Développement d'applications pour Android

M. Dalmau, IUT de Bayonne-Pays Basque

NOTES

Ce manuel n'a pas la prétention d'être exhaustif, il ne présente que quelques classes de l'API Android (les plus utilisées).
De même pour les classes décrites seules les méthodes les plus utilisées le sont.

SOURCE DU CODE ANDROID

Il est accessible sur : **https://source.android.com/setup**

DOCUMENTATION

Pour une documentation complète d'Android on se reportera au site :
https://developer.android.com/reference/classes qui contient la documentation en ligne des classes de l'API.
L'onglet " Guides" propose des aides pour le développement.
L'onglet " Samples" propose des exemples.

D'autres tutoriels sont disponibles sur bien des sites (attention : certains se contentent de donner des "trucs et astuces" sans explications et parfois même pas très bien maîtrisés par leurs auteurs !).

Un forum très actif est celui de : **https://stackoverflow.com**

Un manuel intitulé "**Développer des applications mobiles pour les Google Phones**" écrit par F. Garin et publié chez Dunod (EAN13 : 9782100531813) peut être téléchargé en pdf sur : **http://kslimi.files.wordpress.com/2011/01/dunod-android.pdf** Il est toutefois un peu ancien.

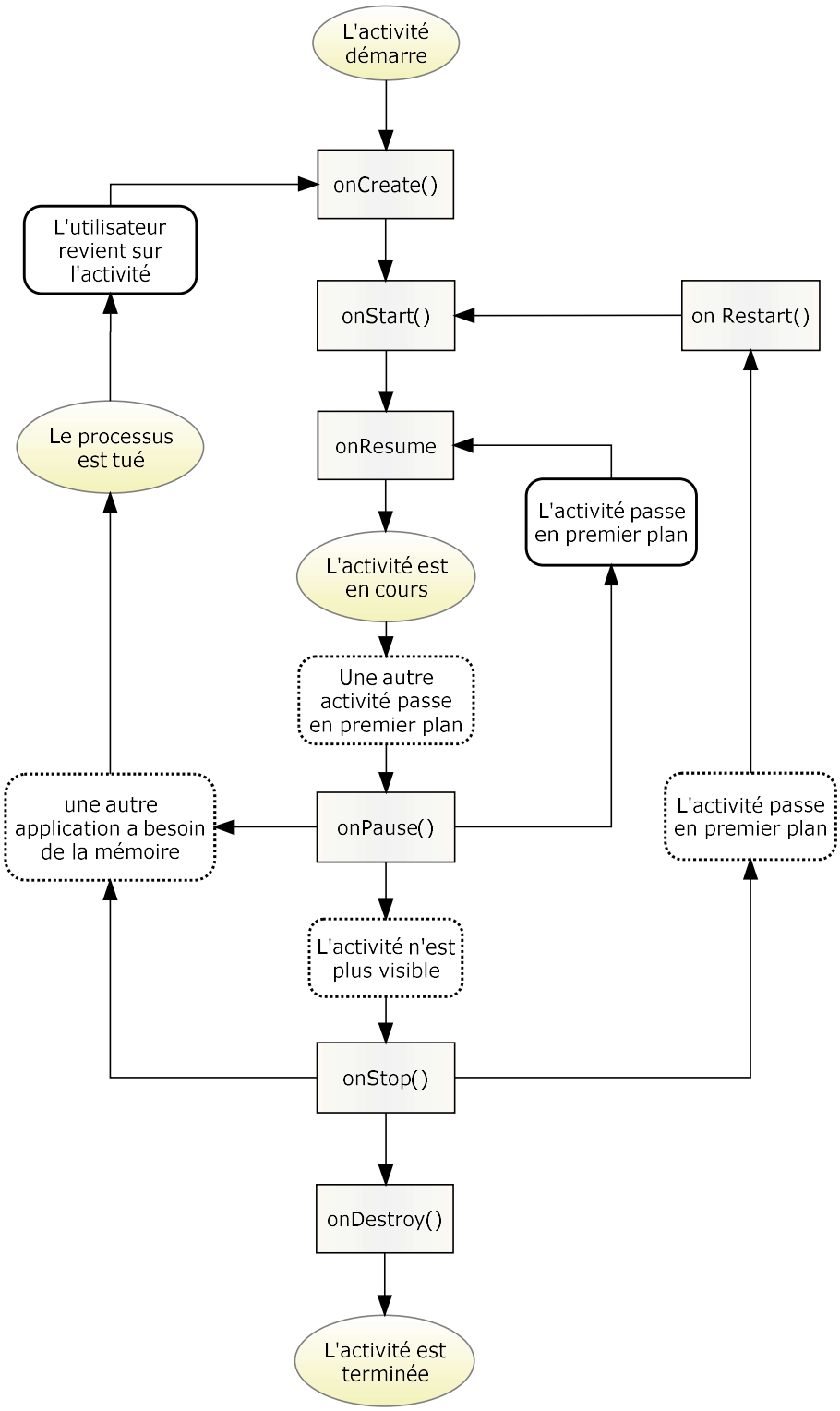
Sommaire

Activité

Une activité est une application ou une partie d'application interactive d'Android.

Cycle de vie d'une activité Android.

Android contrôle le cycle de vie des activités, à chaque étape correspond l'appel d'une méthode que l'on peut surcharger pour définir le comportement correspondant à cette étape.



Principales méthodes de la classe Activity

Toute activité hérite de la classe Activity ou de l'une de ses descendantes (FragmentActivity, AppCompatActivity, …)

Méthodes correspondant au cycle de vie

Les méthodes onXxx sont appelées au cours du cycle de vie. Si elles sont surchargées elles doivent faire appel à leur homologue de la classe supérieure (super.onXxx). Voir schéma du cycle de vie d'une activité ci-dessus.

- `onCreate(Bundle)` appelée à la création. Le paramètre permet de récupérer un état sauvegardé lors de l'arrêt de l'activité (si on a fait une sauvegarde)
- `onPause()` appelée quand l'activité n'est plus en premier plan
- `onDestroy()` appelée quand l'activité se termine
- `onStart()` appelée quand l'activité démarre
- `onRestart()` appelée quand l'activité redémarre
- `onStop()` appelée quand l'activité n'est plus visible
- `onResume()` appelée quand l'activité vient en premier plan
- `finish()` permet de terminer une activité

Méthodes liées au contexte

- `getApplicationContext()` renvoie le contexte de l'application courante

Méthodes relatives à l'interface

- `setContentView(int)` permet de créer l'interface depuis un fichier XML, le paramètre désigne ce fichier par son identifiant
- `findViewById(int)` récupération d'un élément d'interface (renvoie un objet de classe View) , le paramètre désigne cet élément par son identifiant (normalement défini dans la classe R)
- `showDialog(int)` ouverture d'une fenêtre de dialogue. Le paramètre désigne la fenêtre de dialogue par son identifiant
- `showDialog(int, Bundle)` ouverture d'une fenêtre de dialogue. Le 1er paramètre désigne la fenêtre de dialogue par son identifiant, le second permet de lui passer des paramètres
- `dismissDialog(int)` fermeture d'une fenêtre de dialogue désignée par son identifiant
- `onCreateDialog(int, Bundle)` appelée lors de l'ouverture d'une fenêtre de dialogue. Le 1er paramètre est l'identifiant de la fenêtre de dialogue, le second est celui qui avait été passé lors de l'appel de showDialog
- `onPrepareDialog(int, Bundle)` appelée lors de l'activation d'une fenêtre de dialogue déjà ouverte. Le 1er paramètre est l'identifiant de la fenêtre de dialogue, le second est celui qui avait été passé lors de l'appel de showDialog
- `getCurrentFocus()` renvoie l'élément de l'interface (objet de classe View) ayant actuellement le focus
- `runOnUiThread(Thread)` permet de lancer un thread qui peut modifier l'interface ou afficher des Toasts

Méthodes relatives aux menus et menus contextuels

- `onCreateOptionsMenu(Menu)` appelée lorsqu'un menu de l'activité est affiché. Le paramètre est le menu affiché.
- `onOptionsItemSelected(MenuItem)` appelée lors d'un choix dans un menu de l'activité. Le paramètre est le choix effectué.
- `onOptionsMenuClosed(Menu)` appelée lors de la fermeture d'un menu de l'activité. Le paramètre est le menu fermé.
- `registerForContextMenu(View)` associe un menu contextuel à l'élément passé en paramètre
- `unregisterForContextMenu(View)` supprime l'association du menu contextuel à l'élément passé en paramètre
- `onCreateContextMenu(ContextMenu, View, ContextMenu.ContextMenuInfo)` appelée lorsqu'un menu contextuel est affiché. Le 1er paramètre est ce menu contextuel, le 2ème est

l'élément d'interface auquel il est associé, le dernier donne des informations sur le contenu de l'élément d'interface qui a causé l'apparition du menu contextuel
- `onContextItemSelected(MenuItem)` appelée lors d'un choix dans un menu contextuel. Le paramètre est le choix effectué.
- `onContextMenuClosed(ContextMenu)` appelée lors de la fermeture d'un menu contextuel. Le paramètre est le menu fermé.
- `getMenuInflater()` renvoie un objet de classe MenuInflater permettant de créer le contenu d'un menu à partir d'un fichier XML de description

Méthodes relatives à l'environnement et aux ressources

- `getResources()` renvoie un objet de classe `Resources` permettant l'accès aux ressources de l'applicaction (voir plus loin la classe `Resources`)
- `onUserInteraction()` appelée à chaque interaction de l'utilisateur (clavier, trackball, touché)
- `onBackPressed()` appelée lorsque l'utilisateur appuie la touche de retour arrière du téléphone
- `onLowMemory()` appelée lorsque la quantité de mémoire disponible devient faible
- `getFilesDir()` renvoie un objet de classe File représentant le chemin où sont placés les fichiers propres à l'application
- `openFileInput(String)` renvoie un objet de classe FileInputStream correspondant à un flux de lecture connecté au fichier dont le nom est passé en paramètre.
- `openFileOutput(String, int)` renvoie un objet de classe FileOutputStream correspondant à un flux d'écriture connecté au fichier dont le nom est passé en premier paramètre. Le second paramètre est le mode d'ouverture qui peut être : `MODE_PRIVATE` (mode normal), `MODE_APPEND` (ajout à un fichier existant), `MODE_WORLD_READABLE` et `MODE_WORLD_WRITEABLE` (pour les droits d'accès). On peut cumuler plusieurs attributs par |

Méthodes de lancement d'activités et de services

- `startActivity(Intent)` Lance une autre activité, le paramètre désigne l'activité ou les propriétés permettant à Android de l'identifier.
- `startActivityForResult(Intent, int)` Lance une autre activité dont on attend un retour, le 1er paramètre désigne l'activité ou les propriétés permettant à Android de l'identifier, le second paramètre est un numéro associé à cette activité qui permettra d'en identifier la réponse.
- `startNextMatchingActivity(Intent)` Utilisé pour demander à Android de lancer une autre activité lorsque celle proposée précédemment n'a pas donné satisfaction. Le paramètre correspond à celui qui a donné un résultat infructueux.
- `finishActivity(int)` Termine une activité dont on attendait un retour, le paramètre est le numéro associé à cette activité lors de son lancement
- `startService(Intent)` Lance un service, le paramètre désigne le service ou les propriétés permettant à Android de l'identifier
- `stopService(Intent)` Arrête un service, le paramètre désigne le service ou les propriétés permettant à Android de l'identifier
- `getIntent()` renvoie l'objet de classe `Intent` associé à l'activité lors de son lancement
- `setResult(int)` Utilisé par l'activitée appelée pour renvoyer un code de résultat à l'activité appelante
- `setResult(int, Intent)` comme la précédente, le paramètre supplémentaire est un `Intent` contenant les valeurs de retour qui sera également renvoyé.
- `onActivityResult(int, int, Intent)` cette méthode de l'activité appelante est exécutée lorsqu'une activité lancée se termine, le 1er paramètre est le numéro associé à cette activité lors de son lancement, le deuxième est le code de retour de cette activité, le dernier est l'`Intent` de retour de cette activité

Manifeste

Une application est décrite par un fichier XML appelé **AndroidManifest**. Ce fichier permet d'indiquer :

- Le SDK utilisé par une balise `uses-sdk`
- Les permissions : une balise `uses-permission` par autorisation demandée

- Les activités de l'application : une balise `activity` pour chaque activité contenant chacune une ou plusieurs balises `intent-filter`
- Les services de l'application : une balise `service` pour chaque service contenant chacune une ou plusieurs balises `intent-filter`
- les écouteurs d'intentions diffusées : une balise `receiver` pour chaque écouteur contenant chacune une ou plusieurs balises `intent-filter`
- Les fournisseurs de contenu : une balise `provider` pour chaque fournisseur de contenu
- Les bibliothèques utilisées : une balise `uses-library` pour chaque bibliothèque

La structure générale de ce fichier est la suivante :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-sdk />
    <uses-permission />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
        </activity>
        <service>
            <intent-filter> . . . </intent-filter>
        </service>
        <receiver>
            <intent-filter> . . . </intent-filter>
        </receiver>
        <provider>
            <grant-uri-permission />
        </provider>
        <uses-library />
    </application>
</manifest>
```

Activités

Chaque activité constituant une application doit être décrite par une balise `<activity>`. La balise minimale de l'activité principale est de la forme :

```
<activity android:name=".ClasseDeLActivite"
          android:label="nom_de_l_activite"
          android:icon="@drawable/nom_du_fichier_icone"
>
      <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
</activity>
```

La balise `<activity>` peut contenir de nombreux paramètres. Les paramètres généralement utilisés sont :

- `name` qui désigne la classe de l'activité
- `label` qui désigne le nom sous lequel elle apparaîtra sur le terminal
- `icon` qui déésigne l'icône à afficher sur le terminal

Une activité secondaire (lancée par l'activité principale) n'a pas de proriété `android:icon` ni `android:label`. Elle peut ne pas avoir de balise `<intent-filter>` et si elle en a une on n'y rencontrera pas la propriété `<action android:name="android.intent.action.MAIN" />` puisque ce n'est pas un programme principal.

La balise `<intent-filter>` permet d'indiquer ce que fait l'activité (on parle d'intension). Elle sera utilisée par Android pour rechercher une activité pouvant répondre à une spécification donnée (cette spéfication lui sera fournie dans un objet de classe `Intent`). La forme générale de la balise `<intent-filter>` est la suivante:

```
<intent-filter>
    <action android:name="nom_d_action_1" />
    …
    <action android:name="nom_d_action_N" />
    <category android:name="nom_de_categorie_1" />
    …
    <category android:name="nom_de_categorie_N" />
    <data android:mimeType="nom_de_type_mime"
          android:scheme="protocole://hote:port/chemin" />
    />
</intent-filter>
```

On y trouve les rubriques suivantes :

- `Action` qui indique le type d'action effectuée par l'activité (par exemple affichage, édition …)
- `Category` qui indique la catégorie d'action (par exemple CATEGORY_BROWSABLE indique une activité qui peut être appelée par un navigateur)
- `Data` qui indique le type de données transmises à l'activité lancée ou le type de réponse attendu ainsi que le protocole (`http`, `content`, `file` …)

Bien que le concepteur puisse définir ses propres valeurs de paramètres, il existe des valeurs prédéfinies pour ces diverses rubriques dont les principales sont :

- Actions
  - `android.intent.action.VIEW` affichage de données
  - `android.intent.action.EDIT` affichage de données pour édition par l'utilisateur
  - `android.intent.action.MAIN` activité principale d'une application
  - `android.intent.action.CALL` appel téléphonique
  - `android.intent.action.WEB_SEARCH` recherche sur le WEB
- Catégories
  - `android.intent.category.LAUNCHER` activité proposée au lancement par Android
  - `android.intent.category.DEFAULT` activité pouvant être lancée explicitement
  - `android.intent.category.BROWSABLE` peut afficher une information désignée par un lien
  - `android.intent.category.TAB` activité associée dans un onglet d'interface (TabHost)

Permissions

Permissions dans le manifest

Pour autoriser une application à accéder à certaines ressources il faut lui en donner l'autorisation par une balise `<uses-permission>`. Les principales permissions sont :

Géolocalisation (GPS)

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
```

Accès aux données personnelles

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.READ_CALENDAR" />
<uses-permission android:name="android.permission.WRITE_CALENDAR" />
<uses-permission android:name="android.permission.ACCOUNT_MANAGER" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

Modification de paramètres

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

```
<uses-permission android:name="android.permission.SET_TIME_ZONE" />
<uses-permission android:name="android.permission.SET_WALLPAPER" />
<uses-permission android:name="android.permission.CHANGE_CONFIGURATION" />
```

Téléphonie

```
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.CALL_PRIVILEGED" />
<uses-permission android:name="android.permission.ANSWER_PHONE_CALLS" />
<uses-permission android:name="android.permission.READ_PHONE_NUMBERS" />
<uses-permission android:name="android.permission. READ_PHONE _STATE" />
<uses-permission android:name="android.permission.MODIFY_PHONE_STATE" />
```

Envoi et réception de SMS/MMS

```
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.WRITE_SMS" />
<uses-permission android:name="android.permission.BROADCAST_SMS" />
<uses-permission android:name="android.permission.RECEIVE_WAP_PUSH" />
<uses-permission android:name="android.permission.RECEIVE_MMS" />
```

Audio Vidéo

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.CAPTURE_AUDIO_OUTPUT" />
<uses-permission android:name="android.permission.CAPTURE_VIDEO_OUTPUT" />
```

Réseau

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

Access au matériel

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH_PRIVILEGIED" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.BATTERY_STATS" />
<uses-permission android:name="android.permission.BODY_SENSORS" />
<uses-permission android:name="android.permission.SET_ALARM" />
```

Access à la carte SD

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

Permissions lors de l'exécution

Une application peut vérifier si une permission lui est occordée par :

```
if (ContextCompat.checkSelfPermission(this, Manifest.permission.XXX)
                                        != PackageManager.PERMISSION_GRANTED)
```

XXX représente la permission à vérifier (par exemple READ_EXTERNAL_STORAGE).

