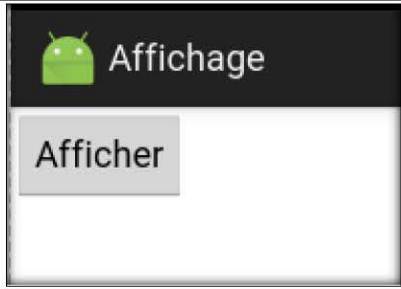


Chapitre 5 : Les éléments graphiques

Introduction









Une interface n'est pas une image statique mais un ensemble de composants graphiques (vues), qui peuvent être des boutons, du texte, mais aussi des groupements d'autres composants graphiques, pour lesquels nous pouvons définir des identifiants et les récupérer à travers la méthode `findViewById` de la classe `Activity`.

Exemple :

<u>activity_main.xml</u>	<u>MainActivity.java</u>
<pre><Button android:id="@+id/btnAfficher" android:text="Afficher" /></pre>	<pre>public class MainActivity extends Activity { private Button bouton1; @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); bouton1=(Button) findViewById(R.id.btnAfficher); } }</pre>
	

I. Les éléments graphiques simples

Le tableau suivant illustre les principaux éléments graphiques simples :

Image de l'élément graphique	Intitulé	Usage
	TextView	Affiche de texte a priori non éditable
	EditText	Affiche de texte éditable
	Button	Affiche un bouton de commande
	CheckBox	Affiche une case à cocher
	RadioButton	Affiche un bouton radio
	DatePicker	Affiche un calendrier
	SeekBar	Affiche une barre horizontale dotée d'un curseur permettant de modifier la valeur.
	ImageView	Affiche une zone dans laquelle s'affiche une image.

I.1. TextView

Le `TextView` est le widget le plus basique qu'il soit : une simple zone de texte équivalent à la balise `label` en HTML. Son contenu ne peut pas être changé.

Voici quelques propriétés qui s'appliquent au `TextView` :

- `android:textStyle` : accepte les valeurs « normal », « bold » et « italic »
- `android:typeface` : peut prendre l'une des valeurs : « normal », « sans », « serif » et « monospace »

Les valeurs peuvent se cumuler grâce au symbole « | ». Par exemple, si l'on affecte à l'attribut `textStyle` la valeur « italic|bold », le texte apparaîtra à la fois en italique et en gras.

```
En XML : android:text="Nouveau text"
En Java : setText("Nouveau text")
```

De même, le développeur peut modifier la couleur du texte :

```
En XML : android:textColor=""
En Java : setTextColor(int)
```

Egalement, on peut modifier la taille du texte :

```
En XML : android:textSize="...sp"
En Java : set textSize(float)
```

I.2.

La classe `EditText` hérite de la classe `TextView`, en plus elle est éditable. Elle est équivalente à la zone de saisie de texte en HTML.


La syntaxe XML de base d'un élément `EditText` est le suivant :

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:inputType="text" />
```

Des masques de saisie peuvent être rattachés à l'`EditText` en spécifiant une valeur à la propriété XML :

