findViewById(R.id.monTexte);
      TextView textView2 = (TextView) rowView.findViewById(R.id.autreTexte);
      ImageView imageView = (ImageView) rowView.findViewById(R.id.monIcone);
      textView1.setText(values.get(position).getFstName()+" "+values.get(position).getLstName());
      textView2.setText(values.get(position).getDescription());

      if (values.get(position).isOnline()) {
         imageView.setImageResource(R.drawable.online);
      } else {
         imageView.setImageResource(R.drawable.offline);
      }

      return rowView;
 }
}
```

Les galeries

L'affichage des éléments de l'interface en "galerie" est une pratique aussi répondu que les Listes dans le domaine du mobile. Android prévoit une sorte de liste spéciale appelée **GridView** conçu pour cet effet.

GridView et ListView sont toutes les deux dérivées de AbsListView ce qui simplifie le codage pour les deux cas et permet une transition facile entre les deux modes d'affichage.

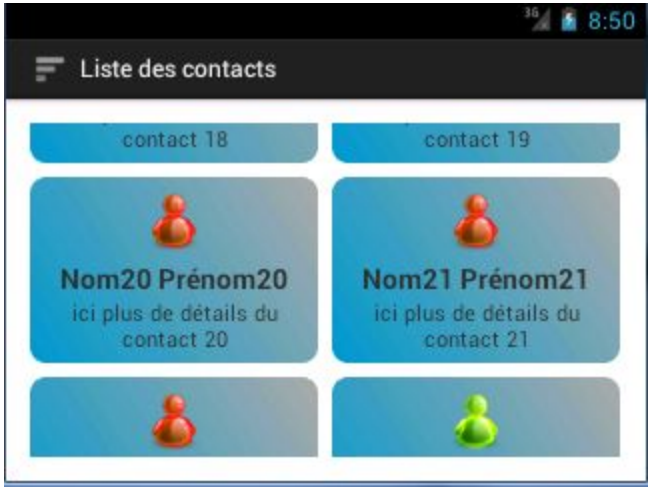
Le layout de l'activité devient

```
<LinearLayout ...>
<GridView android:id="@+id/maliste" ...>
</GridView></LinearLayout>
```

Le layout d'un élément de la grille (ligne dans la liste) doit être également adapté au nouvel affichage :

```
<LinearLayout ...>

    <ImageView  android:id="@+id/monIcone" .../>

    <TextView android:id="@+id/monTexte"..../>

    <TextView android:id="@+id/autreTexte".../>

</LinearLayout>
```

Les galeries



3.6 Les onglets

La réalisation d'onglets permet de mieux utiliser l'espace réduit de l'écran. Pour réaliser les onglets, les choses se sont considérablement simplifiées depuis l'API 11. Dans l'activité principale, il faut ajouter à l'*action bar* existante des onglet des objets de type **Tab** ayant obligatoirement un listener de type **TabListener**, comme présenté dans la documentation sur les onglets:

```
final ActionBar actionBar = getActionBar();
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

 ActionBar.TabListener tabListener = new ActionBar.TabListener() {
    public void onTabSelected(ActionBar.Tab tab,
FragmentTransaction ft) {
        // show the given tab
    }
    public void onTabUnselected(ActionBar.Tab tab,
FragmentTransaction ft) {
        // hide the given tab
    }
    public void onTabReselected(ActionBar.Tab tab,
FragmentTransaction ft) {
        // Selected but not first time
 }};
 // Add 3 tabs, specifying the tab's text and TabListener
 for (int i = 0; i < 3; i++) {
    Tab t = actionBar.newTab().setText("Tab " + (i + 1));
    t.setTabListener(tabListener);
    actionBar.addTab(t);
 }
```

Il faut ensuite coder le changement d'écran lorsque l'événement *onTabSelected* survient. On utilise pour cela l'objet **FragmentTransaction** pour lequel on passe l'*id* du composant graphique à remplacer par un objet de type **Tab**:

```
public void onTabSelected(ActionBar.Tab tab,
FragmentTransaction ft) {
FragmentTab newft = new FragmentTab();
if (tab.getText().equals("Tab 2")) {
  newft.setChecked(true);
}
ft.replace(R.id.fragmentContainer, newft);
}
```

La classe **FragmentTab** hérite de **Fragment**: c'est une sous partie d'activité qui a son propre cycle de vie. Dans cet exemple, au lieu de faire 2 fragments différents, on surcharge la méthode *onCreateView* pour cocher la case à cocher:

```
public class FragmentTab extends Fragment {
...
  public View onCreateView(LayoutInflater inflater,
ViewGroup container,
        Bundle savedInstanceState)   {
    View view =  inflater.inflate(R.layout.fragment1,
container, false);
    CheckBox cb = (CheckBox)
view.findViewById(R.id.checkBox1);
    cb.setChecked(this.checked);
    return view;
  }}
```

Le gabarit de l'activité, est donc:

```
<RelativeLayout>
 <TextView
   android:id="@+id/textView1"
   android:text="@string/hello_world" />
  <FrameLayout
   android:id="@+id/fragmentContainer" />
  </FrameLayout>
</RelativeLayout>
```

Et celui du fragment fragment1.xml:

```
<LinearLayout
  <CheckBox android:id="@+id/checkBox1" />
</LinearLayout>
```

Il faut bien remarquer que:

- R.id.fragment1 est l'*id* d'un composant de type **ViewGroup** de l'activité.
- la méthode *inflate* crée la vue en se basant sur le layout *R.layout.fragment1*.

Exercices

- Lors de la consommation du webservice Profile, récupérez la liste des notes de l'étudiant et stockez-les dans un ArrayList .
- Ajoutez une ListeView à votre activité pour afficher les notes seulement quand l'appareil est en mode paysage.
- Définir le layout d'un item de la liste qui va afficher la matière, le score et une icone indiquant si la note est bonne ou pas.
- Définir l'Adapter adéquat qui permettra de lier la source de données à la ListeView