Une application peut demander une permission à l'utilisateur. Une boîte de dialogue permettra à l'utilisateur d'accorder ou pas cette permission. La méthode *onRequestPermissionsResult* de l'activité sera exécutée lorsque l'utilisateur aura répondu (il faut donc surcharger cette méthode).

Demander une permission :

```
String [] permission = { Manifest.permission.XXX} ;
ActivityCompat.requestPermissions(this, permission, ID_REP) ;
```

- permission est un tableau qui permet de demander plusieurs permissions
- **XXX** représente la permission à demander (par exemple READ_EXTERNAL_STORAGE).
- ID_REP est un entier qui servira à identifier la demande concernée par la réponse dans la méthode *onRequestPermissionsResult*.

Récupérer la réponse :

```
void onRequestPermissionsResult(int code, String[] permissions, int[] resultats) {
    switch (code) {
        case ID_REP:
            if ((resultats.size() !=0 && resultats[0] == PackageManager.PERMISSION_GRANTED)) {
                // La permission est donnée
            }
            else {
                // La permission est refusée
            }
            break;
        …
    }
}
```

Ressources

Répertoires des ressources

Le répertoire *res* contient toutes les ressources qui seront mises dans le fichier application (apk). Il est constitué de sous répertoires :

- mipmap (icones)
- drawable (images .png, .9, .jpg, .gif ou XML)
- color (couleurs définies en XML)
- layout (description en XML des interfaces)
- values (définitions en XML de valeurs : chaînes, tableaux, valeurs numériques …)
- anim (description en XML d'animations)
- menu (description en XML de menus pour l'application)
- xml (fichiers XML utilisés directement par l'application)
- raw (tous les autres types de ressources : fichiers texte, vidéo, son …)
- fonts (fontes de caractères .ttf, .otf, .ttc ou XML)

Remarque : Il est possible de créer d'autres répertoires que ceux indiqués ci-dessus pour des cas particuliers.

Qualification des ressources

Les répertoires contenus dans *res* peuvent être qualifiés en fonction de la taille, de la direction, de l'orientation et de la définition de l'écran ainsi que de la version d'Android et de la langue. Android ira chercher la ressource dans le répertoire correspondant à la configuration du périphérique.

Exemple : si on crée un répertoire *res/layout-fr/* les fichiers XML de descriptions d'interface seront pris dans ce répertoire lorsque le périphérique est en français sinon ils seront pris dans *res/layout*.

Cumul de qualifications : On peut créer des répertoires qualifiés en fonction de :

1. La langue (fr, es, …)
2. La direction de l'écran (ldrtl gauche à droite ou ldltr droite à gauche)
3. La taille de l'écran (small, normal, large, xlarge)
4. L'orientation de l'écran (port, land)

5. La densité de l'écran (ldpi, hdpi, mdpi, xhdpi, xxhdpi, xxxhdpi)[1]
6. La version de l'API (v3, v11 …)[2]

Puis les combiner à condition de respecter l'ordre ci-dessus :
- res/drawable-fr-mdpi-v11 est correct
- res/layout-xlarge-fr est incorrect il faudrait : res/layout-fr-xlarge

Classe de référencement des ressources (classe R)

Android Studio explore les fichiers contenus dans les sous-répertoires de *res* et génére une classe (appelée `R`) dans laquelle un identifiant est créé pour chacun des éléments trouvés :
- Fichiers (images, textes ..) placés dans *drawable-xxxx* , *raw* et *xml*
- Eléments d'interface définis dans les fichiers placés dans *layout* et *menu*
- Valeurs définies dans les fichiers placés dans *values*
- Animations définies dans les fichiers placés dans *anim*

On pourra ensuite, dans la plupart des méthodes, désigner ces ressources par leur identifiant dans cette classe `R` sous la forme : `R.type.nom`.
Par exemple une image "photo.png" placée dans drawable-hdpi sera désignée par `R.drawable.photo`

Ressources de type valeurs

On peut définir des ressources de type valeur (entiers, booléens, chaînes de caractères, etc. et des tableaux) en les décrivant dans des fichiers xml ayant la forme suivante :

```
<?xml version="1.0 " encoding= "utf-8 "?>
<resources>
      <color name= "coulfond">#AA7B03</color>
      <integer name= "limite">567</integer>
      <integer-array name= "codes_postaux">
            <item>64100</item>
            …
            <item>33000</item>
      </integer-array>
      <string name= "titre">Un titre</string>
      <string-array name= "planetes">
            <item>Mercure</item>
            …
            <item>Venus</item>
      </string-array>
      <bool name="actif">true</bool>
      <dimen name "taille">55px</dimen>
</resources>
```

Utilisation de ressources

Les ressources peuvent être utilisées dans les fichiers XML (comme ceux de description d'interface) ou dans le code java.

Référencement d'une ressource dans un fichier XML

On référence une resource dans un fichier XML par :

```
"@[paquetage:]type/identificateur"
```

Par exemple : @string/*machaine* fait référence à une chaîne décrite dans un fichier XML placé dans le répertoire *res/values* (le nom de ce fichier n'a aucune importance) et définie par :

```
<string name="machaine">contenu de cette chaine</string>
```

[1] ldpi=200x320, mdpi=320x480, hdpi=480x800, xhdpi =720x1280, xxhdpi =960x1600, xxxhdpi =1440x2560
[2] Les versions sont celles de l'API soit de V1 (android 1.0) à V28 (Android 9)

Récupération d'une ressource dans le code (la classe Resources)

Dans le code les ressources sont désignées par leur identifiant dans la classe `R` de la forme : `R.type.nom`. Certaines méthodes acceptent cette désignation comme paramètre pour accéder à la ressource. Toutefois lorsque l'on doit accéder au contenu de ressources on peut faire applet à la classe `Resources`.

import android.content.res.Resources

On obtient une instance de la classe `Resources` par la méthode `getResources()` de l'activité. On accède ensuite aux ressources grâce aux méthodes de cet objet qui acceptent en paramètre un identifiant de la forme `R.type.nom`, ces méthodes sont :

- boolean `getBoolean`(int) renvoie le booléen désigné
- int `getInteger`(int) renvoie l'entier désigné
- int[] `getIntArray`(int) renvoie le tableau d'entiers désigné
- String `getString`(int) renvoie la chaîne désignée
- String[] `getStringArray`(int) renvoie le tableau de chaînes désigné
- int `getColor`(int) renvoie le code de la couleur désignée
- float `getDimension`(int) renvoie la valeur de l'unité désignée
- Drawable `getDrawable`(int) renvoie l'image désignée (formats possibles du fichier placé dans le répertoire res : gif , jpeg , png et bmp)
- InputStream `openRawResource`(int) renvoie un flux de lecture sur la ressource désignée.

Par exemple : String titre = getResources().getString(R.string.*texte_titre*);
permet de récupérer la chaîne définie dans un fichier XML par : <string name="*texte_titre*">…..</string>

Uri

Les Uri sont un moyen utilisé pour désigner des resources que ce soit sur Internet ou localement

Ressource sur Internet

Pour créer une Uri faisant référence à une ressource sur Internet :

```
Uri ressource =
Uri.parse("http://domaine.sous_domaine/chemin/nom_du_fichier");
```

Exemple : Uri.parse("http://www.iutbayonne.univ-pau.fr/~dalmau/testTPs/ma_video.3gp");
Désigne le fichier vidéo *ma_video.3gp* accessible sur http://www.iutbayonne.univ-pau.fr/~dalmau/testTPs/

Ressource locale

Pour créer une Uri faisant référence à une ressource embarquée :

```
Uri ressource =
Uri.parse("android.resource://nom_du_paquetage_de_l_activité/" +
R.chemin.mon_son);
```

Exemple : Uri.parse("android.resource://iut.TP/" + R.raw.mon_son");
Désigne le fichier son *mon_son* placé dans le répertoire *res/raw* de l'activité dont le paquetage est *iut.TP*

Fichiers et répertoires sur Android

Répertoires d'Android

import android.os.Environment

Android permet la manipulation de fichiers de façon classique en java (classe File). Certains répertoires ont un rôle particulier dans le système. On peut les obtenir grâce l'activité ou à la classe `Environment` comme suit :

- Répertoire propre à l'application : on l'obtient par la méthode `getFilesDir`() de l'activité
- Répertoire de données : on l'obtient par `Environment.getDataDirectory`()
- Répertoire des téléchargements : on l'obtient par `Environment.getDownloadCacheDirectory`()

- Répertoire de stockage externe (en général une carte SD) : on l'obtient par `Environment.getExternalStorageDirectory()`
- Répertoire racine d'Android : on l'obtient par `Environment.getRootDirectory()`

Toutes ces méthodes renvoient un objet de classe File. Lorsqu'il s'agit d'un répertoire il peut être utilisé pour ouvrir un fichier par : `File monFichier = new File(repertoire, "nom_du_fichier");`

ATTENTION : certains répertoires peuvent être totalement fermés ou n'autoriser que la lecture.

La classe File

import java.io.File

C'est une classe java qui n'est pas propre à Android, elle permet de manipuler des fichiers et des répertoires

Méthodes de la classe File

Ouverture

- `File`(String, String) ouvre un fichier, le premier paramètre est le nom du chemin, le second le nom du fichier
- `File`(File, String) ouvre un fichier, le premier paramètre est le répertoire, le second le nom du fichier
- `File`(URI uri) ouvre un fichier, le paramètre désigne ce fichier sous forme d'URI (de la forme file:/chemin/nom). Ce constructeur peut lever une exception de classe `IllegalArgumentException` si le paramètre n'est pas une URI correcte

Création

- `createNewFile`() crée le fichier correspondant à l'objet File. Cette méthode peut lever une exception de classe `IOException` si le fichier ne peut pas être créé.

Supression

- `delete`() supprime le fichier, cette méthode renvoie on booléen indiquant si la destruction a pu se faire ou pas
- `deleteOnExit`() le fichier sera supprimé lorsque la machine virtuelle java se terminera si elle se termine normalement

Etats

- `exists`() renvoie true si le fichier ou le répertoire existe
- `canRead`() renvoie true si le fichier ou le répertoire peut être lu
- `canWrite`() renvoie true si le fichier ou le répertoire peut être écrit
- `isDirectory`() renvoie true si c'est un répertoire
- `isFile`() renvoie true si c'est un fichier
- `isHiden`() renvoie true si le fichier ou le répertoire est caché
- `length`() renvoie la taille du fichier (ne fonctionne pas pour un répertoire)
- `setReadOnly`() positionne le fichier ou le répertoire en lecture seule

Noms

- `getName`() renvoie la partie correspondant au nom du fichier ou du répertoire (String)
- `getPath`() renvoie la partie correspondant au chemin du fichier ou du répertoire (String)
- `getAbsolutePath`() renvoie le chemin absolu du fichier ou du répertoire (String)
- `getCanonicalPath`() renvoie le chemin relatif du fichier ou du répertoire (String). Cette méthode peut lever une exception de classe `IOException`.
- `toURI`() renvoie le fichier ou le répertoire sous forme d'URI (de la forme file:/chemin/nom)

Répertoires

- `list`() renvoie la liste des noms des fichiers contenus dans le répertoire (String[])

- `listFiles`() renvoie la liste des fichiers contenus dans le répertoire (File[])
- `mkdirs`() créer le répertoire désigné par l'objet, si nécessaire les répertoires du chemin son également crées

Lecture/écriture dans un fichier

On utilise des flux de lecture ou d'écriture de classe FileInputStream et FileOutputStream qui sont construits à partir de l'objet de classe File par :

- new `FileInputStream`(`File`)
- new `FileOutputStream`(`File`)

Puis, à partir de ces flux liés au fichier, on construit des flux adaptés aux types de données à lire ou écrire.

Interfaces

Mise en place d'une interface

Elle peut se faire de deux façons :

- Par description de l'interface dans des fichiers XML
- Par programme

Les fichiers XML qui décrivent une interface sont placés dans le répertoire *`res/layout`*.
Ils sont référencés par `R.layout.`*nom_du_fichierXML*.

Les activités peuvent utiliser la méthode `setContentView`(`R.layout.`*nom_du_fichierXML*) pour mettre en place l'interface décrite par un tel fichier.

Leur forme générale est :

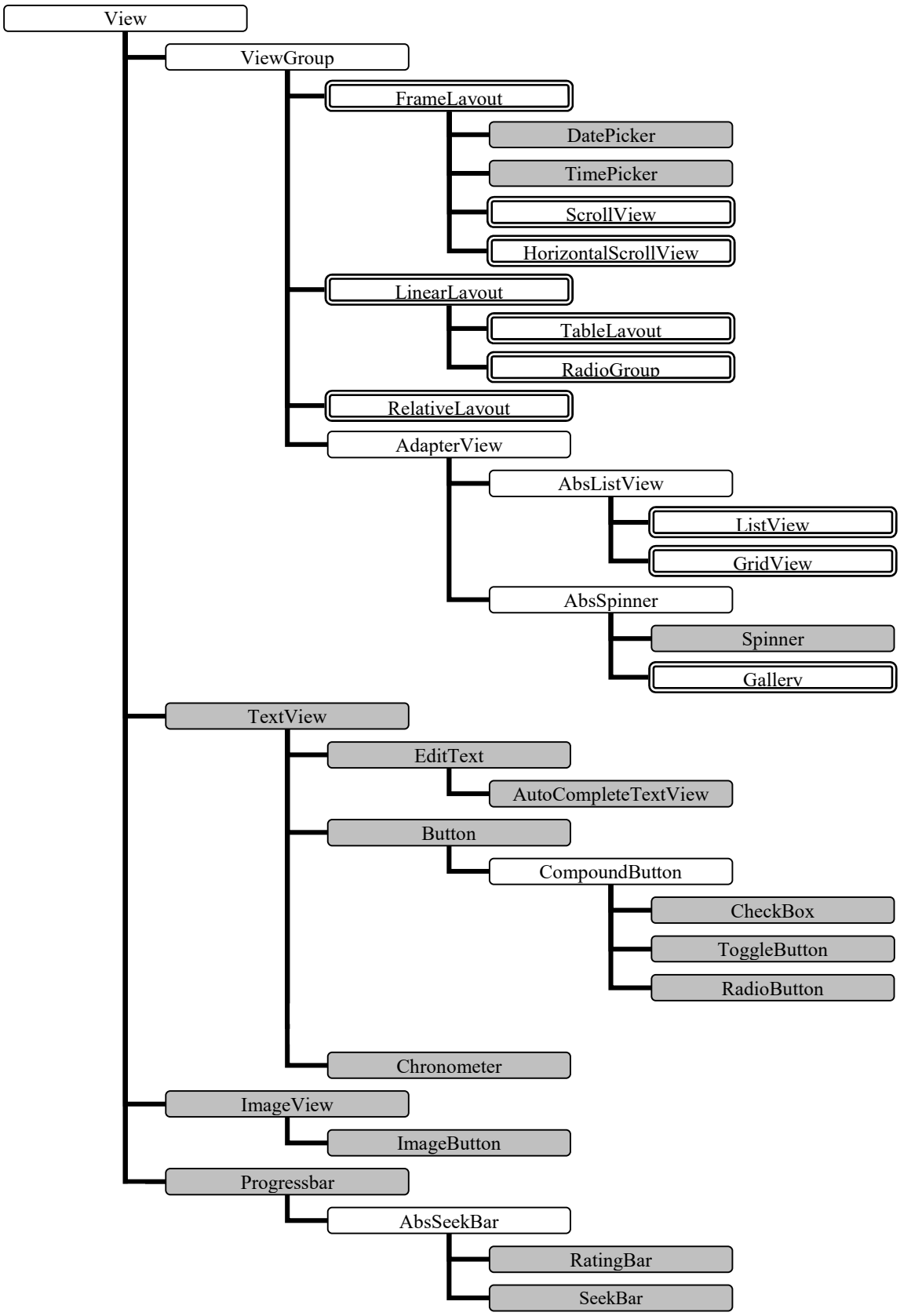
```
<?xml version="1.0" encoding="utf-8"?>
<!— Commentaire -->

<Classe_du_conteneur_principal
       xmlns:android="http://schemas.android.com/apk/res/android"
       propriétés du conteneur principal
>
    <Classe de conteneur ou d'élément d'interface
      propriétés  du conteneur ou de l'élément d'interface
    />
   …
    <Classe de conteneur ou d'élément d'interface
      propriétés  du conteneur ou de l'élément d'interface
    />
</Classe_du_conteneur_principal>
```

Lorsque l'un des éléments est un conteneur, il faut indiquer les éléments qu'il contient :

```
<Classe_de_conteneur
       propriétés du conteneur (voir ci-dessous)
>
    <Classe de conteneur ou d'élément d'interface
      propriétés  du conteneur ou de l'élément d'interface
    />
   …
    <Classe de conteneur ou d'élément d'interface
      propriétés  du conteneur ou de l'élément d'interface
   />
</Classe_du_conteneur>
```

Hiérarchie (partielle) des classes pour les interfaces



Propriétés et classes de base des interfaces

Les unités

Lorsque l'on indique des unités dans les fichiers XML elles peuvent l'être de plusieurs façons :

- en pixels (faire suivre la valeur de **px**)
- relativement à la taille de la fonte de caractères (faire suivre la valeur de **sp**)
- en millimètres (faire suivre la valeur de **mm**)
- en pouces (faire suivre la valeur de **in**)
- en points = 1/72 pouce (faire suivre la valeur de **pt**)
- en pixels à densité indépendante = 1 pixel pour un écran de 160 dpi (faire suivre la valeur de **dp**)

La valeur est exprimée par un réel de la forme "14.3mm" par exemple.
Dans ce qui suit on désignera les unités par "unité"

Les couleurs

Elles sont définies par un code hexadécimal indiquant la transparence et les composantes (RVB) sous la forme "#AARRVVBB" où :

- AA est la transparence (FF opaque, 00 totalement transparent). Si le code de couleur ne comporte que 3 composantes AA est considéré comme FF c'est-à-dire que #44FA8B est équivalent à #FF44FA8B
- RR VV et BB sont, respectivement, les composantes rouge, verte et bleue

Dans ce qui suit on désignera les couleurs par "couleur"

La classe View

Les classes View et ViewGroup ne sont pas utilisées directement mais constituent les classes mères des autres. Elles sont décrites ici pour éviter de répéter leurs propriétés dans les autres classes d'interface.
View est la classe dont héritent toutes les classes utilisées pour réaliser des interfaces. Ses propriétés et ses méthodes se retrouvent donc dans tous les éléments d'interface.