```
android:inputType
```

L'exemple suivant utilise 7 valeurs différentes pour la propriété `inputType` :

Interface	Tableau : Description des valeurs de la propriété <code>inputType</code>	
	Champs	Valeurs de la propriété <code>inputType</code>
	Nom et prénom	"textCapCharacters textPersonName"
	Age	"number"
	Adresse E-mail	"textEmailAddress"
	Site web	"textUri"
	Téléphone	"phone"
	Date de naissance	"date"
	Mot de passe	"textPassword"

Pour définir le texte à afficher quand la zone est vide :

En XML : `android:hint="initial"`

En Java : `setHint("String")`

Afin d'activer/désactiver l'`EditText`, le développeur utilise :

En XML : `android:enabled="true/false"`

En Java : `setEnabled(true/false)`

I.3. Button

La classe `Button` permet de créer un bouton pour déclencher un traitement donné. A noter que cette classe, comme la classe `EditText` hérite de la classe `TextView`.

I.4. CheckBox

La classe `CheckBox` est une case à cocher identique au tag `<input type="checkbox"/>` des formulaires HTML.

Afin de cocher/décocher un `Checkbox` :

En XML : `android:checked="true/false"`

En Java : `setChecked(true/false)`

Pour récupérer sa valeur, utiliser la méthode :

`isChecked()`

Pour inverser l'état de la case à cocher, utiliser la méthode :

`toggle()`

I.5. RadioGroup/RadioButton

La classe `RadioButton` est un bouton radio identique au tag `<input type="radio"/>`. Un ensemble de bouton radio peuvent être regroupés dans un seul `RadioGroup` de telle façon l'utilisateur ne peut choisir qu'un seul.

✂ Exemple :

XML :	Interface :
<pre> <TextView android:id="@+id/textView1" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Sélectionner votre option:" /> <RadioGroup android:id="@+id/rdgOption" android:layout_width="wrap_content" android:layout_height="wrap_content"> <RadioButton android:id="@+id/rdDSI" android:layout_width="wrap_content" android:layout_height="wrap_content" android:checked="true" android:text="@string/dsi" /> <RadioButton android:id="@+id/rdRSI" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="@string/rsi" /> <RadioButton android:id="@+id/rdSEM" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="@string/sem" /> <RadioButton android:id="@+id/rdMDW" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="MDW" /> </RadioGroup> </pre>	<div data-bbox="890 239 1388 474"> <p>Sélectionner votre option :</p> <p><input checked="" type="radio"/> DSI</p> <p><input type="radio"/> RSI</p> <p><input type="radio"/> SEM</p> <p><input type="radio"/> MDW</p> </div> <div data-bbox="890 533 1461 1131"> <p>Java :</p> <pre> private RadioGroup rdgOption; rdgOption = (RadioGroup) findViewById(R.id.rdgOption); switch (rdgOption.getCheckedRadioButtonId()) { case R.id.rdDSI: Trait("DSI"); break; case R.id.rdRSI: Trait("RSI"); break; case R.id.rdSEM: Trait("SEM"); break; case R.id.rdMDW: Trait("MDW"); break; } </pre> </div>

`getCheckedRadioButtonId()` permet d'obtenir l'identifiant du `RadioButton` actuellement coché

I.6. ImageView/ImageButton

`ImageView` et `ImageButton` permettent d'intégrer des images dans les activités Android. `ImageButton` est une sous-classe de `ImageView` avec les comportements d'un bouton standard.

Il est possible d'attribuer la source de l'image aux widgets `ImageView` et `ImageButton` :

En XML : `android:src="@drawable/nomImage"`

En Java : `setImageResource(R.drawable.nomImage)`

I.7. ProgressBar/SeekBar

`ProgressBar` est une barre de progression sous forme horizontale ou circulaire. `SeekBar` est un `ProgressBar` sous forme de barre horizontale dotée d'un curseur permettant de modifier sa valeur.

Pour modifier la valeur maximale d'un `ProgressBar/SeekBar`, utiliser :

En XML : `android:max="entier"`

En Java : `setMax("entier")`


Le développeur peut changer la valeur d'un `ProgressBar/SeekBar` :

En XML : `android:progress="entier"`

En Java : `setProgress("entier")`

Afin de récupérer la valeur d'un `ProgressBar/SeekBar`, utiliser :

En Java : `getProgress()`**✂ Exemple :**

Code XML	Aperçu
<pre><SeekBar android:id="@+id/seekBar" android:layout_width="match_parent" android:layout_height="wrap_content" android:max="100" android:progress="70"/></pre>	

I.8. WebView

Avec le composant `WebView`, il est possible d'encapsuler au sein d'une interface graphique classique une fenêtre web affichant le contenu dont l'URL aura été spécifiée.

Pour spécifier le client qui sera chargé d'afficher et de gérer les requêtes et les notifications :

En Java : `setWebViewClient(new WebViewClient())` ou bien
`setWebChromeClient(new WebChromeClient())`

Pour activer/désactiver le JavaScript qui est initialement désactivé :

En Java : `getSettings.setJavaScriptEnabled(true/false)`

Pour charger le contenu à une URL spécifique :

En Java : `loadUrl(String url)`

Exemple :

En Java	Aperçu
<pre>WV.setWebViewClient(new WebViewClient()); //WV.setWebChromeClient(new WebChromeClient()); WV.getSettings().setJavaScriptEnabled(true); WV.loadUrl("http://www.google.com");</pre>	




✍ Remarque :

