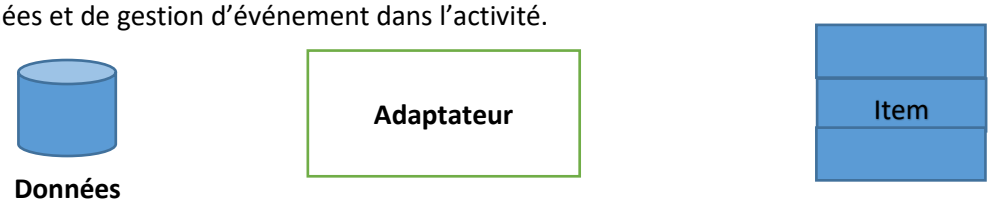
Android – Les listes

Dans les applications informatiques, nous avons souvent besoin d'afficher des informations sous forme de listes de choix. Le framework Android propose deux techniques pour intégrer une liste dans un écran :

- Utiliser la classe **`ListActivity`** ;
- Intégrer le composant de **`ListView`** dans un layout et implémenter le mécanisme de présentation des données et de gestion d'événement dans l'activité.



1. Le composant ListView

Le composant `ListView` est un conteneur de vue, au même titre que les différents Layout étudiés. Il se chargera de lier les éléments de la liste et de les afficher en une liste (AdapterView). Dans la suite des exemples, le composant `ListView` est ajouté dans le fichier activity_main.xml associé à la classe MainActivity.
Pour afficher chaque élément de la liste, il faut implémenter un adaptateur qui fera le lien entre les données et le fichier de layout qui contient le composant.
L'**`ArrayAdapter`** est le plus simple à utiliser. Il appelle la méthode `toString()` de chaque objet de la liste et enveloppe chaque chaîne obtenue dans une vue associée à un item de la liste présentée.

1.1.Interface utilisateur :

Syntaxe :

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

Exemple : fichier activity_main.xml

```
....
<TextView
    android:id="@+id/idTvSelection"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pizza_selectionnee"/>

<ListView
    android:id="@+id/idLvPizzas"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
...
```

1.2.Implémenter l'adaptateur et l'associer au composant

L'adaptateur est implémenté dans le fichier MainActivity.java. Il va chercher les données et les instancie dans le layout d'item avec les différentes valeurs. On utilise ici l'ArrayAdapter. Le constructeur utilisé dans notre exemple est :

```
public ArrayAdapter (Context context, int resource, List<T> objects)
```

`Context context` : représente le contexte d'exécution de l'application (informations sur l'activité).
`int resource` : identifiant de la ressource layout utilisée pour l'affichage de chaque élément. Android définit des layouts pour des éléments de listes simples : R.layout.simple_list_item1, R.layout.simple_list_item2…

`List<T> objects` : éléments de type T qui sont affichés par la liste.

Une fois l'adaptateur créé, il faut l'affecter au composant ListView. La classe ListView fournit la méthode setAdapter pour cela :

```
public void setAdapter(ListAdapter adapter) ;
```

Exemple : MainActivity.java correspondant au fichier activity_main.xml ci-dessus

```
public class MainActivity extends AppCompatActivity {

    TextView tvSelection ;
    ListView lvPizzas ;

    private String[] pizzas = {"Royale","Océane","Paysanne","Végétarienne"} ;

    @Override
    protected void onCreate( Bundle savedInstanceState ) {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_main );

        tvSelection = ( TextView ) findViewById( R.id.idTvSelection ) ;
        lvPizzas = ( ListView ) findViewById( R.id.idLvPizzas ) ;

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

        lvPizzas.setAdapter( adaptateur ) ;
    }
}
```

ArrayAdapter accède aux données à l'aide de méthodes telles que : getItem(int position), getCount()…
ArrayAdapter crée les vues d'affichage des items à l'aide de getView(). Cele revient à instancier le layout (expanser le layout ou inflate en anglais).

1.3.Gestion du clic sur un élément de la liste

Lorsque l'utilisateur sélectionne un élément de la liste, il déclenche un événement sur l'élément de la liste. Pour traiter cet événement, l'objet `ListView` définit une méthode `setOnItemClickListener()`. La fonction de callback associée est : `onItemClick`().

```
public void setOnClickListener(AdapterView.OnItemClickListener listener) ;
```

L'activité doit donc implémenter l'interface **`AdapterView.OnItemClickListener.`** Cette interface expose une méthode **`onItemClick`**() qui est appelée pour gérer le traitement du clic.
L'écouteur n'a pas besoin d'être modifié si l'activité est réactivée, l'affectation peut alors être traitée dans la méthode `onCreate()` de l'activité.

```
list.setOnItemClickListener(new AdaptaterView.OnItemClickListener () {
   @Override
   public void onItemClick(AdapterView<?> parent , View vue , int position , long id){
      // Traitement du clic
   }
}
```

Les paramètres de la méthode `onItemClick()` sont :
`AdapterView<> parent` : le parent de l'élément cliqué, c'est-à-dire la liste elle-même ;
`View v` : la vue qui contient l'élément de la liste qui a déclenché l'événement
`int position` : la position de l'élément dans la source de données de l'adaptateur