Propriétés positionnables en XLM

Identifiant

Un identifiant peut être associé à chaque élément décrit dans un fichier XML, cet identifiant permet d'accéder à l'objet créé dans le code. Les éléments ne devant pas être accédés dans le code peuvent ne pas avoir d'identifiant.

- `android:id="@+id/monident"` permettra de retrouver cet élément par `findViewById(R.id.monident)`.

Visibilité

- `android:visibility="x"` (les valeurs possibles pour x sont : `visible`, `invisible` ou `gone`)
  Avec `invisible` la place est conservée, avec `gone` la place n'est pas conservée.

Fond

- `android:background="couleur"` pour définir une couleur ou `android:background="@drawable/monimage"` pour utiliser une image. L'image est placée dans `res/drawable-xxxx/` et s'appelle monimage.t (où t est `png`, `jpg`, ou `gif`).
  ATTENTION : les noms des images ne doivent utiliser que des minuscules ou des chiffres.

Taille et marges internes

- `android:minHeight="unité"` définit la hauteur minimale si possible
- `android:minWidth="unité"` définit la largeur minimale si possible
- `android:paddingBottom="unité"` marge interne en bas
- `android:paddingLeft="unité"` marge interne à gauche
- `android:paddingRight="unité"` marge interne à droite
- `android:paddingTop="unité"` marge interne en haut

Ascenseurs

Ces paramètres ne permettent pas d'ajouter des ascenseurs mais seulement de gérer leur comportement lorsqu'ils existent.

- `android:fadeScrollbars="b"` où b vaut true ou false selon que l'on souhaite que les ascenseurs disparaissent ou pas lorsqu'ils ne sont pas utilisés.
- `android:scrollbarDefaultDelayBeforeFade="x"` où x est un entier qui définit le délai (en ms) avant que les ascenseurs non utilisés ne disparaissent
- `android:scrollbarFadeDuration="x"` où x est un entier qui définit la durée d'effacement des ascenseurs (en ms)

Evénements

- `android:clickable="b"` (où b vaut true ou false).
- `android:longClickable="b"` (où b vaut true ou false).

Méthodes de la classe View :

import android.view.View

Construction

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

Placement

- `setLayoutParams(ViewGroup.LayoutParams)` permet de définir le placement des objets dans la vue. Le paramètre de cette méthode est généralement new `ViewGroup.LayoutParams(l, h)` où l définit la largeur et h la hauteur. Les valeurs pour l et h peuvent être exprimées en pixels ou prendre les valeurs `ViewGroup.LayoutParams.WRAP_CONTENT` pour adapter la taille au contenu ou `ViewGroup.LayoutParams.FILL_PARENT` pour adapter la taille à la place disponible.

Position

- `bringToFront()` met la vue en premier plan
- `getLeft()` renvoie la position en pixels de la gauche de la vue
- `getTop()` renvoie la position en pixels du haut de la vue
- `getRight()` renvoie la position en pixels de la droite de la vue
- `getBottom()` renvoie la position en pixels du bas de la vue

Taille et marges internes

- `getWidth()` renvoie la largeur de la vue en pixels
- `getHeight()` renvoie la hauteur de la vue en pixels
- `setPadding(int, int, int, int)` définit les marges (en pixels). Les paramètres sont, dans l'ordre : gauche, haut, droite, bas.
- `getPaddingLeft()` renvoie la marge gauche (en pixels).
- `getPaddingTop()` renvoie la marge haute (en pixels).
- `getPaddingRight()` renvoie la marge droite (en pixels).
- `getPaddingBottom()` renvoie la marge basse (en pixels).

Fond

- `setBackgroundColor(int)` définit la couleur du fond
- `setBackgroundDrawable(Drawable)` définit une image de fond
- `setBackgroundResource(int)` définit le fond à partir d'un identificateur de ressource

Rafraichissement

- `requestLayout()` provoque un rafraîchissement des positions des éléments de la vue
- `invalidate()` provoque un rafraîchissement de la vue

Focus

- `isFocusable()` renvoie true si la vue peut prendre le focus
- `setFocusable(boolean)` fait ou pas prendre le focus à la vue
- `isFocusableInTouchMode()` renvoie true si la vue peut prendre le focus lorsqu'elle est touchée
- `setFocusableInTouchMode(boolean)` autorise ou non la vue a prendre le focus lorsqu'elle est touchée

Evénements

- `setOnClickListener(OnClickListener)` associe un écouteur d'événements aux clics sur la vue
- `setOnLongClickListener(OnLongClickListener)` associe un écouteur d'événements aux clics longs sur la vue
- `setOnKeyListener(OnKeyListener)` associe un écouteur d'événements aux actions clavier sur la vue
- `setOnTouchListener(OnTouchListener)` associe un écouteur d'événements aux touchés sur la vue

La classe ViewGroup

Propriétés positionnables en XLM pour les éléments placés à l'intérieur

Taille

- `android:layout_height="t"` (où t peut être une unité ou prendre les valeurs : `FILL_PARENT` ou `WRAP_CONTENT`) pour occuper tout l'espace en hauteur ou seulement ce qui est nécessaire
- `android:layout_width="t"` (où t peut être une unité prendre les valeurs : `FILL_PARENT` ou `WRAP_CONTENT`) pour occuper tout l'espace en largeur ou seulement ce qui est nécessaire

Marges externes

- `android:layout_marginBottom="unité"` marge externe en bas
- `android:layout_marginLeft="unité"` marge externe à gauche
- `android:layout_marginRight="unité"` marge externe à droite
- `android:layout_marginTop="unité"` marge externe en haut

La méthode correspondant à ces 4 paramètres est : `setPadding(int, int, int, int)`

Méthodes de la classe ViewGroup :

import android.view.ViewGroup

Construction

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

Eléments du groupe

- `addView(View)` ajoute un élément au groupe
- `getChildCount()` renvoie le nombre d'éléments du groupe
- `getFocusedChild()` renvoie l'élément qui a actuellement le focus (objet de classe View)
- `removeView(View)` enlève un élément du groupe
- `removeAllViews()` enlève tous les éléments du groupe

Marges

- `setPadding(int, int, int, int)` définit les marges (en pixels). Les paramètres sont, dans l'ordre : gauche, haut, droite, bas.

Evenements

Mêmes méthodes que la classe View plus :

- `setOnHierarchyChangeListener(OnHierarchyChangeListener)` associe un écouteur d'événements aux ajouts ou suppressions d'éléments dans le groupe

Les conteneurs

Les conteneurs sont utilisés pour placer des éléments d'interface ou d'autres conteneurs. Ils héritent tous de ViewGroup. Leurs propriétés sont donc au moins celles décrites ci-dessus pour View et ViewGroup auxquelles viennent s'ajouter des propriétés spécifiques décrites dans cette partie.

La classe FrameLayout

Placement en haut à gauche. Si l'on place plusieurs éléments ils se superposent, généralement FrameLayout est utilisé pour ne placer qu'un seul élément.

Propriétés supplémentaires positionnables en XLM

Couleurs ou images

- `android:foreground="couleur"` pour définir une couleur ou `android:foreground="@drawable/monimage"` pour utiliser une image. L'image est placée dans `res/drawable/` et s'appelle monimage.x (où x est `png`, `jpg`, ou `gif`).
- `android:foregroundGravity="g"` (où g peut prendre les valeurs : `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`) utilisé lorsque `android:foreground` est défini par une image pour donner sa position.

Méthodes de la classe FrameLayout

android.widget.FrameLayout

Construction

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

Fond

- `setForeground(Drawable)` définit une image comme fond
- `getForeground()` renvoie l'image de fond (objet de classe `Drawable`)
- `setForegroundGravity(int)` Le paramètre est l'une des constantes définies dans la classe `Gravity` on utilise donc `Gravity`. `TOP`, `BOTTOM`, `LEFT`, `RIGHT`, `CENTER_VERTICAL`, `FILL_VERTICAL`, `CENTER_HORIZONTAL`, `FILL_ HORIZONTAL`, `CENTER`, `FILL`

Evenements

Mêmes méthodes que la classe ViewGroup

La classe LinearLayout

Place les éléments les uns à coté des autres horizontalement ou verticalement.

Propriétés supplémentaires positionnables en XLM

- `android:orientation="o"` (où o peut prendre les valeurs : `vertical` ou `horizontal`)
- `android:weightSum="x"` (où x est un réel entre 0 et 1) désigne la somme des poids de tous les éléments mis dans le LinearLayout. Un paramètre de type `android:layout_weight="x"` peut être

associé à chacun des éléments placés dans le LinearLayout pour indiquer leur poids de redimensionnement relatif à la valeur de `layout_weightSum`.
- `android:gravity="g"` (où g peut prendre les valeurs : `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`) définit comment se placent les éléments contenus par ce conteneur.

Méthodes de la classe LinearLayout
android.widget.LinearLayout

Construction
- `LinearLayout(Context)` le paramètre est généralement l'activité elle-même

Position et disposition
- `getOrientation()` renvoie un entier qui vaut `LinearLayout.HORIZONTAL` ou `LinearLayout.VERTICAL`
- `setOrientation(int)` définit l'orientation, le paramètre est un entier qui vaut `LinearLayout.HORIZONTAL` ou `LinearLayout.VERTICAL`
- `getWeightSum()` renvoie un réel correspondant à la somme des poids associée au LinearLayout
- `setWeightSum(float)` définit la somme des poids associée au LinearLayout
- `setGravity(int)` Le paramètre est l'une des constantes définies dans la classe `Gravity` on utilise donc `Gravity`. (`TOP`, `BOTTOM`, `LEFT`, `RIGHT`, `CENTER_VERTICAL`, `FILL_VERTICAL`, `CENTER_HORIZONTAL`, `FILL_ HORIZONTAL`, `CENTER`, `FILL`)

Evenements
Mêmes méthodes que la classe ViewGroup

La classe GridLayout
Place les éléments dans un tableau.

Propriétés supplémentaires positionnables en XLM
- `android:columnCount="v"` (nombre de colonnes)
- `android:lineCount="v"` (nombre de lignes)

Méthodes de la classe GridLayout

Construction
- `GridLayout(Context)` le paramètre est généralement l'activité elle-même

Dimensions
- `getColumnCount()` renvoie le nombre de colonnes
- `getRowCount()` renvoie le nombre de lignes
- `setColumnCount(int)` définit le nombre de colonnes
- `setRowCount(int)` définit le nombre de lignes

Evénements
Mêmes méthodes que la classe ViewGroup

Les classe ScrollView et HorizontalScrollView
Un ScrollView est une zone à défilement vertical tandis qu'un HorizontalScrollView permet un défilement horizontal. Ils ne peuvent contenir qu'un seul élément. Généralement ils sont utilisés pour contenir un LinearLayout (dont l'orientation correspond à celle du ScrollView) et offrir des ascenseurs. Lorsque l'on veut avoir un défilement horizontal et vertical il suffit de mettre un HorizontalScrollView dans un ScrollView.

ATTENTION : En raison de l'écran tactile il n'est pas possible de mettre une zone avec ascenseurs dans une zone ayant elle-même des ascenseurs car le défilement se fera toujours sur la zone la plus externe (sur le contenant et non sur le contenu).

Propriétés supplémentaires positionnables en XLM

Mêmes paramètres XML que FrameLayout plus :

- `android:fillViewport="b"` (où b vaut true ou false) indique si le contenu doit être étiré pour occuper la place disponible ou pas.

Méthodes des classes ScrollView et HorizontalScrollView

android.widget.ScrollView
android.widget.HorizontalScrollView

Construction

- `ScrollView(Context)` le paramètre est généralement l'activité elle-même
- `HorizontalScrollView(Context)` le paramètre est généralement l'activité elle-même

Défilement

- `smoothScrollBy(int, int)` permet de faire défiler la vue. Les paramètres indiquent le nombre de pixels de défilement (resp horizontalement et verticalement)
- `smoothScrollTo(int, int)` permet de faire défiler la vue. Les paramètres indiquent la nouvelle position de la vue en pixels (resp horizontalement et verticalement)
- `awakenScrollBars()` provoque l'affichage des ascenseurs (qui s'effaceront au bout d'un moment).
- `isHorizontalScrollBarEnabled()` renvoie true si l'ascenseur horizontal est autorisé.
- `setHorizontalScrollBarEnabled(boolean)` active ou pas l'ascenseur horizontal.
- `isVerticalScrollBarEnabled()` renvoie true si l'ascenseur vertical est autorisé.
- `setVerticalScrollBarEnabled(boolean)` active ou pas l'ascenseur vertical.

Placement

- `setFillViewport(boolean)` permet de définir si le contenu doit être étiré pour occuper la place disponible (paramètre à true) ou pas (paramètre à false)

Evénements

Mêmes méthodes que la classe ViewGroup

La classe TableLayout

Dispose les éléments en lignes et colonnes. Chaque ligne est désignée dans le fichier XML par `<TableRow>` … `</TableRow>`. Elle contient 1 ou plusieurs éléments. Le nombre de colonnes est déterminé par la ligne contenant le plus d'éléments. TableLayout n'a pas d'ascenseurs, on peut les ajouter en l'incluant dans un ScrollView ou HorizontalScrollView.

Propriétés supplémentaires positionnables en XLM

- `android:collapseColumns="x,y, …"` (où x, y, ... sont des numéros de colonnes à cacher (démarrant à 0)).
- `android:shrinkColumns="x,y, …"` (où x, y, ... sont des numéros de colonnes qui peuvent être rétrécies en fonction de la place disponible (démarrant à 0)).
- `android:stretchColumns="x,y, …"` (où x, y, ... sont des numéros de colonnes qui peuvent être agrandies en fonction de leur contenu (démarrant à 0)).

Propriétés positionnables en XLM pour les éléments placés à l'intérieur

- `android:layout_span="x"` (où x est un entier indiquant sur combien de colonnes s'étend l'élément (par défaut 1))

Méthodes de la classe TableLayout
android.widget.TableLayout

Construction

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

Aspect

- `isColumnCollapsed(int)` indique si la colonne dont le numéro est donné en paramètre est cachée
- `setColumnCollapsed(int, boolean)` cache ou pas (second paramètre) la colonne dont le numéro est donné en premier paramètre
- `isColumnShrinkable(int)` indique si la colonne dont le numéro est donné en paramètre est rétrécissable
- `setColumnShrinkable(int, boolean)` rend ou pas rétrécissable (second paramètre) la colonne dont le numéro est donné en premier paramètre
- `isColumnStretchable(int)` indique si la colonne dont le numéro est donné en paramètre est étirable
- `setColumnStretchable(int, boolean)` rend ou pas étirable (second paramètre) la colonne dont le numéro est donné en premier paramètre
- `isShrinkAllColumns()` indique si toutes les colonnes sont rétrécissables
- `setShrinkAllColumns(boolean)` rend toutes les colonnes (ou aucune) rétrécissables
- `isStretchAllColumns()` indique si toutes les colonnes sont étirables
- `setStretchAllColumns(boolean)` rend toutes les colonnes (ou aucune) étirables

Evénements

Mêmes méthodes que la classe ViewGroup

La classe RelativeLayout

Permet de placer des éléments les uns relativement aux autres ou relativement au conteneur.

Propriétés positionnables en XLM pour les éléments placés à l'intérieur

Pour définir la position d'un élément par rapport à un autre ou par rapport au `RelativeLayout` lui-même (parent) on ajoute dans la balise de cet élément :

Pour les positions par rapport au `RelativeLayout` :

- `android:layout_alignParentBottom="b"` (où b vaut true ou false)
- `android:layout_alignParentLeft="b"` (où b vaut true ou false)
- `android:layout_alignParentRight="b"` (où b vaut true ou false)
- `android:layout_alignParentTop="b"` (où b vaut true ou false)
- `android:layout_centerHorizontal="b"` (où b vaut true ou false)
- `android:layout_centerInParent="b"` (où b vaut true ou false)
- `android:layout_centerVertical="b"` (où b vaut true ou false)

Pour les positions par rapport à d'autres éléments désignés par leur identifiant :

- `android:layout_above="@+id/ident"`
- `android:layout_below="@+id/ident"`
- `android:layout_toLeftOf="@+id/ident"`
- `android:layout_toRightOf="@+id/ident"`
- `android:layout_alignLeft="@+id/ident"`
- `android:layout_alignRight="@+id/ident"`
- `android:layout_AlignTop="@+id/ident"`
- `android:layout_AlignBottom="@+id/ident"`

Méthodes de la classe RelativeLayout
android.widget.RelativeLayout

Construction

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

Disposition

- `setGravity(int)` Le paramètre est l'une des constantes définies dans la classe Gravity on utilise donc Gravity. `(TOP, BOTTOM, LEFT, RIGHT, CENTER_VERTICAL, FILL_VERTICAL, CENTER_HORIZONTAL, FILL_ HORIZONTAL, CENTER, FILL)`

Evénements

Mêmes méthodes que la classe ViewGroup

La classe ConstraintLayout

Permet de placer des éléments les uns relativement aux autres comme un RelativeLayout mais offre plus de possibilités :

- Placement circulaire
- Marges selon que l'autre élément est présent ou pas
- Poids de répartition des marges.