Afin que l'émulateur et/ou le smartphone puisse se connecter à Internet, ajouter dans le fichier `AndroidManifest.xml` la permission suivante :

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

I.9. DatePicker

Le widget `DatePicker` est un sélecteur de dates. En XML, il offre 3 modes d'affichages de date :

Mode 1 : Par défaut	<pre><DatePicker android:id="@+id/datePicker" android:layout_width="314dp" android:layout_height="209dp" tools:layout_editor_absoluteX="54dp" tools:layout_editor_absoluteY="155dp" android:startYear="2017" android:endYear="2018" android:minDate="01/02/2017" android:maxDate="31/08/2018" /></pre>	
Mode 2 : Spinner	<pre><DatePicker android:id="@+id/datePicker" android:layout_width="304dp" android:layout_height="415dp" android:datepickerMode="spinner" android:calendarViewShown="false" tools:layout_editor_absoluteX="73dp" tools:layout_editor_absoluteY="5dp" /></pre>	
Mode 3 : Calendrier	<pre><DatePicker android:id="@+id/datePicker" android:datepickerMode="calendar" android:layout_width="304dp" android:layout_height="415dp" tools:layout_editor_absoluteX="73dp" tools:layout_editor_absoluteY="5dp" /></pre>	

Les propriétés XML les plus utilisées sont :

- `startYear` et `endYear` : permettent de donner respectivement une année de départ et une année de fin. Elles sont valables uniquement pour les modes 1 et 2.
- `minDate` et `maxDate` : permettent de spécifier une date minimale et une date maximale. Leurs valeurs de dates doivent être sous la forme "mm/jj/aaaa". Egalement, elles sont valables uniquement pour les modes 1 et 2.
- `datepickerMode` : définit le mode d'affichage du `DatePicker`. Par défaut "spinner", cet attribut peut être utilisé pour forcer l'affichage en mode "calendar".
- `calendarViewShown` définit si le calendrier est affiché ou non. Cette propriété est valide seulement pour le mode "spinner".

En programmation JAVA, les méthodes `getDayOfMonth()`, `getMonth()` et `getYear()` permettent de récupérer respectivement le jour, le mois et l'année du `DatePicker` dont la valeur peut être modifiée avec la méthode `updateDate(int year, int month, int dayOfMonth)`.

II. Spinner/ListView

II.1. Présentation

Ces éléments graphiques servent d'aide à la rédaction ou à la sélection d'éléments : Ils permettent de sélectionner rapidement une valeur parmi plusieurs :

1.1. Spinner

Spinner propose une liste de choix. Le choix actuellement sélectionné est affiché, la flèche permet de faire apparaître les autres possibilités sous la forme d'un `RadioGroup`.



1.2. ListView

`ListView` place les éléments en liste avec un ascenseur vertical si nécessaire. `ListView` est normalement utilisé pour afficher des éléments textuels. Il est toutefois possible d'y afficher des éléments plus complexes en utilisant un gestionnaire de contenu.



1.3. Méthodes Java

`ListView` et `Spinner` héritent de la classe `AdapterView`.

Parmi ses méthodes, nous pouvons citer :

- `setSelection()` : définit l'élément actuellement sélectionné
- `getItemAtPosition(int pos)` : récupère les données associées à la position "pos"
- `getCount()` : renvoie le nombre d'éléments
- `getSelectedItem()` : renvoie l'objet sélectionné
- `getSelectedItemPosition()` : renvoie l'indice de l'objet sélectionné

II.2. Gestionnaire de contenu

Les données qui sont affichées dans un `Spinner/ListView` peuvent être gérées de deux façons selon leurs types : statique et dynamique :

2.1. Méthode statique

Il est facile d'ajouter des données statiques à un `Spinner/ListView`. Pour cela, il suffit tout d'abord de déclarer dans le fichier "strings.xml" un tableau de chaînes de caractères contenant les éléments à afficher puis d'affecter à l'attribut `android:entries` de la balise du `Spinner/ListView` la valeur `@array/nomTableau`

✂ Exemple :

Fichier	Code XML
strings.xml	<pre><resources> <string-array name="niveau"> <item>Débutant</item> <item>Intermédiaire</item> <item>Expert</item> </string-array> </resources></pre>
activity_main.xml	<pre><ListView android:layout_width="wrap_content" android:layout_height="wrap_content" android:entries="@array/niveau" /></pre>

2.2. Méthode dynamique

Cette méthode est utilisée pour des données dynamiques à travers un **ArrayAdapter** (collection) que l'on remplit (méthodes `add` ou `insert` du `ArrayAdapter`) et que l'on associe au `Spinner/ListView` grâce à la méthode `setAdapter(ArrayAdapter)`.

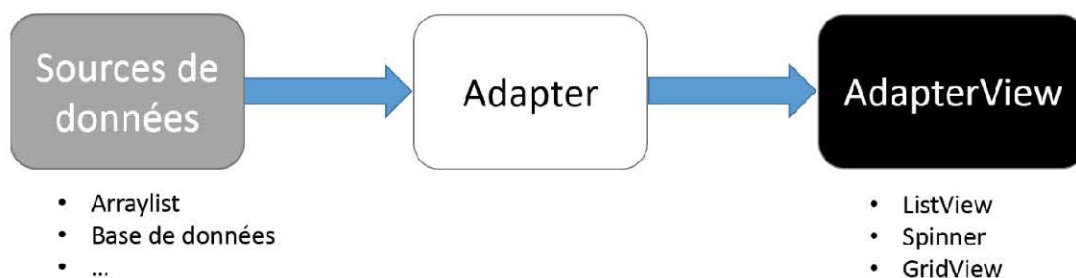


Figure : Fonctionnement des Adapter et des AdapterView

La classe `ArrayAdapter` possède les méthodes suivantes :

- `getItem(int pos)` : renvoie l'élément dont le rang est donné dans la position « pos »
- `getPosition(Object o)` : renvoie le rang de l'élément désigné en paramètre
- `add(Object o)` : pour ajouter l'élément désigné à la fin
- `insert(Object o, int index)` : pour insérer un élément au rang donné en 2^{ème} paramètre
- `remove(Object o)` : pour supprimer un élément
- `getCount()` : renvoie le nombre d'éléments
- `clear()` : pour vider l' `ArrayAdapter`

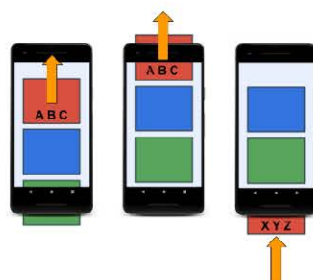
🔧 Exemple :