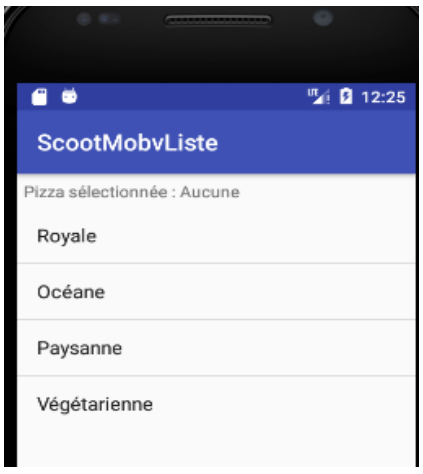
`long id` : l'identifiant de l'élément, c'est-à-dire la valeur retournée par la méthode `getItemId()` de la classe ArrayAdapter.

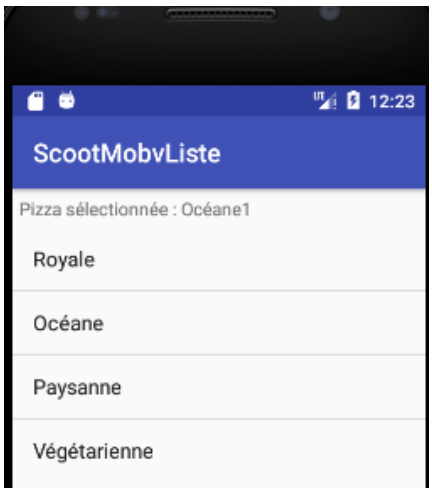
Exemple : MainActivity.java

```
public class MainActivity extends AppCompatActivity implements AdapterView.OnItemClickListener
{
    TextView tvSelection ;
    ListView lvPizzas ;
    private String[] pizzas = {"Royale","Océane","Paysanne", "Végétarienne"} ;

    @Override
    protected void onCreate( Bundle savedInstanceState ) {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_main );

        tvSelection = ( TextView ) findViewById( R.id.idTvSelection ) ;
        tvSelection.setText("Pizza sélectionnée : Aucune");
        lvPizzas = ( ListView ) findViewById( R.id.idLvPizzas ) ;

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

        lvPizzas.setAdapter( adaptateur ) ;
        lvPizzas.setOnItemClickListener( this ) ;
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View vue, int position, long id){
        String pizzaSelectionnee = pizzas[ position ] ;
        tvSelection.setText( "Pizza sélectionnée : " + pizzaSelectionnee + " " + id);
    }
}
```

Il est également possible de traiter l'événement en instanciant une classe anonyme.

```
lvPizzas.setOnItemClickListener(
    new OnItemClickListener(){

      @Override
      public void onItemClick(AdapterView<?> parent,View vue,int position,long id)
      {
            String pizzaSelectionnee = pizzas[ position ] ;
            tvSelection.setText( pizzaSelectionnee ) ;
       }
   }

 ) ;
```





2. La classe ListActivity

Si l'activité est "pilotée" par une seule liste, elle peut alors être une classe dérivée de la classe ListActivity.

Fichier MainActivity.java

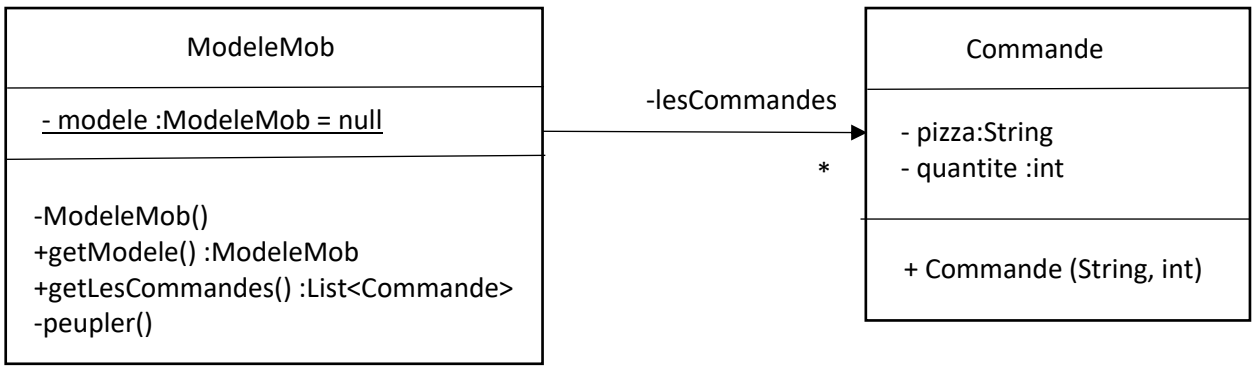
```
public class MainActivity extends ListActivity {

     TextView tvSelection ;

     private String[] pizzas = {"Royale","Océane","Campagnarde","Paysanne"} ;

     @Override
     protected void onCreate( Bundle savedInstanceState ) {
           super.onCreate( savedInstanceState );
           setContentView( R.layout.activity_main );
           tvSelection = ( TextView ) findViewById( R.id.tvSelection ) ;
           ArrayAdapter<String> adaptateur = new  ArrayAdapter<String>(
                       this, android.R.layout.simple_list_item_1 , pizzas
           ) ;
           this.setAdapter( adaptateur ) ;
     }

     @Override
     public void onListItemClick(ListView parent, View vue, int position,long id){
           String pizzaSelectionnee = pizzas[ position ] ;
           tvSelection.setText( pizzaSelectionnee ) ;
     }
}
```

3. Liste personnalisée

ArrayAdapter n'affiche qu'un seul texte par item. Si on a besoin d'afficher plusieurs informations dans un item, il faut définir son propre adaptateur et son propre fichier de type layout dans le répertoire res/layout.