Propriétés positionnables en XLM pour les éléments placés à l'intérieur

Positionnement par rapport à d'autres éléments ou par rapport au conteneur lui même :

- Pour positionner par rapport à un autre élément on référence cet élément par son identifiant ("@+id/ident")
- Pour positionner par rapport au conteneur on indique "parent"

Dans ce qui suit XXX est donc remplacé par **@+id/ident** ou par **parent** :

- `app:layout_constraintLeft_toLeftOf="XXX"`
- `app:layout_constraintLeft_toRightOf="XXX"`
- `app:layout_constraintRight_toLeftOf="XXX"`
- `app:layout_constraintRight_toRightOf="XXX"`
- `app:layout_constraintTop_toTopOf="XXX"`
- `app:layout_constraintTop_toBottomOf="XXX"`
- `app:layout_constraintBottom_toTopOf="XXX"`
- `app:layout_constraintBottom_toBottomOf="XXX"`
- `app:layout_constraintBaseline_toBaselineOf="XXX"`
- `app:layout_constraintStart_toEndOf="XXX"`
- `app:layout_constraintStart_toStartOf="XXX"`
- `app:layout_constraintEnd_toStartOf="XXX"`
- `app:layout_constraintEnd_toEndOf="XXX"`
- `app:layout_constraintCircle="XXX"`
- `app:layout_constraintCircleRadius="rayon"`
- `app:layout_constraintCircleAngle="angle"`

Marges si l'autre élémént est présent :

- `app:layout_marginStart="@+id/ident"`
- `app:layout_marginEnd="@+id/ident"`
- `app:layout_marginLeft="@+id/ident"`
- `app:layout_marginTop="@+id/ident"`
- `app:layout_marginRight="@+id/ident"`
- `app:layout_marginBottom="@+id/ident"`

Marges si l'autre élémént est absent (GONE) :

- `app:layout_goneMarginStart="@+id/ident"`
- `app:layout_goneMarginEnd="@+id/ident"`
- `app:layout_goneMarginLeft="@+id/ident"`
- `app:layout_goneMarginTop="@+id/ident"`
- `app:layout_goneMarginRight="@+id/ident"`
- `app:layout_goneMarginBottom="@+id/ident"`

Méthodes de la classe ConstraintLayout

android.support.constraint.ConstraintLayout

Construction

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

Dimensions

- `getMaxHeight()`
- `getMaxWidth()`
- `getMinHeight()`
- `getMinWidth()`

Evénements

Mêmes méthodes que la classe ViewGroup

Les groupes

Les groupes permettent de regrouper des éléments en liste, tableaux, etc. Ils héritent tous de `ViewGroup`. Leurs propriétés sont donc au moins celles décrites ci-dessus pour `View` et `ViewGroup` auxquelles viennent s'ajouter des propriétés spécifiques décrites dans cette partie.
Le groupe `RadioGroup` ne permet que de regrouper des boutons radio et de faire en sorte qu'un seul soit sélectionné à la fois.
Les groupes `Listiez` et `GridView` permettent d'organiser des éléments quelconques.
Généralement `ListView` et `GridView` sont utilisés pour des listes ou des tableaux de chaînes de caractères. Toutefois ces groupes peuvent gérer des vues quelconques (classe `View` ou dérivée) dans ce cas il faut écrire un gestionnaire de contenu (`Adapter`) et le leur associer.

RadioGroup

Réservé au groupage de boutons radio (classe `RadioButton`). Le `RadioGroup` garantit qu'un seul bouton soit coché à la fois toutefois il ne fait aucune hypothèse sur l'état initial des boutons. Ceci signifie que si dans le fichier XML vous avez activé l'un des boutons du groupe il le restera même lorsque vous en activerez un autre. Il est donc préférable de ne mettre dans le fichier XML que des boutons non cochés puis d'initialiser le RadioGroup par sa méthode `check` (voir plus loin).

Propriétés supplémentaires positionnables en XLM

Mêmes paramètres XML que `LinearLayout`

Exemple de fichier XML :

```
<RadioGroup android:id="@+id/groupe"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:orientation="vertical">
      <RadioButton
      …
      />
      …
```

```
</RadioGroup>
```

Méthodes de la classe RadioGroup
android.widget.RadioGroup

Construction

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

Selection

- `check(int)` active le bouton radio dont l'identifiant est passé en paramètre
- `clearCheck()` désactive tous les boutons radio
- `getCheckedRadioButtonId()` renvoie l'identifiant du bouton radio actif

Evénements

Mêmes méthodes que la classe ViewGroup plus :

- `setOnCheckedChangeListener(RadioGroup.OnCheckedChangeListener)` associe un écouteur aux changements de sélection. La méthode `onCheckedChanged(RadioGroup, int)` de l'interface `RadioGroup.OnCheckedChangeListener` doit être surgargée pour traiter les événements. Le premier paramètre est la `RadioGroup` concerné, le second indique le rang du bouton radio activé.

ListView

`ListView` place les éléments en liste avec un ascenseur vertical si nécessaire. `ListView` est normalemnt 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 utisant un gestionnaire de contenu.

Propriétés supplémentaires positionnables en XLM

Contenu de type texte

- `android:entries="@array/maliste"` définit le contenu de la liste à partir d'une ressource de type tableau de chaînes de caractères définie dans un fichier xml placé dans **res/values/** sous la forme suivante :

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

Séparateurs

- `android:divider="couleur"` définit la couleur des séparateurs ou `android:divider="@drawable/monimage"` pour utiliser une image. L'image est placée dans `res/drawable/` et s'appelle monimage.x (où x est `png`, `jpg`, ou `gif`).
- `android:dividerHeight="unité"` définit la hauteur des séparateurs (si c'est une image elle sera déformée).

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

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)`. La classe `ArrayAdapter` possède les méthodes suivantes :

Construction

- `ArrayAdapter(Context, type)` le premier paramètre est le contexte de l'activité elle-même, le second paramètre peut être une valeur prédéfinie :
  `android.R.layout.simple_list_item_1` pour une liste à choix unique ou `android.R.layout.simple_list_item_multiple_choice` pour une liste à choix multiple (dans ce cas une case à cocher apparaît à coté de chaque élément de la liste). Le second paramètre peut également être l'identificateur d'un widget `TextView` personnalisé défini dans un fichier xml placé dans **res/layout** et désigné par `R.layout.`*nom_du_fichier_xml*

Eléments

- `add(Object)` pour ajouter à la fin
- `insert(Object, int)` pour insérer un élément au rang donné en 2ème paramètre
- `clear()` pour enlever tous les éléments
- `remove(Object)` pour enlever un élément
- `getCount()` renvoie le nombre d'éléments
- `getItem(int)` renvoie l'élément (Object) dont le rang est donné en 2ème paramètre
- `getPosition(Object)` renvoie le rang de l'objet désigné en paramètre

Remarque : Pour gérer un contenu autre que du texte il existe un gestionnaire (classe `SimpleAdapter`) permettant que chaque élément de la liste soit une petite interface définie dans un fichier xml. On peut également écrire son propre gestionnaire de contenu en définissant une classe héritant de `BaseAdapter` (voir exemple fourni pour la classe `Gallery` plus loin).

Aspect

- `setDivider(Drawable)` définit l'image qui sera dessinée entre chaque élément de la liste
- `getDivider()` renvoie 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 la 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

- `setItemChecked(int, boolean)` sélectionne (true) ou désélectionne (false) un élément de liste par son rang (à partir de 0).
- `isItemChecked(int)` renvoie un booléen qui indique si l'élément de liste désigné par son rang (à partir de 0) est selectionné.
- `clearChoices()` invalide les choix actuels
- `getChoiceMode()` renvoie le mode de choix qui peut prendre les valeurs : `none`, `singleChoice` ou `multipleChoice`

- `setChoiceMode(int)` définit le mode de choix, le paramètre peut prendre les valeurs : `none`, `singleChoice` ou `multipleChoice`
- `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()` renvoie un booléen qui 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

Mêmes méthodes que la classe ViewGroup plus :

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

GridView

Fonctionne comme un `ListView` mais place les éléments en tableau avec ascenseur vertical seulement si nécessaire.

Propriétés supplémentaires positionnables en XLM

- `android:gravity="g"` (où g peut prendre les valeurs : `top`, `bottom`, `left`, `right`, center_vertical, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`) définit comment se placent les éléments contenus par ce conteneur.

Couleurs

- `android:listSelector="couleur"` pour définir une couleur pour désigner l'objet sélectionné

Sélection

- `android:choiceMode="m"` (où m peut prendre les valeurs : `none`, `singleChoice` ou `multipleChoice` définit le type de sélection
- `android:textFilterEnabled="b"` (où b vaut true ou false) mise en place d'un filtrage des éléments au fur et à mesure de la saisie
- `android:drawSelectorOnTop="b"` (où b vaut true ou false) true indique que le selecteur est dessiné sur l'élément choisi (par défaut c'est false)
- `android:fastScrollEnabled="b"` (où b vaut true ou false) autorise ou pas le parcours rapide

Dimensions et espacement des colonnes

- `android:numColumns="x"` (où est une valeur entière) définit le nombre de colonnes

- `android:columnWidth="unité"`
- `android:horizontalSpacing="unité"`
- `android:verticalSpacing="unité"`
- `android:stretchMode="s"` (où s peut prendre les valeurs : `none`, `spacingWidth`, `columnWidth`, `spacingWidthUniform`) définit comment les colonnes occupent la place disponible : `none`=non occupée, `spacingWidth`=étirement de l'espace entre les colonnes, `columnWidth`=étirement des colonnes, `spacingWidthUniform`=étirement uniforme de toutes les colonnes.

Méthodes de la classe GridView
android.widget.GridView

Construction

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

Contenu

Pour remplir un `GridView` on utilise les mêmes mécanismes que ceux décrits plus haut pour la classe `ListView`.

Position et disposition

- `setGravity(int)` Le paramètre est l'une des constantes définies dans la classe Gravity on utilise donc Gravity. (`TOP`, `BOTTOM`, `LEFT`, `RIGHT`, `CENTER_VERTICAL`, `FILL_VERTICAL`, `CENTER_HORIZONTAL`, `FILL_ HORIZONTAL`, `CENTER`, `FILL`)
- `setStretchMode(int)` définit comment les colonnes occupent la place disponible le paramètre peut prendre les valeurs : `none`, `spacingWidth`, `columnWidth`, `spacingWidthUniform`) `none`=non occupée, `spacingWidth`=étirement de l'espace entre les colonnes, `columnWidth`=étirement des colonnes, `spacingWidthUniform`=étirement uniforme de toutes les colonnes
- `getStretchMode()` renvoie l'information de placement des colonnes (voir `setStretchMode` pour la valeur renvoyée).
- `setColumnWidth(int)` définit en pixels la largeur des colonnes
- `setHorizontalSpacing(int)` définit en pixels l'espacement entre les colonnes
- `setVerticalSpacing(int)` définit en pixels l'espacement entre les lignes
- `setNumColumns(int numColumns)` définit le nombre de colonnes
- `invalidateViews()` provoque le rafraichissement des tous le éléments internes

Sélection

- `setSelection(int)` sélectionne un éléments de liste par son rang (à partir de 0).
- `clearChoices()` invalide les choix actuels
- `getChoiceMode()` renvoie le mode de choix peut prendre les valeurs : `none`, `singleChoice` ou `multipleChoice`
- `setChoiceMode(int)` définit le mode de choix, le paramètre peut prendre les valeurs : `none`, `singleChoice` ou `multipleChoice`
- `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()` renvoie un booléen qui 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

Mêmes méthodes que la classe ListView

Les composants d'interface

Ce sont les composants finals d'interface qui permettent l'interaction de l'utilisateur.

A) Leur position est définie :

- Lorsqu'ils sont placés dans un `RelativeLayout` ou un `ConstraintLayout` par rapport à ce contenant et/ou aux autres composants (voir `RelativeLayout` et `ConstraintLayout`)
- Lorsqu'ils sont placés dans un autre contenant par ce contenant (voir ci-dessous)

B) Leur taille est définie par :

- `android:layout_height="t"` (où t peut être une unité ou prendre les valeurs : `MATCH_PARENT` ou `WRAP_CONTENT`) pour occuper tout l'espace en hauteur ou seulement ce qui est nécessaire
- `android:layout_width="t"` (où t peut être une unité prendre les valeurs : `MATCH_PARENT` ou `WRAP_CONTENT`) pour occuper tout l'espace en largeur ou seulement ce qui est nécessaire

C) Sauf pour `RelativeLayout` et `ConstraintLayout` leur occupation du conteneur est définie par :

- `android:layout_gravity="g"` (où g peut prendre les valeurs : `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`). Il est possible de composer ces valeurs par | (par exemple : `top|right`)

Remarque : le fonctionnement de `layout_gravity` est plutôt étrange dans un `LinearLayout`. En effet ce contenant place les éléments les uns à coté des autres ou les uns sous les autres selon son orientation, et `layout_gravity` n'a d'effet que sur l'autre direction c'est-à-dire que pour un `LinearLayout` horizontal seules les valeurs `top`, `center` et `bottom` ont un sens alors que pour un `LinearLayout` vertical seules les valeurs `left`, `center` et `right` ont un sens. Si on veut définir un placement de certains éléments dans le sens du `LinearLayout` lui-même il faut les inclure dans un `FrameLayout` et les placer dans ce `FrameLayout`.

Exemple :
La description ci-dessous ne placera pas, comme on pourait s'y attendre, la case à cocher à droite de l'écran


```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
      <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_vertical"
      android:text="Essai placement"
      />
      <CheckBox
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="top|right"
      />
</LinearLayout>
```

Pour obtenir le résultat souhaité il faut encapsuler la case à cocher dans un FrameLayout :


```
<FrameLayout
android:layout_width="fill_parent"
android:layout_height="wrap_content"
>
      <CheckBox
      android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_gravity="top|right"
        />
    </FrameLayout>
```

D) Pour un `TableLayout` le nombre de colonnes occupéespar l'élément est défini par :

- `android:layout_span="s"` (où s indique le nombre de colonnes (par défaut 1))

E) Les marges externes sont définies par :

- `android:layout_marginBottom="unité"` marge externe en bas
- `android:layout_marginLeft="unité"` marge externe à gauche
- `android:layout_marginRight="unité"` marge externe à droite
- `android:layout_marginTop="unité"` marge externe en haut

ImageView

Il s'agit simplement d'une zone dans laquelle s'affiche une image.

Propriétés supplémentaires positionnables en XLM :

Contenu

- `android:src="couleur"` pour définir une couleur ou `="@drawable/monimage"` pour définir une image. L'image est placée dans `res/drawable/` et s'appelle monimage.x (où x est `png`, `jpg`, ou `gif`). Méthodes correspondantes : `setImageBitmap`, `setImageDrawable`, `setImageResource`

Dimensions et teinte

- `android:tint="couleur"` pour définir une couleur qui teinte l'image
- `android:adjustViewBounds="b"` (où b vaut true ou false) false indique que la taille de la zone d'affichage correspond à celle de l'image

Dans le cas où ce paramètre est à true on précise le comportement de l'image

- `android:baselineAlignBottom="b"` (où b vaut true ou false) indique si l'image est placée en bas de la zone
- `android:cropToPadding="b"` (où b vaut true ou false) true indique que l'image sera coupée si elle est plus grande que la taille de la zone d'affichage
- `android:scaleType="s"` (où s peut prendre les valeurs :
    - `matrix` l'image n'est ni centrée ni redimensionnée
    - `center` l'image est centrée mais non redimensionnée
    - `fitXY` l'image est déformée pour occuper tout l'espace
    - `fitStart` l'image est cadrée en haut à gauche et redimensionnée
    - `fitCenter` l'image est centrée et redimensionnée
    - `fitEnd` l'image est cadrée en bas à droite et redimensionnée
    - `centerCrop` l'image est centrée et redimensionnée selon sa dimension minimale et non déformée, l'autre dimension pourra donc être coupée
    - `centerInside` l'image est centrée et redimensionnée selon sa dimension maximale et non déformée; elle ne sera donc pas coupée

Exemple de fichier XML :

```
<ImageView android:id="@+id/image"
  android:src="@drawable/photo"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:adjustViewBounds="true" />
```

Méthodes de la classe ImageView

android.widget.ImageView

Construction

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

Aspect

- `getDrawable()` renvoie l'image de la zone (objet de classe `Drawable`)
- `setAlpha(int)` définit la transparence (le paramètre est un réel entre 0 et 1)
- `setImageBitmap(Bitmap)` définit l'image (objet de classe `Bitmap`)
- `setImageDrawable(Drawable)` définit l'image (objet de classe `Drawable`)
- `setImageResource(int)` définit l'image par son identificateur
- `setMaxHeight(int)` définit la hauteur maximale en pixels
- `setMaxWidth(int)` définit la largeur maximale en pixels
- `setScaleType(int)` définit le positionnement de l'image. les valeurs possibles du paramètre sont : `ImageView.ScaleType.MATRIX`, `CENTER`, `FIT_XY`, `FIT_START`, `FIT_CENTER`, `FIT_END`, `CENTER_CROP` ou `CENTER_INSIDE`. Après avoir modifié le positionnement si on n'a pas changé l'image il faut appeler la méthode `invalidate()` de l'`ImageView` pour que ce soit pris en compte.