```
private ArrayAdapter<String> adapter;  
spPension=(Spinner)findViewById(R.id.spPension);  
adapter=new  
ArrayAdapter<String>(this,android.R.layout.simple_list_item_1);  
adapter.add("LPD");  
adapter.add("DP");  
adapter.add("PC");  
adapter.add("All in");  
spPension.setAdapter(adapter);
```

III. RecyclerView

III.1. Présentation

`RecyclerView` est un composant d'interface utilisateur (UI) dans le framework Android qui a été introduit pour remplacer `ListView` comme moyen de gérer les listes et les grilles d'éléments dans une application Android. Il est principalement utilisé pour afficher des données sous forme de listes déroulantes, de grilles ou de carrousels.

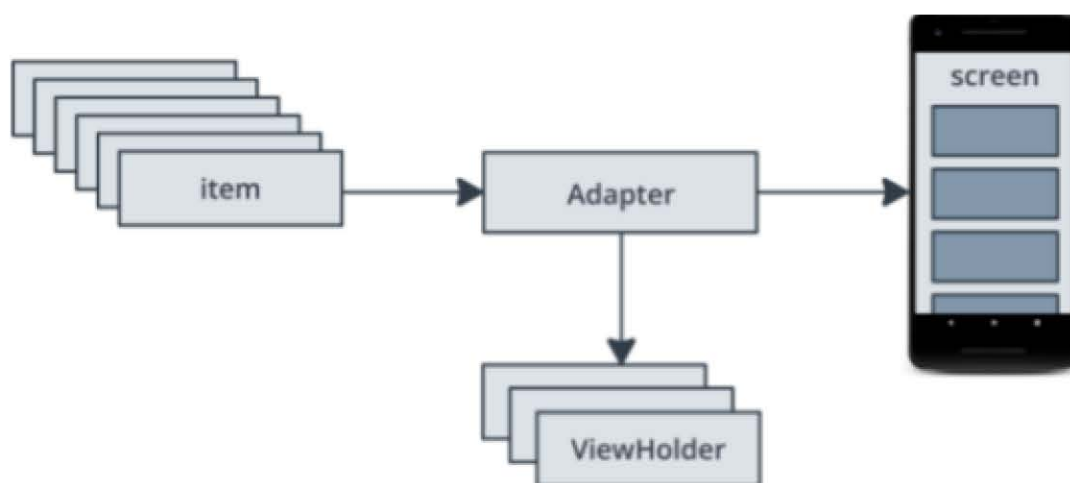


Voici quelques caractéristiques et avantages clés de RecyclerView par rapport à ListView :

- **Performances améliorées** : RecyclerView optimise l'utilisation de la mémoire grâce au recyclage des vues, garantissant une utilisation plus efficace des ressources système.
- **Flexibilité de la mise en page** : Il offre une variété de dispositions, y compris des listes, des grilles et des carrousels, s'adaptant facilement aux besoins de votre application.
- **Animations intégrées** : IL est possible d'ajouter des animations de manière transparente lors de l'ajout, de la suppression ou de la modification des éléments, améliorant ainsi l'expérience utilisateur.
- **Personnalisation avancée** : On peut personnaliser l'apparence et le comportement des éléments de manière approfondie grâce à des adaptateurs et des gestionnaires de mise en page personnalisés.

III.2. Fonctionnement

La création et l'utilisation d'une RecyclerView reposent sur un certain nombre d'éléments, on peut les considérer comme une division du travail. Le schéma ci-dessous donne une vue d'ensemble sur le fonctionnement d'une RecyclerView :



Fonctionnement d'une RecyclerView

- **Un LayoutManager (RecyclerView.LayoutManager)** : Permet de positionner correctement l'ensemble des données de la liste.
- **Un ViewHolder (RecyclerView.ViewHolder)** : Permet de représenter visuellement un élément de la liste de données dans le RecyclerView (Une ligne).
- **Item** : un élément de données de la liste à afficher. Représente le modèle de données de l'application.
- **Un Adapter (RecyclerView.Adapter)** : Permet de faire la liaison (Bind) entre la vue RecyclerView et une liste de données (items).

IV. Éléments graphiques avancés

IV.1. Les Toasts

Un « toast » est un message apparaissant par défaut en bas de l'écran pendant un instant et n'affichant aucun bouton : il n'est pas actif.

1.1. Les Toasts simples

Un Toast simple sert à afficher un message basique, c'est-à-dire uniquement un texte sans image ni style particulier, il suffit :

- d'utiliser la méthode statique `makeText` :