Exemple :
Nous souhaitons afficher dans une liste des commandes de pizzas. Chaque élément de la liste contient le nom de la pizza et le nombre commandé.

Dans notre exemple, nous allons travailler avec des données récupérées depuis un modèle. Les classes métiers utilisées seront :



3.1. Fichier de layout personnalisé :

Le premier travail à faire est de créer le fichier de layout correspondant à un item de la liste. Ce fichier sera créé dans le répertoire res/layout.

Fichier res/layout/item_commande.xml

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_row="2">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="10dp"
        android:id="@+id/idTvItemPizza"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="right"
        android:paddingRight="10dp"
        android:id="@+id/idTvItemQuantite"/>
</GridLayout>
```

Fichier res/layout/main_activity.xml

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

Fichier MainActivity.java

```
public class MainActivity extends AppCompatActivity implements OnItemClickListener {

    ListView lvCommandes ;
    TextView tvSelectionItem;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        lvCommandes = (ListView) findViewById(R.id.idLvCommande);
        tvSelectionItem = (TextView) findViewById(R.id.idTvSelectionItem);
        tvSelectionItem.setText("Pizza sélectionnée : aucune");

        ItemCommandeAdapter adapter = new ItemCommandeAdapter(this,
ModeleMob.getModele().getLesCommandes());
        lvCommandes.setAdapter(adapter);
        lvCommandes.setOnItemClickListener(this);
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Commande commandeSelectionnee =
ModeleMob.getModele().getLesCommandes().get(position);

        tvSelectionItem.setText("Pizza sélectionnée : "+
commandeSelectionnee.getPizza());
    }

}
```

Un adaptateur personnalisé est une classe qui peut implémenter l'interface Adapter ou hériter de la classe BaseAdapter. Il doit également définir la méthode getView() (il faut respecter la signature de la méthode getView() de la classe surtype). Cette méthode est appelée à chaque fois qu'un élément est affiché à l'écran.

```
public View getView(int position, View convertView, ViewGroup parent)
```

`position` représente la position de l'élément dans la liste
`parent` est le layout auquel rattacher la vue
`convertView` vaut null tant que la liste s'affiche sur un écran. Dès qu'on fait défiler la liste pour afficher un élément, `convertView` contiendra la vue qui vient de disparaître de l'écran. On dit que la vue est recyclée puisqu'on n'a pas besoin de la voir. Pour cela, il faut utiliser le `LayoutInflater` pour construire la vue (on dit « inflater ») à l'aide de la méthode `inflate`().

```
public class ItemCommandeAdapter extends ArrayAdapter<Commande>  {
    public ItemCommandeAdapter (Context context, List<Commande> lesCommandes) {
        super(context, -1, lesCommandes );
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater layoutInflater = (LayoutInflater) getContext().
                            getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        View vItem = null;
        // Si la vue est recyclée, elle contient déjà le bon layout
```

```
        if (convertView != null) {
            vItem = convertView ;
        } else { // il faut utiliser le LayoutInflater
            vItem = layoutInflater.inflate(R.layout.item_commande, parent, false);
        }
        TextView tvPizza = (TextView) vItem.findViewById(R.id.idTvItemPizza);
        tvPizza.setText(ModeleMob.getModele().getLesCommandes().
                                            get(position).getPizza());
        TextView tvQuantite = (TextView) vItem.findViewById(R.id.idTvItemQuantite);
        tvQuantite.setText(""+ ModeleMob.getModele().getLesCommandes().
                                                get(position).getQuantite());

        return vItem;
    }
}
```

MainActivity.java correspondant :

```
public class MainActivity extends AppCompatActivity
                          implements AdapterView.OnItemClickListener {

    ListView lvCommandes ;
    TextView tvSelectionItem;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        lvCommandes = (ListView) findViewById(R.id.idLvCommande);
        tvSelectionItem = (TextView) findViewById(R.id.idTvSelectionItem);
        tvSelectionItem.setText("Pizza sélectionnée : aucune");

        ItemCommandeAdapter adapter = new ItemCommandeAdapter(this,
                                            ModeleMob.getModele().getLesCommandes());
        lvCommandes.setAdapter(adapter);
        lvCommandes.setOnItemClickListener(this);
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Commande commandeSelectionnee = ModeleMob.getModele().getLesCommandes()
                                                        .get(position);
        tvSelectionItem.setText("Pizza : "+ commandeSelectionnee.getPizza());
    }
}
```