Evénements

Mêmes méthodes que la classe View

TextView

Affichage de texte a priori non éditable (pour du texte éditable utiliser `EditView`).

Propriétés supplémentaires positionnables en XLM

Contenu

- `android:text="texte à afficher"`
- `android:hint="initial"` définit le texte à afficher quand la zone est vide

Taille et aspect du texte

- `android:gravity="g"` (où g peut prendre les valeurs : `top`, `bottom`, `left`, `right`, center_vertical, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`) définit comment se place le texte
- `android:textSize="unité"` utiliser de préférence l'unité **sp** qui est liée aux polices
- `android:textScaleX="x"` (où est une valeur réelle) définit l'échelle horizontale du texte
- `android:texteStyle="g"` (où s peut être `normal`, `bold`, `italic`) ces styles peuvent être combinés par |
- `android:typeface="s"` (où s peut être `normal`, `sans`, `serif`, `monospace`)
- `android:singleLine="b"` (où b vaut true ou false) limite le texte à une seule ligne
- `android:lines="x"` (où est une valeur entière) définit le nombre de lignes du texte
- `android:maxLines="x"` (où est une valeur entière) définit le nombre maximal de lignes du texte
- `android:minLines="x"` (où est une valeur entière) définit le nombre minimal de lignes du texte
- `android:lineSpacingExtra="unité"` espace supplémentaire entre les lignes
- `android:scrollHorizontally="b"` (où b vaut true ou false) autorise ou pas le défilement horizontal du texte

Dimensions

- `android:ems="x"` (où est une valeur entière) définit la taille du texte en caractères
- `android:maxEms="x"` (où est une valeur entière) définit le nombre maximal de caractères des lignes du texte
- `android:height="unité"`
- `android:maxHeight="unité"`

- android:minHeight="unité"
- android:width="unité"
- android:maxWidth="unité"
- android:minWidth="unité"

Comportement

- android:autoLink="a" (où a peut être : none, web, email, phone, map ou all) indique si les liens de ce type apparaissant dans le texte sont automatiquement rendus cliquables.
- android:autoText="b" (où b vaut true ou false) valide ou pas le mode correction du texte
- android:capitalize="c" (où c peut être : none, sentences, words, caracters) indique le type de saisies que le texte mémorise et peut re-proposer.
- android:digits="b" (où b vaut true ou false) true indique que la saisie n'accepte que du numérique
- android:numerics="x" (où x peut être integer, signed, decimal) définit le mode de saisie numérique
- android:password="b" (où b vaut true ou false) si true lors de la saisie des points sont affichés
- android:phonenumber="b" (où b vaut true ou false) true indique que la saisie n'accepte que des numéros de téléphone
- android:inputType="t" (où t peut être : none, text, textCapCharacters, textCapWords, textCapSentences, textAutoCorrect, textAutoComplete, textMultiLine, textUri, textEmailAddress, textEmailSubject, textShortMessage, textLongMessage, textPersonName, textPostalAddress, textPassword, textVisiblePassword, textWebEditText, textFilter, textPhonetic, textWebEmailAddress, textWebPassword, number, numberDecimal, numberPassword, phone, datetime, date ou time) définit le mode de saisie.

Affichage

- android:cursorVisible="b" (où b vaut true ou false) rend visible ou non le curseur
- android:editable="b" (où b vaut true ou false) autorise ou pas la modification du texte
- android:ellipsize="e" (où e peut être : none, start, middle, end, marquee) définit le mode de césure du texte
- android:linksClickable="b" (où b vaut true ou false) rend ou pas les liens cliquables
- android:textIsSelectable="b" (où b vaut true ou false) autorise ou interdit la sélection dans le texte

Couleurs et images

- android:textColor="couleur" pour définir la couleur du texte
- android:textColorHighlight="couleur" pour définir la couleur de surlignage du texte
- android:textColorHint="couleur" pour définir la couleur du texte par défaut
- android:textColorLink="couleur" pour définir la couleur des liens dans le texte
- android:drawableBottom="couleur" pour définir la couleur de fond du texte ou ="@drawable/monimage" pour définir une image de fond au texte. L'image est placée dans res/drawable/ et s'appelle monimage.x (où x est png, jpg, ou gif)
- android:shadowColor="couleur" pour définir la couleur d'ombrage du texte
- android:shadowDx="x" (où est une valeur réelle) définit le décalage en X de l'ombre (voir shadowRadius)
- android:shadowDy="x" (où est une valeur réelle) définit le décalage en Y de l'ombre (voir shadowRadius)
- android:shadowRadius="x" (où est une valeur réelle) définit l'épaisseur de l'ombre (ATTENTION : par défaut c'est 0 donc il n'y a pas d'ombre)

Exemple de fichier XML :

```
<TextView android:id="@+id/untexte"
 android:text="Contenu du texte"
 android:layout_width="wrap_content"
```

```
 android:layout_height="wrap_content" />
```

Méthodes de la classe TextView

android.widget.TextView

Construction

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

Manipulation du texte

- `getText()` renvoie le texte contenu dans le TextView sous forme d'un objet de classe `Editable` que l'on peut transformer en `String` par sa méthode `toString`() càd par : `getText`().`toString`()
- `setText(CharSequence)` définit le texte du `TextView`
- `append(CharSequence)` ajoute du texte au `TextView`
- `getLineCount()` renvoie le nombre de lignes du texte contenu dans le `TextView`
- `length()` renvoie la longueur du texte contenu dans le `TextView`
- `getSelectionStart()` renvoie la position de début du texte sélectionné
- `getSelectionEnd()` renvoie la position de fin du texte sélectionné
- `hasSelection()` indique s'il y a ou pas du texte sélectionné (true si oui, false si non)

Aspect

- `setCursorVisible(boolean)` rend le curseur visible (true) ou invisible (false)
- `setEnabled(boolean)` rend le `TextView` actif ou pas
- `getCurrentTextColor()` renvoie la couleur actuelle du texte
- `setTextColor(int)` définit la couleur du texte
- `setHighlightColor(int)` définit la couleur du texte surligné
- `setHint(CharSequence)` définit le texte à afficher quand la zone est vide

Evénements

Mêmes méthodes que la classe View plus :

- `addTextChangedListener(TextWatcher)` pour associer un écouteur d'événements aux modifications du texte.
- `setOnEditorActionListener(TextView.OnEditorActionListener)` pour associer un écouteur d'événements aux actions sur le texte.
- `setKeyListener(KeyListener)` pour associer un écouteur d'événements au clavier. L'interface `KeyListener` possède les méthodes à surcharger suivantes :
  - `onKeyDown(View, Editable, int, KeyEvent)` appelée lorsqu'une touche est appuyée
  - `onKeyUp(View, Editable, int, KeyEvent)` appelée lorsqu'une touche est lâchée

  Pour ces 2 méthodes les paramètres sont :
  - Le `TextView` lui-même
  - Le texte
  - Le code de la touche (voir `OnKeyListener` plus loin)
  - L'événement clavier (voir classe `KeyEvent` plus loin)

EditText

C'est la même chose que `TextView` mais pour du texte éditable

Exemple de fichier XML :

```
<EditText android:id="@+id/texte"
 android:text="contenu"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:typeface="serif"
 android:phoneNumber="true"
```

/>

Méthodes de la classe EditText
android.widget.EditText

Construction

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

Manipulation du texte

- `getText()` renvoie le texte contenu dans le EditText
- `selectAll()` sélectionne tout le texte
- `setSelection(int start, int stop)` sélectionne le texte entre les positions passées en paramètre (début, fin).

Evénements

Mêmes méthodes que la classe TextView

AutoCompleteTextView

Texte éditable proposant de la complétion. Possède les mêmes paramètres XML que `EditText` plus :

- `android:completionHint="texte"` texte affiché en titre du menu déroulant
- `android:completionThreshold="x"` (où x est une valeur entière) définit le nombre de caractères à taper avant que la complétion n'entre en action.
- `android:dropDownHeight="unité"` on peut aussi utiliser les constantes `fill_parent` et `wrap_content`, définit la hauteur du menu déroulant
- `android:dropDownWidth="unité"` on peut aussi utiliser les constantes `fill_parent` et `wrap_content`, définit la largeur du menu déroulant
- `android:dropDownHorizontalOffset="unité"` décalage horizontal du menu déroulant
- `android:dropDownVerticalOffset="unité"` décalage vertical du menu déroulant

Méthodes de la classe AutoCompleteTextView
android.widget.AutoCompleteTextView

Construction

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

Aspect et contenu

- `showDropDown()` montre la liste déroulante
- `dismissDropDown()` cache la liste déroulante
- `getListSelection()` renvoie le rang de la proposition choisie
- `setListSelection(int)` choisit une proposition par son rang
- `setAdapter(ArrayAdapter)` Permet de remplir la liste de propositions. La classe `ArrayAdapter` est décrite dans `ListView`.
- `setCompletionHint(CharSequence)` définit le texte affiché en titre du menu déroulant

Evénements

Mêmes méthodes que la classe EditText plus :

- `setOnItemClickListener(AdapterView.OnItemClickListener)` associe un écouteur d'événements au clic sur un élément proposé. La méthode `onItemClick(AdapterView<?>,`

`View, int, long)` de l'interface `AdapterView.OnItemLongClickListener` est surchargée pour traiter l'événement. Le 3ème paramètre indique le rang de l'élément et le dernier son identifiant.

- `setOnItemSelectedListener(AdapterView.OnItemSelectedListener)` associe un écouteur d'événements à la sélection d'un élément proposé. La méthode `onItemSelected (AdapterView<?>, View, int, long)` de l'interface `AdapterView.OnItemSelectListener` est surchargée pour traiter l'événement de sélection d'un élément. Le 3ème paramètre indique le rang de l'élément et le dernier son identifiant.
  La méthode `onNothingSelected(AdapterView<?>)` est surchargée pour traiter l'événement de non sélection.

MultiAutoCompleteTextView

Même comportement que **AutoCompleteTextView** pour des textes multilignes.

Button

Mêmes paramètres XML que TextView

Exemple de fichier XML :

```
<Button android:id="@+id/compte"
 android:text="Count"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent" />
```

Méthodes de la classe Button

android.widget.Button

Construction

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

Evénements

Mêmes méthodes que la classe TextView

ImageButton

Mêmes paramètres XML que `ImageView`

Exemple de fichier XML :

```
<ImageButton android:id="@+id/boutonloupe"
  android:src="@drawable/loupe"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content" />
```

Méthodes de la classe ImageButton

android.widget.ImageButton

`ImageButton` hérite de `ImageView` il a donc les mêmes méthodes :

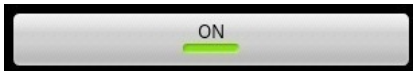
Construction

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

Evénements

Mêmes méthodes que la classe View

ToggleButton



Mêmes paramètres XML que `TextView` plus :

- `android:disabledAlpha="x"` (où est une valeur réelle entre 0 et 1) définit la transparence appliquée lorsque le bouton est inactif
- `android:textOff="txt"` définit le texte quand le bouton n'est pas allumé
- `android:textOn="txt"` définit le texte quand le bouton n'est pas allumé
- `android:checked="b"` (où b vaut true ou false) place le bouton en position allumé ou éteint

Méthodes de la classe ToggleButton
android.widget.ToggleButton

Construction
- `ToggleButton(Context)` le paramètre est généralement l'activité elle-même

Etat
- `isChecked()` indique si le bouton est allumé ou éteint (true/false)
- `setChecked(boolean)` positionne le bouton (true = allumé, false = éteint)
- `toggle()` change l'état du bouton

Aspect
- `getTextOff()` renvoie le texte associé au bouton quand il n'est pas allumé
- `getTextOn()` renvoie le texte associé au bouton quand il est allumé
- `setTextOff(CharSequence)` définit le texte associé au bouton quand il n'est pas allumé
- `setTextOn(CharSequence)` définit le texte associé au bouton quand il est allumé
- `setBackgroundDrawable(Drawable)` définit le fond du bouton

Evénements
Mêmes méthodes que la classe View plus :
- `setOnCheckedChangeListener(RadioGroup.OnCheckedChangeListener)` associe un écouteur aux changements de sélection. La méthode `onCheckedChanged(RadioGroup, int)` de l'interface `RadioGroup.OnCheckedChangeListener` doit être surgargée pour traiter les événements. Le premier paramètre est la `RadioGroup` concerné, le second indique le rang du bouton radio activé.

Switch
Même comportement que **ToggleButton** seule l'apparence change.

CheckBox

Mêmes paramètres XML que `TextView` plus :
- `android:checked="b"` où b vaut true ou false coche ou décoche la case au départ

Exemple de fichier XML :
```
<CheckBox android:id="@+id/case"
 android:text="Checkbox"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:checked="true" />
```

Méthodes de la classe CheckBox
android.widget.CheckBox

Construction
- `Checkbox(Context)` le paramètre est généralement l'activité elle-même

Etat

- `isChecked()` renvoie true si la case est cochée
- `setChecked(boolean)` coche (true) ou décoche (false) la case
- `toggle()` change l'état de la case

Evénements

Mêmes méthodes que la classe ToggleButton

RadioButton

Mêmes paramètres XML que `TextView` plus :

- `android:checked="b"` où b vaut true ou false coche ou décoche le bouton au départ

Exemple de fichier XML :

```
<RadioButton android:id="@+id/gauche"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Left"
android:checked="false"
android:layout_gravity="left"/>
```

Méthodes de la classe RadioButton

android.widget.RadioButton

Construction

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

Etat

- `isChecked()` renvoie true si la case est cochée
- `setChecked(boolean)` coche (true) ou décoche (false) la case
- `toggle()` change l'état de la case

Evénements

Mêmes méthodes que la classe ToggleButton

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`.

Propriétés supplémentaires positionnables en XLM

- `android:gravity="g"` (où g peut prendre les valeurs : `top`, `bottom`, `left`, `right`, center_vertical, `fill_vertical`, `center_horizontal`, `fill_ horizontal`, `center`, `fill`) définit comment se placent les éléments de choix.
- `android:prompt="texte"` 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 d'une ressource de type tableau de chaînes de caractères mise dans un fichier xml placé dans res/values/ sous la forme :

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

```
    </string-array>
</resources>
```

Méthodes de la classe Spinner
android.widget.Spinner

Construction

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

Contenu

- `setAdapter(ArrayAdapter)` Permet de remplir la liste de choix. La façon de procéder est décrite dans `ListView`.

Sélection

- `getCount()` renvoie le nombre de choix
- `setSelection(int)` sélectionne un élément de la liste par son rang (à partir de 0)

Evénements

Mêmes méthodes que la classe ListView

- `setOnItemSelectedListener(AdapterView.OnItemSelectedListener)` associe un écouteur d'événements à la sélection d'une choix.
  La méthode `onItemSelected (AdapterView<?>, View, int, long)` de l'interface `AdapterView.OnItemSelectListener` est surchargée pour traiter l'événement. Le 3ème paramètre indique le rang de l'élément et le dernier son identifiant.
  La méthode `onNothingSelected(AdapterView<?>)` est surchargée pour traiter l'événement de non sélection.

DatePicker

Mêmes paramètres XML que `FrameLayout` plus :

- `android:startYear="x"` où x est l'année de départ du calendrier affiché
- `android:endYear="x"` où x est l'année de fin du calendrier affiché
- `android:minDate="x"` où x est la date de départ du calendrier affiché sous la forme mm/jj/aaaa
- `android:maxDate="x"` où x est la date de fin du calendrier affiché sous la forme mm/jj/aaaa

Exemple de fichier XML :

```
<DatePicker android:id="@+id/date"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:startYear="2000"
 android:endYear="2020" />
```

Méthodes de la classe DatePicker
android.widget.DatePicker

Construction

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

Contenu

- `getDayOfMonth()` renvoie le jour affiché (1 à 31)
- `getMonth()` renvoie le mois affiché (1 à 12)
- `getYear()` renvoie l'année affichée
- `updateDate(int, int, int)` met à jour la date affichée, les paramètres sont dans l'ordre : année, mois, jour)

- `isEnabled()` indique si le DatePicker est actif ou pas
- `setEnabled(boolean)` rend le DatePicker actif ou pas

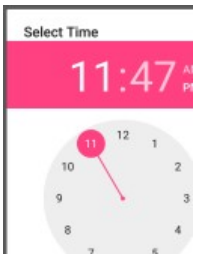
Initialisation et prise en compte des événements

- `init(int, int, int, DatePicker.OnDateChangedListener)` initialise l'année (1er paramètre), le mois (2ème paramètre), le jour (3ème paramètre) et définit un écouteur d'événements lorsque une date est choisie (dernier paramètre). La méthode `onDateChanged(DatePicker, int, int, int)` de l'interface `DatePicker.OnDateChangedListener` doit être surchargée pour traiter l'événement. Ses paramètres sont : le `DatePicker` lui-même, l'année choisie, le mois choisi et le jour choisi.