```
Toast t = Toast.makeText(Context, String, int) ;
```

Cette méthode renvoie l'objet de classe Toast créé. Le premier paramètre est généralement l'activité elle-même, le deuxième paramètre est le message à afficher, le dernier paramètre indique la durée d'affichage ; les seules valeurs possibles sont : `Toast.LENGTH_SHORT` (2 secondes) ou `Toast.LENGTH_LONG` (5 secondes).

- d'utiliser la méthode `show()` afin d'afficher le message du Toast pour la durée définie lors de sa création :

```
t.show() ;
```

Exemple :

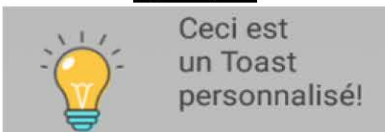
Code	Aperçu
<pre>Toast t=Toast.makeText(getApplicationContext(), "Ceci est un Toast simple",Toast.LENGTH_LONG); t.show();</pre>	

1.2. Les Toasts personnalisés

Il est possible de personnaliser un Toast.

Il faut seulement définir un layout dans `res/layout/toast_layout.xml`. La racine de ce layout doit avoir un identifiant, exemple : `LLT_id` qui est mentionné dans la création.

Exemple :

Fichier : res/layout/toast_layout.xml	Fichier MainActivity.java:
<pre><LinearLayout android:id="@+id/LLT_id" android:orientation="horizontal" android:layout_width="fill_parent" android:layout_height="fill_parent" android:background="#DAAA"> <ImageView android:id="@+id/image" android:layout_width="wrap_content" android:layout_height="fill_parent"/> <TextView android:id="@+id/text" android:layout_width="wrap_content" android:layout_height="fill_parent" android:textSize="24sp"/> </LinearLayout></pre>	<pre>LayoutInflater inflater = getLayoutInflater(); View layout = inflater.inflate(R.layout.toast_layout, (ViewGroup) findViewById(R.id.LLT_id)); ImageView image = (ImageView) layout.findViewById(R.id.image); image.setImageResource(R.drawable.idea); TextView text = (TextView) layout.findViewById(R.id.text); text.setText("Ceci est un Toast personnalisé!"); Toast t = new Toast(this); t.setDuration(Toast.LENGTH_LONG); t.setView(layout); t.show();</pre>
<p><u>Aperçu :</u></p> 	

IV.2. Les Boîtes de dialogues

Une boîte de dialogue est une petite fenêtre qui apparaît au-dessus d'un écran pour afficher ou demander quelque chose d'urgent à l'utilisateur.

 Exemple :

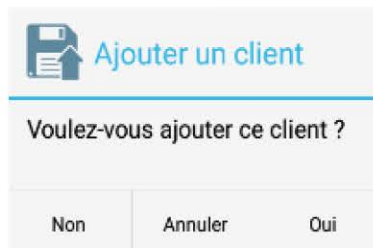


Figure : Exemple de boîte de dialogue

Une boîte de dialogue simple affiche un texte et des boutons (maximum trois). Elle est construite à l'aide d'une classe nommée `AlertDialog.Builder`.

Le principe est de :

- Créer au début un builder et c'est lui qui crée le dialogue :