Evénements

Mêmes méthodes que la classe ViewGroup plus l'association d'écouteur faite par init (voir ci-dessus)

TimePicker



Mêmes paramètres XML que `FrameLayout`

Exemple de fichier XML :

```
<TimePicker android:id="@+id/temps"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />
```

Méthodes de la classe TimePicker

android.widget.TimePicker

Construction

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

Contenu

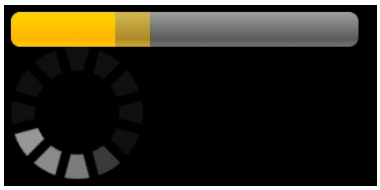
- `getCurrentHour()` renvoie l'heure affichée
- `getCurrentMinute()` renvoie les minutes affichées
- `setCurrentHour(Integer)` définit l'heure affichée
- `setCurrentMinute()` définit les minutes affichées
- `is24HourView()` indique si l'affichage est fait sur 12 ou 24 heures
- `setIs24HourView(Boolean)` définit le mode d'affichage (true ⇒ en 24 heures, false ⇒ en 12 heures)
- `isEnabled()` indique si le `TimePicker` est actif ou pas
- `setEnabled(boolean)` rend le `TimePicker` actif ou pas

Evénements

Mêmes méthodes que la classe ViewGroup plus :

- `setOnTimeChangedListener(TimePicker.OnTimeChangedListener)` définit un écouteur d'événements lorsqu'une heure est choisie.

ProgressBar



Deux comportements possibles selon que l'on connaît le taux de progression ou pas. Dans le premier cas on peut faire évoluer la progression, dans le second on a une animation.
Deux formes sont possibles (barre verticale ou cercle).

Propriétés supplémentaires positionnables en XLM

Mode

- `android:indeterminate="b"` (où b vaut true ou false) définit le type de `ProgressBar` (true = taux indéterminé, false= taux déterminé).

Valeurs si on a choisi le mode déterminé

- `android:max="x"` (où est une valeur entière) définit la valeur maximale
- `android:progress="x"` (où est une valeur entière) définit la valeur initiale
- `android:secondaryProgress="x"` (où est une valeur entière) définit une valeur secondaire (par exemple celle d'un buffer comme on le voit sur des vidéos en streaming)

Animation si on a choisi le mode indéterminé

- `android:indeterminateBehavior="i"` (où i peut être : `repeat` ou `cycle`) définit le comportement de l'animation pour le type indéterminé (`repeat`=recommence l'animation, `cycle`=change le sens de l'animation)
- `android:indeterminateDuration="x"` (où est une valeur entière) définit la durée de l'animation (en ms)
- `android:animationResolution="x"` (où est une valeur entière) définit le rythme de l'animation (en ms)

Aspect

- `style="?android:attr/s"` où s peut être :
  - `progressBarStyleHorizontal` une barre horizontale (cas déterminé)
  - `progressBarStyleSmall` un cercle de petite taille (cas indéterminé)
  - `progressBarStyleLarge` un cercle de grande taille (cas indéterminé)

Exemple de fichier XML :

```
<ProgressBar android:id="@+id/progres"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:minWidth="300px"
  android:minHeight="30px"
  android:max="100"
  android:progress="30"
  android:secondaryProgress="40"
  android:indeterminate="false"
  style="?android:attr/progressBarStyleHorizontal" />
```

Méthodes de la classe ProgessBar

android.widget.ProgressBar

Construction

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

Valeurs

- `getMax()` renvoie la valeur maximale
- `setMax(int)` définit valeur maximale
- `getProgress()` renvoie la valeur actuelle
- `setProgress(int)` définit la valeur actuelle
- `getSecondaryProgress()` renvoie la valeur secondaire actuelle
- `setSecondaryProgress(int)` définit la valeur secondaire actuelle
- `incrementProgressBy(int)` modifie le pas de progression en l'augmentant de la valeur du paramètre

- `incrementSecondaryProgressBy(int)` modifie le pas de progression de la valeurs secondaire en l'augmentant de la valeur du paramètre

Aspect

- `isIndeterminate()` indique si le `ProgressBar` est en mode indéterminé (true) ou pas
- `setIndeterminate(boolean)` met le `ProgressBar` est en mode indéterminé (true) ou pas
- `setVisibility(int)` définit la visibilité. Le paramètre peut être : `VISIBLE`, `INVISIBLE`, ou `GONE`.

Evénements

Mêmes méthodes que la classe View

SeekBar

C'est un `ProgressBar` sous forme de barre horizontale doté d'un curseur permettant d'en modifier la valeur.
Mêmes paramètres XML que `ProgressBar` mais l'aspect ne peut pas être modifié.

Méthodes de la classe SeekBar

android.widget.SeekBar

Construction

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

On trouve les mêmes méthodes que dans la classe `ProgressBar` puisque `SeekBar` en hérite mais on peut gérer les événements de modification de la valeur :

Evénements

Mêmes méthodes que la classe View plus :

- `setOnSeekBarChangeListener(SeekBar.OnSeekBarChangeListener)` permet d'associer un écouteur d'événements aux modifications de la valeur.

RatingBar

Mêmes paramètres XML que ProgressBar plus :

- `android:isIndicator="b"` (où b vaut true ou false) indique si l'utilisateur peut modifier la valeur ou pas (true= non modifiable)
- `android:numStars="x"` (où est une valeur entière) définit le nombre d'étoiles affichées
- `android:rating="x"` (où est une valeur réelle) définit la position initiale
- `android:stepSize="x"` (où est une valeur réelle) définit le pas de progression (on peut cocher des ½ étoiles par exemple)

Méthodes de la classe RatingBar

android.widget.RatingBar

Construction

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

Valeurs

- `getNumStars()` renvoie le nombre d'étoiles
- `setNumStars(int)` définit le nombre d'étoiles
- `getRating()` renvoie la valeur actuelle (réel)
- `setRating(float)` définit la valeur actuelle

- `getStepSize()` renvoie le pas de progression (réel)
- `setStepSize(float)` définit le pas de progression
- `isIndicator()` indique si l'utilisateur peut modifier la valeur ou pas (true= non modifiable)
- `setIsIndicator(boolean)` définit si l'utilisateur peut modifier la valeur ou pas (true= non modifiable)

Evénements

Mêmes méthodes que la classe View plus :

- `setOnRatingBarChangeListener(RatingBar.OnRatingBarChangeListener)` associe un écouteur d'événements aux modifications.

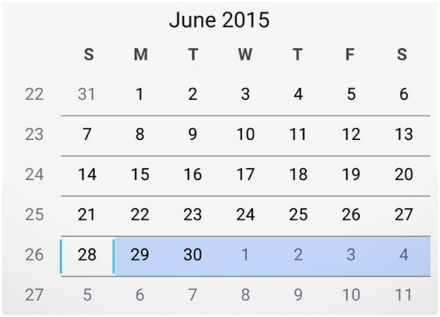
Exemple de fichier XML :

```
<RatingBar android:id="@+id/vote"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:isIndicator="false"
  android:numStars="5"
  android:stepSize="0.1"
  android:rating="3.5" />
```

CalendarView

Exemple de fichier XML :

```
<CalendarView android:id="@+id/calendrier"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
/>
```



Méthodes de la classe CalendarView

android.widget.CalendarView

Construction

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

Valeurs

- `setDate(long date, boolean animate, boolean center)` le premier parameter est la date exprimée en milli secondes depuis le 1er janvier 1970, le deuxième autorise une animation lors de l'affichage, le dernier autorise le centrage du calendrier sur la date même si elle est déjà visible
- `setFirstdayOfweek(int first)` définit le 1er jour de la semaine (les valeurs possibles sont : CalendarView.MONDAY … CalendarView.SuNDAY)
- `setMaxDate(long max)` définit la date maximale affichée
- `setMinDate(long min)` définit la date minimale affichée
- `long getDate()` renvoie la date affichée exprimée en milli secondes depuis le 1er janvier 1970
- `int getFirstdayOfWeek()` renvoie le 1er jour de la semaine
- `long getMaxDate()` renvoie la date maximale affichable
- `long getMinDate()` renvoie la date minimale affichable

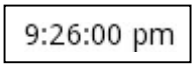
Evénements

Mêmes méthodes que la classe View plus :

- `setOnDateChangeListener(CalendarView.OnDateChangeListener)` associe un écouteur d'événements aux modifications.

TextClock

Mêmes paramètres XML que TextView plus :



```
android:format12Hour=format d'affichage du type : "hh:mm:ss"
android:format24Hour= format d'affichage du type :"hh:mm:ss"
android:timeZone=décalage du type : "GMT -2:25"
```

Exemple de fichier XML :

```
<DigitalClock android:id="@+id/digitalclock"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content" />
```

Méthodes de la classe TextClock

android.widget.TextClock

Construction

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

Evénements

Mêmes méthodes que la classe TextView

Chronometer

C'est un compteur qui démarre à la valeur donnée par `setBase` et compte en incrémentant. Son apparence est celle d'un texte dans lequel apparaît le temps (voir format ci-dessous).

Mêmes paramètres XML que `TextView` plus :

- `android:format="f"` (où f est une chaîne dans laquelle la première occurrence de %s sera remplacée par la valeur du chronomètre sous la forme MM:SS ou H:MM:SS)

Méthodes de la classe Chronometer

android.widget.Chronometer

Construction

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

Utilisation

- `getBase()` renvoie le point de départ du chronomètre (long)
- `setBase(long)` définit le point de départ du chronomètre
- `start()` lance le chronomètre
- `stop()` arrête le chronomètre

Aspect

- `setFormat(String)` le paramètre est une chaîne dans laquelle la première occurrence de %s sera remplacée par la valeur du chronomètre sous la forme MM:SS ou H:MM:SS
- `getFormat()` renvoie le format utilisé (chaîne)

Evénements

Mêmes méthodes que la classe TextView plus :

- `setOnChronometerTickListener(Chronometer.OnChronometerTickListener)` définit un écouteur d'événements pour chaque incrémentation du chronomètre.

Prise en compte des événements

La prise en compte des événements sur les éléments d'interface se fait par association d'un écouteur d'événement grâce à la méthode *setXXX* où XXX désigne la classe d'écouteur à associer.

Il existe quelques exceptions à cette règle :

- Pour les classes contenant du texte (TextView, EditText, AutoCompleteTextView, Button, CheckBox, ToggleButton, RadioButton) l'association d'un écouteur pour les modifications du texte se fait par :
  `addTextChangedListener(TextWatcher)`
  L'association d'un écouteur d'événements pour le clavier se fait par :
  `setKeyListener(KeyListener)`
- Pour un DatePicker elle se fait à l'initialisation par :
  `init(int année, int mois, int jour, OnDateChangedListener écouteur)`

Le tableau suivant indique quels écouteurs peuvent être associés aux différents éléments d'interface :

Classe d'écouteur d'événement	OnClickListener OnLongClickListener OnKeyListener OnTouchListener OnCreateContextMenuListener OnDragListener OnFocusChangeListener	OnTimeChangedListener	OnScrollListener	OnItemClickListener OnItemSelectedListener	OnItemLongClickListener	OnHierarchyChangeListener	Textwatcher OnEditorActionListener	OnCheckedChangeListener	onDateChangeListener	OnChronometerTickListener	OnRatingBarChangeListener	OnSeekBarChangeListener
AutoCompleteTextView	x			x			x					
Button	x						x					
CalendarView									x			
CheckBox	x						x	x				
Chronometer	x						x			x		
DatePicker	x											
EditText	x						x					
GridView	x		x	x	x	x						
ImageButton	x											
HorizontalScrollView	x					x						
ImageView	x											
ListView	x		x	x	x	x						
ProgressBar	x											
RadioButton	x						x	x				
RadioGroup	x					x						
RatingBar	x										x	
SeekBar	x											x
ScrollView	x					x						
Spinner	x			x	x	x						
TextView	x						x					
TimePicker	x	x										
ToggleButton	x						x	x				
View	x											
ViewGroup	x					x						

Le tableau suivant est un récapitulatif de ces écouteurs d'événements avec leurs méthodes et paramètres. Une description détaillée de chacun est donnée par la suite.

classe d'écouteur	événement	méthode à surcharger	paramètres de la méthode
OnClickListener	clic sur la vue	onClick	– la vue elle même
OnLongClickListener	clic long sur la vue	onLongClick	– la vue elle même
OnKeyListener	touche du clavier	onKey	– la vue elle-même – le code de la touche – événement clavier (KeyEvent)
OnTouchListener	touché sur la vue	onTouch	– la vue elle-même – MotionEvent
OnCreateContextMenuListener	ouverture d'un menu contextuel	onCreateContextMenu	– le menu contextuel – la vue elle-même – informations sur l'ouverture du menu
OnDragListener	un objet a été traîné sur la vue	onDrag	– la vue elle-même – événement déplacement (DragEvent)
OnFocusChangeListener	changement de focus de la vue	onFocusChange	– la vue elle-même – booléen indiquant si la vue a ou pas le focus
OnTimeChangedListener	modification de la valeur	onTimeChanged	– le TimePicker lui-même – l'heure – les minutes
OnScrollListener	défilement	onScroll	– la vue elle-même – rang du 1er élément visible – nombre d'éléments visibles – nombre total d'éléments
OnItemClickListener	un élément a été cliqué	onItemClick	– la vue elle-même – l'élément cliqué – la position de l'élément cliqué – l'identificateur de l'élément cliqué
OnItemLongClickListener	un élément a été cliqué (clic long)	onItemLongClick	– la vue elle-même – l'élément cliqué – la position de l'élément cliqué – l'identificateur de l'élément cliqué
OnItemSelectedListener	un élément a été sélectionné	onItemSelected	– la vue elle-même – l'élément cliqué – la position de l'élément cliqué – l'identificateur de l'élément cliqué
OnHierarchyChangeListener	un élément est ajouté à la vue	onClidViewAdded	– la vue elle-même – l'élément ajouté
	un élément est enlevé à la vue	onClidViewRemoved	– la vue elle-même – l'élément enlevé
Textwatcher	le texte a été modifié	afterTextChanged	– le texte modifié
	du texte va être remplacé	beforeTextChanged	– le texte modifié – position de début de la modification – nombre de caractères modifiés – nombre de caractères de remplacement
	du texte a été remplacé	onTextChanged	– le texte modifié – position de début de la modification – nombre de caractères modifiés – nombre de caractères de remplacement
OnEditorActionListener	action sur le texte	onEditorAction	– le texte – l'action – événement clavier (KeyEvent)
OnCheckedChangeListener	l'état a changé	onCheckedChanged	– l'élément lui-même – le nouvel état (booléen)
OnDateChangeListener	La date a changé	onSelecteddayChange	– le calendrier – l'année, le jour, le mois
OnChronometerTickListener	changement de valeur du chronomètre	onChronometerTick	– le chronomètre lui même
OnRatingBarChangeListener	changement de valeur	on RatingChanged	– le RatingBar lui-même – la nouvelle valeur – modifié par l'utilisateur (booléen)
OnSeekBarChangeListener	changement de valeur	onProgressChanged	– Le SeekBar lui-même – la nouvelle valeur – modifié par l'utilisateur (booléen)
	début du déplacement	onStartTrackingTouch	– Le SeekBar lui même
	fin du déplacement	onStopTrackingTouch	– Le SeekBar lui même

L'interface OnClickListener

import android.view.View.OnClickListener

Définit les écouteurs d'événements de type clic. La méthode à surcharger pour traiter les événements est :

- `onClick(View)`

Le paramètre est la vue sur laquelle a eu lieu le clic.

L'interface OnLongClickListener

import android.view.View.OnLongClickListener

Définit les écouteurs d'événements de type clic long. La méthode à surcharger pour traiter les événements est :

- `onLongClick(View)`

Le paramètre est la vue sur laquelle a eu lieu le clic long.

L'interface OnKeyListener

import android.view.View.OnKeyListener

Définit les écouteurs d'événements de type clavier. La méthode à surcharger pour traiter les événements est :

- `onKey(View, int, KeyEvent)`

Le premier paramètre est la vue ayant le focus lors de la frappe au clavier, le deuxième paramètre est le code de la touche tapée, le troisième est l'événement clavier (voir plus loin).
Le code de la touche tapée peut prendre les valeurs :

- `KEYCODE_0` à `KEYCODE_9` pour les chiffres
- `KEYCODE_A` à `KEYCODE_Z` pour les lettres
- `KEYCODE_ALT_LEFT` , `KEYCODE_ALT_RIGHT` , `KEYCODE_SHIFT_LEFT` et `KEYCODE_SHIFT_RIGHT` pour les touches Alt et Shift
- `KEYCODE_NUM` pour la touche Num
- `KEYCODE_APOSTROPHE` , `KEYCODE_GRAVE` pour l'apostrophe et l'anti apostrophe
- `KEYCODE_SLASH` et `KEYCODE_BACKSLASH` pour le / et l'antislash
- `KEYCODE_AT` pour @
- `KEYCODE_STAR` pour la touche *
- `KEYCODE_TAB` pour la touche Tab
- `KEYCODE_SPACE` pour l'espace
- `KEYCODE_COMMA` , `KEYCODE_PERIOD` , `KEYCODE_SEMICOLON` pour la virgule, le point et le point virgule
- `KEYCODE_BACK` pour la touche de correction (backspace)
- `KEYCODE_CALL et KEYCODE_ENDCALL` pour les touches d'appel et de fin d'appel
- `KEYCODE_CAMERA` , `KEYCODE_FOCUS` pour la touche de déclenchement et de réglage de focus de l'appareil photo
- `KEYCODE_CLEAR` pour la touche Clear
- `KEYCODE_DEL` pour la touche Del
- `KEYCODE_HOME` pour la touche Home
- `KEYCODE_MENU` pour la touche Menu
- `KEYCODE_DPAD_CENTER`, `KEYCODE_DPAD_DOWN`, `KEYCODE_DPAD_LEFT`, `KEYCODE_DPAD_RIGHT`, `KEYCODE_DPAD_UP` pour le joystick
- `KEYCODE_ENTER` pour la touche Entrée
- `KEYCODE_ENVELOPE` pour la touche spéciale Enveloppe
- `KEYCODE_EXPLORER` pour la touche spéciale Explorateur
- `KEYCODE_EQUALS`, `KEYCODE_MINUS` ,
- `KEYCODE_LEFT_BRACKET` , `KEYCODE_RIGHT_BRACKET` pour les crochets
- `KEYCODE_MEDIA_FAST_FORWARD` , `KEYCODE_MEDIA_NEXT` , `KEYCODE_MEDIA_PLAY_PAUSE` , `KEYCODE_MEDIA_PREVIOUS` , `KEYCODE_MEDIA_REWIND` , `KEYCODE_MEDIA_STOP` pour les touches de contrôle des médias

- KEYCODE_MUTE , KEYCODE_VOLUME_DOWN, KEYCODE_VOLUME_UP pour les touches de contrôle du volume

La classe KeyEvent

import android.view.KeyEvent

Associée aux événements clavier. Ses principales méthodes sont les suivantes :

- getAction () qui renvoie un entier pouvant prendre les valeurs ACTION_DOWN ou ACTION_UP qui indique l'action faite sur la touche (appuyée, lâchée).
- getRepeatCount() renvoie le nombre de répétitions lorsque la touche est maintenue
- getKeyCode() renvoie le code de la touche (même valeur que le dernier paramètre de onKey, voir ci-dessus)
- Les méthodes isAltPressed() , isShiftPressed() , isCtrlPressed() , isCapsLockOn() , isNumLockOn() , isScrollLockOn() permettent de tester l'état des touches de modification.

L'interface OnTouchListener

import android.view.View.OnTouchListener

Définit les écouteurs d'événements de type touché. La méthode à surcharger pour traiter les événements est :

- onTouch(View, MotionEvent)

Le premier paramètre est la vue touchée, le second paramètre est l'événement de touché (voir ci-dessous).

La classe MotionEvent

import android.view.MotionEvent

Associée aux événements de déplacement (souris, écran tactile). Ses méthodes sont les suivantes :

- getAction() qui renvoie un entier pouvant prendre les valeurs ACTION_DOWN , ACTION_UP , ACTION_MOVE ou ACTION_OUTSIDE
- getPressure() renvoie un réel entre 0 et 1 indiquant la force de pression du touché sur l'écran
- getX() et getY() renvoient un réel entre -1 et 1 indiquant les coordonnées (resp en x et en y).
- getXPrecision() et getYPrecision() renvoient un réel indiquant la précision des coordonnées (resp en x et en y).

L'interface OnCreateContextMenuListener

import android.view.View.OnCreateContextMenuListener

Définit les écouteurs d'événements de type ouverture de menu contextuel. La méthode à surcharger pour traiter les événements est :

- onCreateContextMenu(ContextMenu, View, ContextMenu.ContextMenuInfo)

Le premier paramètre est le menu ouvert, le deuxième paramètre est la vue associée au menu contextuel, le troisième paramètre contient des informations sur ce menu, son contenu dépend de la vue.

L'interface OnDragListener

import android.view.View.OnDragListener

Définit les écouteurs d'événements de type déplacement d'un objet vers la vue. La méthode à surcharger pour traiter les événements est :

- onDrag(View, DragEvent)

Le premier paramètre est la vue sur laquelle a été déplacé l'objet, le deuxième paramètre est l'événement de déplacement.

La classe DragEvent

import android.view.DragEvent

Associée aux événements de déplacement d'objets. Ses principales méthodes sont les suivantes :

- `getAction()` retourne l'identificateur de l'action effectuée. Cet indicateur peut prendre les valeurs : `ACTION_DRAG_STARTED` , `ACTION_DRAG_ENTERED` , `ACTION_DRAG_LOCATION` , `ACTION_DROP` , `ACTION_DRAG_EXITED` ou `ACTION_DRAG_ENDED`
- `getResult()` renvoie un booléen qui indique si le déplacement a pu être fait (accepté par la vue). Cette méthode ne renvoie une valeur significative que si la précédente renvoie `ACTION_DRAG_ENDED`
- `getX()` et `getY()` renvoient les coordonnées du point correspondant à l'événement. Ces valeurs ne sont significatives que si l'action est `ACTION_DRAG_ENTERED`, `ACTION_DRAG_LOCATION`, `ACTION_DROP`, ou `ACTION_DRAG_EXITED`.

L'interface OnFocusChangeListener

import android.view.View.OnFocusChangeListener

Définit les écouteurs d'événements de type changement de focus. La méthode à surcharger pour traiter les événements est :

- `focusChange(View, boolean)`

Le premier paramètre est la vue ayant changé de focus, le deuxième paramètre indique si le focus a été gagné ou perdu (true/false).

L'interface OnTimeChangedListener

import android.view.View.OnFocusChangeListener

Définit les écouteurs d'événements de type choix dans un TimePicker. La méthode à surcharger pour traiter les événements est :

- `onTimeChanged(TimePicker, int, int)`

Le premier paramètre est le `TimePicker` lui-même, le deuxième paramètre est l'heure choisie et le dernier paramètre la minute choisie.

L'interface OnScrollListener

import android.widget.AbsListView.OnScrollListener

Définit les écouteurs d'événements de type défilement. Les méthodes à surcharger pour traiter les événements sont :

- `onScroll(AbsListView, int, int, int)`

Le premier paramètre est le `ListView` ou le `GridView` lui-même, le deuxième paramètre est le rang du premier élément visible, le troisième paramètre est le nombre d'éléments visibles et le dernier paramètre le nombre total d'éléments.

- `onScrollStateChange(AbsListView, int)`

Le premier paramètre est le `ListView` ou le `GridView` lui-même. Le deuxième paramètre est l'état du défilement, il peut prendre les valeurs `SCROLL_STATE_IDLE`, `SCROLL_STATE_TOUCH_SCROLL` ou `SCROLL_STATE_FLING`.

L'interface OnItemClickListener

import android.widget.AdapterView.OnItemClickListener

Définit les écouteurs d'événements de type clic sur un élément dans une liste. La méthode à surcharger pour traiter les événements est :

- `onItemClicked(AdapterView, View, int, long)`

Le premier paramètre est la liste elle-même, le deuxième paramètre est l'élément de la liste qui a été cliqué, le troisième paramètre est le rang de cet élément dans la liste et le dernier paramètre son identificateur.

L'interface OnItemLongClickListener

import android.widget.AdapterView.OnItemLongClickListener

Définit les écouteurs d'événements de type clic long sur un élément dans une liste. La méthode à surcharger pour traiter les événements est :

- `onItemLongClicked(AdapterView, View, int, long)`

Le premier paramètre est la liste elle-même, le deuxième paramètre est l'élément de la liste qui a été cliqué, le troisième paramètre est le rang de cet élément dans la liste et le dernier paramètre son identificateur.

L'interface OnItemSelectedListener

import android.widget.AdapterView.OnItemselectedListener

Définit les écouteurs d'événements de type séléction d'un élément dans une liste. Les méthodes à surcharger pour traiter les événements sont :

- `onItemSelected(AdapterView, View, int, long)`

Le premier paramètre est la liste elle-même, le deuxième paramètre est l'élément de la liste qui a été sélectionné, le troisième paramètre est le rang de cet élément dans la liste et le dernier paramètre son identificateur.

- `onNothingSelected(AdapterView)`

Le paramètre est la liste elle-même, aucun élément n'a été sélectionné.

L'interface OnHierarchyChangeListener

import android.view.ViewGroup.OnHierarchyChangeListener

Définit les écouteurs d'événements de type ajout ou supression d'un élément dans un groupe. Les méthodes à surcharger pour traiter les événements sont :

- `onChildViewAdded(View, View)`

Le premier paramètre est le groupe, le deuxième paramètre est l'élément ajouté à ce groupe.

- `onChildViewRemoved(View, View)`

Le premier paramètre est le groupe, le deuxième paramètre est l'élément enlevé à ce groupe.

L'interface TextWatcher

import android.text.TextWatcher

Définit les écouteurs d'événements sur la modification de texte. Les méthodes à surcharger pour traiter les événements sont :

- `beforeTextChanged(CharSequence, int, int, int)` appelée avant la modification du texte

Le premier paramètre est le texte avant sa modification, le deuxième paramètre est le point de départ de la modification (rang du caractère), le dernier paramètre est le nombre de caractères remplacés.

- `onTextChanged(CharSequence, int, int, int)` appelée dès la modification du texte

Les paramètres sont la mêmes que pour la précédente.

- `afterTextChanged(Editable)` appelée après la modification du texte

Le paramètre est le texte après sa modification.

Attention : ces méthodes ne doivent pas modifier le texte (1er paramètre) afin d'éviter les appels récursifs.

L'interface OnEditorActionListener

import android.view.ViewGroup.OnHierarchyChangeListener

Définit les écouteurs d'événements sur les actions sur du texte. La méthode à surcharger pour traiter les événements est :

- `onEditorAction`(`TextView`, int, `KeyEvent`) Cette méthode doit retourner un booléen (true si l'événement a été traité)

Le premier paramètre est le widget qui gère le texte (`TextView` ou classe dérivée) lui-même, le deuxième paramètre est un identificateur d'action ou la valeur `EditorInfo.IME_NULL` lorsque l'action provient de la touche Entrée. Le dernier paramètre est l'événement clavier (voir ci-dessus la classe KeyEvent), il n'a de sens que si l'action provient de la touche Entrée (`EditorInfo.IME_NULL`) dans le cas contraire il vaut *null*.

L'interface OnCheckedChangeListener

import android.widget.CompoundButton.OnCheckedChangeListener

Définit les écouteurs d'événements de type changement d'état d'un bouton ou d'une case à cocher. La méthode à surcharger pour traiter les événements est :

- `onCheckedChanged(CompoundButton, boolean)`

Le premier paramètre est le bouton ou la case, le deuxième paramètre indique son nouvet état (true si coché).

L'interface OnDateChangeListener

import android.widget.CalendarView.OnDateChangeListener

Définit les écouteurs d'événements de type changement de date. La méthode à surcharger pour traiter les événements est :

- `onSelectedDayChange(Calendar, int, int, int)`

Le premier paramètre est le CalendarView, les paramètres suivants sont, dans l'ordre, l'anné, le mois et la date choisis.

L'interface OnChronometerTickListener

import android.widget.Chronometer.OnChronometerTickListener

Définit les écouteurs d'événements de type changement de valeur d'un chronomètre. La méthode à surcharger pour traiter les événements est :

- `onChronometerTick(Chronometer)`

Le paramètre est le chronomètre lui-même.

L'interface OnRatingBarChangeListener

import android.widget.RatingBar.OnRatingBarChangeListener

Définit les écouteurs d'événements de type changement de valeur d'un RatingBar. La méthode à surcharger pour traiter les événements est :

- `onRatingChanged(RatingBar, float, boolean)`

Le premier paramètre est le RatingBar lui-même, le deuxième paramètre est la valeur actuelle, le troisième paramètre vaut true si la valeur a été modifiée par l'utilisateur (false si c'est par programme).

L'interface OnSeekBarChangeListener

import android.widget.RatingBar.OnRatingBarChangeListener

Définit les écouteurs d'événements de type changement de position d'un curseur. Les méthodes à surcharger pour traiter les événements sont :

- `onProgressChanged(SeekBar, int, boolean)` quand le curseur a été déplacé

Le premier paramètre est le curseur lui-même, le deuxième paramètre est la valeur actuelle, le troisième paramètre vaut true si la valeur a été modifiée par l'utilisateur (false si c'est par programme).

- `onStartTrackingTouch(SeekBar)` lorsque l'utilisateur commence à toucher le curseur
- `onStopTrackingTouch(SeekBar)` lorsque l'utilisateur cesse de toucher le curseur

Notifications

Android propose un mécanisme simple pour envoyer des informations visuelles à l'utilisateur : la classe `Toast`. Il s'agit d'une fenêtre qui apparaît en premier plan puis disparaît seule au bout d'un temps donné (2 ou 5 secondes).

Notifications non persistantes

Elles sont réalisées à l'aide de la classe statique Toast. Elles apparaissent pour une durée courte à l'écran.

Méthodes de la classe Toast :

Création d'un Toast

- `Toast.makeText(Context, String, int)` cette méthode renvoie l'objet de classe `Toast` créé. Le premier paramètre est 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).

Positionnement d'un Toast

- `setGravity(int, int, int)` cette méthode doit être appelée avant l'affichage par show (voir ci-dessous) pour indiquer où s'affichera le message. Le premier paramètre sert à placer le message par rapport à l'écran. Il peut prendre l'une des valeurs définies dans la classe Gravity soit : `Gravity.(TOP, BOTTOM, LEFT, RIGHT, CENTER_VERTICAL, FILL_VERTICAL, CENTER_HORIZONTAL, FILL_ HORIZONTAL, CENTER, FILL)`. Les deux paramètres suivants indiquent le décalage horizontal et vertical (en pixels).

Affichage d'un Toast

- `show()` cette méthode affiche le message pour la durée définie lors de sa création.

Notifications persistantes

Android propose également des notifications persistantes qui apparaissent en haut de l'écran sous la forme d'une icône, d'un titre et d'un texte. Le texte est cliquable et permet de lancer une sous acticité (comme startActivityForResult).

Pour créer le contenu de la notification :

```
NotificationCompat.Builder notification = new NotificationCompat.Builder(contexte);
notification.setSmallIcon(R.drawable.icone_de_notification); // icone
notification.setContentTitle("Titre de la Notification"); // titre
notification.setContentText("Texte cliquable de la Notification");  // texte
```

Pour indiquer quelle sous activité doit être lancée par la notification :

```
Intent appel = new Intent(this, ActiviteAppelee.class); // Activité lancée par la notification
PendingIntent demarrage = PendingIntent.getActivity(this, 0, appel,
                                                    PendingIntent.FLAG_UPDATE_CURRENT);
notification.setContentIntent(demarrage); // Associer l'activité à la notification
```

Pour afficher la notification :

```
int identNotification = 1; // Identifiant de la notification (comme pour startActivityForResult)
NotificationManager gestNotif = (NotificationManager)
                                                    getSystemService(NOTIFICATION_SERVICE);
gestNotif.notify(identNotification, notification.build()); // La notification apparaît
```

On peut également supprimer la notification par :

```
gestNotif.cancel(identNotification); // Identifiant de la notification
```

Textes Formatés

Les widgets qui affichent du texte (TextView, Button, CheckBox …) acceptent, pour définir ce texte, des objets de classe `CharSequence`. On peut utiliser la classe `String` qui hérite de `CharSequence` mais elle ne permet pas de formatter le texte. Le format (taille, police, ..) défini dans le fichier XML s'applique donc à tout le texte. La classe `SpannableStringBuilder` qui hérite aussi de `CharSequence` (et qui peut donc être utilisée au lieu de `String`) permet d'apposer des formats à des parties de texte.

Création d'un texte formaté :

`SpannableStringBuilder` texte = new `SpannableStringBuilder`("chaine à formater");

Application de formats :

Cela se fait par la méthode `setSpan` de `SpannableStringBuilder` :

```
texte.setSpan(CharacterStyle, int, int,
                        Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
```

– Le premier paramètre permet de définir :
- La couleur du texte en y mettant : new `ForegroundColorSpan`(int) qui reçoit en paramètre un code de couleur (voir classe Color)
- La couleur du fond en y mettant : new `BackgroundColorSpan`(int) qui reçoit en paramètre un code de couleur (voir classe Color)
- La police en y mettant : new `TypeSpaceSpan`(String) qui reçoit en paramètre le nom de police qui peut être : `normal, sans, serif` ou `monospace`
- La taille en y mettant l'un des objets suivants :
  - new `AbsoluteSizeSpan`(int, boolean) qui reçoit en paramètre la taille en pixels (px) si le 2ème paramètre est false ou est omis et en pixels à densité indépendante (dp) sinon
  - new `RelativeSizeSpan`(float) qui reçoit en paramètre l'échelle de taille (1.5F signifie 1 fois et demi plus gros)
- L'étirement horizontal en y mettant : new `ScaleXSpan`(float) qui reçoit en paramètre l'échelle de taille (1.5F signifie étiré 1 fois et demi en largeur)
- Le style du texte en y mettant : new `StyleSpan`(`Typeface`) qui reçoit en paramètre le style qui est l'une des constantes :
  - `android.graphics.Typeface.BOLD`
  - `android.graphics.Typeface.BOLD_ITALIC`
  - `android.graphics.Typeface.ITALIC`
  - `android.graphics.Typeface.NORMAL`
- Le mode souligné en y mettant : new `UnderlineSpan`()

– Le deuxième paramètre permet de définir le rang du caractère à partir duquel commence à s'appliquer ce format

– Le troisième paramètre permet de définir le rang du caractère à partir duquel cesse de s'appliquer ce format

Bien entendu plusieurs formats peuvent s'appliquer sur la même partie de texte (par exemple : gras + souligné+ rouge sur fond vert) en appelant plusieurs fois la méthode `setSpan`.

Couleurs et Images

Les couleurs sont représentées par un entier sur 32 bits (8 bits de transparence (FF=opaque, 0=totalement transparent), 8 bits de rouge, 8 bits de vert et 8 bits de bleu). On peut manipuler ces codes à l'aide de la classe `Color`.