```
AlertDialog.Builder alertDialog = new AlertDialog.Builder(Activity.this);
```

- Changer ensuite le titre de la boîte de dialogue :

```
alertDialog.setTitle("Ajouter un client");
```

- Modifier également le message de la boîte de dialogue :

```
alertDialog.setMessage("Voulez-vous ajouter ce client ?");
```

- Changer également son icône : `alertDialog.setIcon(R.drawable.save);`

- Configurer par la suite ses boutons ainsi que leurs écouteurs correspondants : il existe trois types de boutons: positif, négatif et neutre

- Bouton positif :

```
alertDialog.setPositiveButton("Oui", new
DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(getApplicationContext(), "Vous avez cliqué sur
Oui", Toast.LENGTH_SHORT).show();} });
```

- Bouton négatif :

```
alertDialog.setNegativeButton("Non", new
DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(getApplicationContext(), "Vous avez cliqué sur
Non", Toast.LENGTH_SHORT).show();} });
```


- Bouton annuler :

```
AlertDialog.setNeutralButton("Annuler", new  
DialogInterface.OnClickListener()  
{  
    public void onClick(DialogInterface dialog, int which) {  
        Toast.makeText(getApplicationContext(), "Vous avez cliqué sur  
Annuler", Toast.LENGTH_SHORT).show(); } });
```

- Afficher finalement l'AlertDialog : `alertDialog.show();`

IV.3. Les menus

Le menu option est une liste d'items qui apparaît soit quand on appuie sur le bouton menu. Ils peuvent être accompagnés d'une icône. Ces menus sont destinés à présenter une liste d'actions ou à fournir des choix de navigation.

Afin de faire un menu d'une application Android, il faudrait créer un fichier `xml` représentant le menu de l'application dans `res/menu` :

Structure du fichier "main_menu.xml"

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">  
    <item  
        android:id="@+id/choix_1"  
        android:icon="@drawable/image_1"  
        android:title="@string/titre_1" />  
    <item  
        android:id="@+id/choix_2"  
        android:icon="@drawable/image_2"  
        android:title="@string/titre_2" />  
    <item ... />  
</menu>
```

Notez bien:

- Chaque élément du menu correspond à la déclaration d'une balise `item`
- Chaque élément possède un identifiant, une icône un titre avec respectivement les attributs `android:id`, `android:icon` et `android:title`.

Une fois que le fichier `xml` relatif au menu est créé, la seconde étape consiste à :

- implémenter la fonction de callback `onCreateOptionsMenu(Menu menu)` afin de charger le menu créé précédemment.

```
protected public boolean onCreateOptionsMenu(Menu menu) {  
    getMenuInflater().inflate(R.menu.main_menu, menu);  
    return true;  
}
```

- implémenter la méthode `onOptionsItemSelected()` afin de spécifier l'action à exécuter en fonction de l'élément cliqué :

```
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.choix_1:  
            .....  
            return true;  
        case R.id.choix_2:  
            .....  
            return true;  
        .....  
        case R.id.choix_n:  
            .....  
            return true;  
    }  
    return super.onOptionsItemSelected(item);  
}
```

Conclusion

Dans ce chapitre, nous avons appliqué dans des exemples les principaux éléments graphiques utilisés par les applications mobiles.

Le chapitre suivant permet d'expliquer le principe de gestion des événements dans Android.