La classe Color d'Android

import android.graphics.Color

Méthodes de la classe Color

- `argb`(int, int, int, int) renvoie le code de la couleur définie par les 4 paramètres (transparence, rouge, vert, bleu). Le 1er paramètre peut être omis pour obtenir une couleur opaque.
- `alpha`(int) renvoie la transparence de la couleur dont le code est passé en paramètre
- `red`(int) renvoie la composante rouge de la couleur dont le code est passé en paramètre
- `green`(int) renvoie la composante verte de la couleur dont le code est passé en paramètre
- `blue`(int) renvoie la composante bleue de la couleur dont le code est passé en paramètre

Couleurs prédéfinies

Un certain nombre de couleurs sont prédéfinies en contantes dans la classe `Color` : `Color.BLACK`, `Color.WHITE`, `Color.LTGRAY`, `Color.GRAY`, `Color.DKGRAY`, `Color.RED`, `Color.GREEN`, `Color.BLUE`, `Color.CYAN`, `Color.MAGENTA`, `Color.YELLOW` et `Color.TRANSPARENT`

La classe Drawable

import android.graphics.drawable.Drawable

Les images, quand elles sont récupérées dans les ressources, apparaissent sous la forme d'objets de la classe `Drawable`. C'est sous cette forme qu'elles peuvent être utilisées dans les divers éléments d'interface (images de fond, images affichées dans des boutons ou des `ImageView`, séparateurs de liste, …). Elles peuvent également être affichées dans un objet de classe `Canvas` par leur méthode `draw`(`Canvas`).
Un objet de classe Drawable peut être créé à partir d'un fichier, d'un flux d'entrée, d'une ressource ou d'un fichier XML grâce aux méthodes statiques :

- `createFromPath`
- `createFromStream`
- `createFromResourceStream`
- `createFromXml`

Les menus

import android.view.Menu
import android.view.MenuItem
import android.view.MenuInflater
import android.view.ContextMenu
import android.view.ContextMenuInfo

Android propose 2 types de menus:

- Menu général associé à l'interface d'une activité et qui apparaît lorsque l'on utilise la touche "menu" du téléphone
- Menu contextuel lié à un élément d'interface qui apparaît lorsque l'on touche longuement cet élément

Description en XML d'un menu

Le fichier de description de menu doit être placé dans **res/menu** (ce répertoire n'est pas créé par Android Studio il faudra donc le faire).
Structure du fichier XML :

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/nom_du_choix_1"
          android:icon="@drawable/image_du_choix_1"
          android:title="@string/texte_du_choix_1" />
     …
    <item … />
</menu>
```

A chaque élément de menu peut être associée une icône et un texte. On peut, bien entendu ne mettre que l'un des deux.
La création de sous menus dans un item se fait comme suit :

```
<item android:id="@+id/nom_du_choix_N"
      android:icon="@drawable/image_du_choix_N"
      android:title="@string/texte_du_choix_N">
    <menu>
        <item android:id="@+id/nom_du_sous_choix_1"
              android:title="texte_du_sous_choix_1" />
        …
    </menu>
</item>
```

Menu général

Création du menu

Elle se fait dans la méthode `onCreateOptionMenu(Menu)` de l'activité à partir du fichier xml de description sous la forme :

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.nom_du_fichier_xml_du_menu, menu);
    return true;
}
```

Réactions aux choix

Elle se fait dans la méthode `onOptionsItemSelected`(MenuItem) de l'activité. Cette méthode est appelée lorsque l'utilisateur fait un choix dans un menu ou un sous menu. Elle doit renvoyer un booléen à *true* si l'événement a été traité. Sa forme est celle d'un switch où l'on traite chaque item de menu et de sous menu :

```
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
      case R.id.nom_du_choix_1:
          // traitement du choix 1
          return true;
      …
      case R.id.nom_du_sous_choix_1:
          // traitement du sous choix 1
          return true;
      …
      default:
          return super.onOptionsItemSelected(item);
     }
}
```

Menus contextuels

Association

Un menu contextuel est associé à un élément d'interface. Ceci se fait lors de la création de l'interface (dans `onCreate`) de la façon suivante :

```
registerForContexMenu(element_associe_au_menu_contextuel);
```

Création du menu contextuel

La création se fait dans la méthode `onCreateContextMenu` sous la forme :

```
public void onCreateContextMenu(ContextMenu menu, View element,
                                ContextMenuInfo info) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.nom_du_fichier_xml_du_menu, menu);
}
```

Réactions aux choix

Elle se fait dans la méthode `onContextItemSelected(MenuItem)` de l'activité. Cette méthode est appelée lorsque l'utilisateur fait un choix dans un menu contextuel. Elle doit renvoyer un booléen à *true* si l'événement a été traité. Sa forme est celle d'un switch où l'on traite chaque item de menu et de sous menu :

```
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.nom_du_choix_1:
        // traitement du choix 1
        return true;
    …
    case R.id.nom_du_sous_choix_1:
        // traitement du sous choix 1
        return true;
    …
    default: return super.onContextItemSelected(item);
   }
}
```

Barre de menu (ActionBar)

C'est une barre de menu qui apparaît en haut de la fenêtre et qui contient l'icône de l'application, son nom et des boutons déclechant des actions.

Pour assurer la compatibilité avec des versions antérieures d'Android :

- L'activité doit hériter de AppCompatActivity
- Le thème de l'application doit être de type : NoActionBar (si on veut assurer la compatibilité) :
  `<application android:theme="@style/Theme.AppCompat.Light.NoActionBar" />`

Création de la barre de menu

La barre doit être définie dans le fichier XML de l'interface par :

```
<android.support.v7.widget.Toolbar
  android:id = "@+id/maBarreDAction"
  android:layout_width = "match_parent"
  android:layout_height = "?attr/actionBarSize"
  android:elevation = "4dp"
  android:theme = "@style/ThemeOverlay.AppCompat.ActionBar"
  app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
```

Enfin, elle est récupérée et ajoutée à l'interface (dans le onCreate de l'activité) par :

```
Toolbar maBarre = (Toolbar) findViewById(R.id.maBarreDAction);
setSupportActionBar(maBarre );
```

Le contenu de la barre est défini par un fichier XML du même type que celui utilisé pour les menus (voir ci-dessus) :

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
   <item
      android:id = "@+id/action1"
      android:icon = "@drawable/icone1"
      android:title = "@string/action 1"
      app:showAsAction = "visibilité"/>
          ….
</menu>
```

Pour chaque élément, on peut mettre :

- Une icône (android:icon)
- Un titre (android:title)
- Ou les deux

visibilité peut être :

- always apparaît toujours
- never n'apparaît que dans les actions supplémentaires
- ifRoom apparaît s'il y a suffisamment de place

Réactions aux choix

L'écouteur d'événements associé aux boutons de la barre s'écrit comme pour les menus par surcharge de la méthode onOptionsItemSelected de l'activité :

```
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.action1:
         // L'utilisateur a cliqué sur le bouton correspondant à l'action 1
                  // traitement de l'action 1
         return true;

      case R.id.action2:
         // L'utilisateur a cliqué sur le bouton correspondant à l'action 2
               // traitement de l'action 2
         return true;

      default:
         return super.onOptionsItemSelected(item);

   }
}
```

Navigation entre sous activités par l'ActionBar

Normalement quand une sous activité se termine on revient à celle qui l'a lancée (empilement d'écrans).

On peut ajouter un bouton à la barre d'actions pour quitter la sous activité actuelle et revenir à une autre (pas forcément celle qui l'a lancée) par :

```
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

If faut alors définir dans le fichier AndroidManifest une activité parente pour chaque sous activité :

```
<activity
      android:name = ".ActivitePrincipale" ...>
      …
 </activity>

 <activity
          android:name = ".sousActivite1"
          android:label = "@string/nomSousActivite1 "
          android:parentActivityName = ".ActivitePrincipale" >
          <meta-data
          android:name = "android.support.PARENT_ACTIVITY"
          android:value = ".ActivitePrincipale" />
</activity>
```

Communication entre activités

```
import android.os.Bundle
import android.content.Intent
import android.net.Uri
```

Une activité peut en lancer d'autres, leur passer des paramètres et en récupérer des valeurs de retour.

Filtres d'intensions

Ils permettent à Android de savoir si une activité correspond à un type d'appel. On peut associer de tels filtres à toutes les activités d'une application.

Forme générale

```
<activity android:name=".Nom_De_La_Classe_De_L_Activité"
…
>
      <intent-filter>
            <action android:name="nom_d_action_1" />
            …
            <action android:name="nom_d_action_N" />
            <category android:name="nom_de_categorie_1" />
            …
            <category android:name="nom_de_categorie_N" />
            <data android:mimeType="nom_de_type_mime"
                  android:scheme="protocole://hote:port/chemin" />
            />
      </intent-filter>
            …
      <intent-filter>
            …
      </intent-filter>
</activity>
```

Lancer une activité

Lorsqu'une activité lance une autre activité elle peut ou pas vouloir en récupérer une réponse (valeur de retour).

Sans attente de réponse

Dans le cas où l'activité lancée est incluse dans l'application et connue par le nom de sa classe :

```
Intent demarre = new Intent(getApplicationContext(),
                                    NomDeLaClasseDeLActiviteALancer.class);
startActivity(demarre);
```

Dans le cas où l'activité lancée est installée sur le périphérique et on laisse Android choisir l'activité à lancer grâce aux filtres d'intensions :

```
Intent naviguer = new Intent(action);
startActivity(naviguer);
```

Remarque : le paramètre du constructeur de l'Intent est une chaîne de caractères prédéfinie correspondant au filtre d'intensions de l'activité à lancer. On peut l'obtenir par la documentation de cette activité ou en utilisant une constante prédéfinie de l'une des bibliothèques d'Android.

Pour ajouter des informations à un `Intent` il existe les méthodes :

- `addCategory`(String) ajoute une catégorie
- `setData`(`URI`) définit une `URI` pour les données
- `setDataAndType`(`URI`, String) définit une `URI` et un type mime pour les données

Avec attente de réponse

Le processus de lancement est le même que précédemment la seule différence porte sur la méthode utilisée pour le lancement : `startActivity` est remplacé par `startActivityForResult`.

Lancement de l'activité B dans l'activité A :

```
private static final int ACTIVITE_B = 1; // code servant à identifier l'activité quand elle répondra
// Si l'activité lancée est incluse dans l'application
Intent demarre = new Intent(getApplicationContext(), NomDeLaClasseDeLActiviteB.class);
// Si l'activité lancée est installée sur le périphérique
```

```
Intent demarre = new Intent(action);
startActivityForResult(demarre, ACTIVITE_B); // lancement de l'activité B
```

Renvoi de réponse et terminaison de l'activité B :

```
Intent intent_retour = new Intent(); // Préparer un Intent pour les valeurs de retour
// Ajouter les valeurs de retour à l'Intent intent_retour (voir plus loin)
setResult(code, intent_retour);  // renvoyer un code de retour (entier :  on peut utiliser les constantes
                                 //  RESULT_CANCELED et  RESULT_OK)  et l'Intent de retour
finish(); // terminer l'activité B
```

Récupération de la réponse de l'activité B dans l'activité A :

```
protected void onActivityResult(int ident, int code_retour, Intent retour) {
    switch(ident) {
        case ACTIVITE_B : // c'est l'activité B qui a répondu
            // traitement selon le code de retour contenu dans l'entier "code_retour"
            // récupération des valeurs de retour contenues dans l'Intent "retour" (voir ci-dessous)
            return;
        …
    }
}
```

Les valeurs de retour sont récupérées grâce à leur nom dans l'Intent "retour", selon leur type, par :

Type ou classe de la valeur de retour	Méthode à utiliser	Paramètre(s)
boolean	getBooleanExtra	Nom, Valeur par défaut
boolean[]	getBooleanArrayExtra	Nom
byte	getByteExtra	Nom, Valeur par défaut
byte[]	getByteArrayExtra	Nom
char	getCharExtra	Nom, Valeur par défaut
char[]	getCharArrayExtra	Nom
double	getDoubleExtra	Nom, Valeur par défaut
double[]	getDoubleArrayExtra	Nom
float	getFloatExtra	Nom, Valeur par défaut
float[]	getFloatArrayExtra	Nom
int	getIntExtra	Nom, Valeur par défaut
int[]	getIntArrayExtra	Nom
long	getLongExtra	Nom, Valeur par défaut
long[]	getLongArrayExtra	Nom
short	getShortExtra	Nom, Valeur par défaut
short[]	getShortArrayExtra	Nom
CharSequence	getCharSequenceExtra	Nom
String	getStringExtra	Nom
ArrayList<Integer>	getIntegerArrayListExtra	Nom
ArrayList<String>	getStringArrayListExtra	Nom

Exemples :

```
boolean x =  retour.getBooleanExtra("nom_de_la_valeur_de_retour_x", false) ;
int y = retour.getIntExtra("nom_de_la_valeur_de_retour_y", -1) ;
…
boolean[] tab = retour.getBooleanArrayExtra ("nom_de_la_valeur_de_retour_tab") ;
…
```

ATTENTION : tester le cas où "retour" serait **null** pour éviter le plantage si on quite l'activité par le bouton de retour en arrière du téléphone plutôt que normalement.

Passage de paramètres et de valeurs de retour

Le passage des paramètres de A à B ainsi que des valeurs de retour de B à A se font par l'`Intent` en utilisant :

```
putExtra(String, val)
```

Le 1er paramètre est un nom (clé) qui permet d'identifier la valeur associée
Le second paramètre est la valeur :

- De type simple (boolean, byte, int, short, long, float, double, char)
- Tableau de types simples
- String ou CharSequence
- Tableau de Strings

Ou les méthodes :

- `putIntegerArrayListExtra`(String, ArrayList<Integer>)
- `putStringArrayListExtra`(String, ArrayList<String>)

Pour les collections d'entiers ou de chaînes.

La récupération des paramètres dans B se fait en récupérant le `Bundle` qui les contient dans `l'Intent` :

```
Bundle params = getIntent().getExtras()
```

puis en utilisant les méthodes `getBoolean` , `getInt`, … , `getBooleanArray`, `getIntArray` … de cet objet selon le type du paramètre à récupérer. Ces méthodes sont les mêmes que celles du tableau précédent sans la terminaison « Extra » :

```
boolean x =  params.getBoolean("nom_du_paramètre_x")
int taille = params.getInt("nom_du_paramètre_taille")
…
boolean[] tab = params.getBooleanArray("nom_du_paramètre_tab")
…
```

Remarque : Android ne permet pas de passer simplement des objets de classe quelconque dans un `Intent`. Si l'on veut passer des objets de classe A il faut que celle-ci implemente l'interface `Parcelable` qui est comparable à l'interface Serializable de java mais nécessite la réecriture par le programmeur de méthodes pour créer, lire et écrire un objet de classe A ainsi qu'un créateur d'objet ou de tableaux d'objets de classe A.

Le matériel et les capteurs

Envoi d'un SMS

```
import android.telephony.SmsManager
<uses-permission android:name="android.permission.SEND_SMS" />
```

On utilise la classe `SmsManager` dont une instance est obtenue par :

```
SmsManager gestionnaireSms = SmsManager.getDefault();
```

Pour envoyer un message on fait :

```
gestionnaireSms.sendTextMessage("numero", null, "message", null, null);
```

Utilisation du GPS

```
import android.location.LocationManager
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
```

On utilise la classe LocationManager dont une instance est obtenue par :

```
LocationManager  geoloc =
(LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

On lui associe un écouteur d'événements qui recevra les informations de géolocalisation par la méthode :

```
geoloc.requestLocationUpdates(String,  long, float,  LocationListener)
```

- Le premier paramètre indique le moyen utilisé pour la géolocalisation, il peut en particulier prendre les valeurs :
  - LocationManager.GPS_PROVIDER pour utiliser le GPS
  - LocationManager.NETWORK_PROVIDER pour utiliser la triangulation des antennes réseau

- Le deuxième paramètre est un entier qui indique l'intervalle minimal de temps (en ms) entre les notifications. On peut le mettre à 0 pour obtenir l'intervalle minimal possible
- Le troisième paramètre est un réel qui indique la distance minimale (en mètres) de déplacement pour provoquer une nouvelle notification. On peut le mettre à 0 pour obtenir l'intervalle minimal possible.
- Le dernier paramètre est l'écouteur d'événements qui est un objet d'une classe à définir par héritage de `LocationListener` et par surcharge des méthodes :
  - public void `onLocationChanged`(Location) qui est exécutée à chaque notification
  - public void `onProviderDisabled`(String) qui est exécutée si le fournisseur de localisation est désactivé par l'utilisateur. Le paramètre est le nom de ce fournisseur.
  - public void `onProviderEnabled`(String) qui est exécutée si le fournisseur de localisation est activé par l'utilisateur. Le paramètre est le nom de ce fournisseur.
  - public void `onStatusChanged`(String, int, Bundle) qui est exécutée si l'état du fournisseur de localisation change (redevient actif après une interruption …). Le premier paramètre est le nom de ce fournisseur, le deuxième indique son nouvel état (`OUT_OF_SERVICE`, `AVAILABLE` ou `TEMPORARILY_UNAVAILABLE`), le dernier paramètre contient des information supplémentaires sur ce fournisseur.

C'est surtout la première méthode qui est utile (les autres peuvent être écrites vides). Le paramètre de cette méthode est un objet de classe `Localisation` qui contient la position. Ses principales méthodes sont :

- `getLatitude`() qui renvoie un réel correspondant à la latitude
- `getLongitude`() qui renvoie un réel correspondant à la longitude
- `getAltitude`() qui renvoie un réel correspondant à l'altitude (en mètres)
- `getAccuracy`() qui renvoie un réel correspondant à la précision de la localisation (en mètres)
- `distanceTo`(`Location`) qui renvoie un réel représentant en mètres la distance entre la position actuelle et celle passée en paramètre.

Appareil photo

import android.hardware.Camera
import android.view.SurfaceView
import android.view.SurfaceHolder
import android.view.SurfaceHolder.Callback
<uses-permission android:name="android.permission.CAMERA" />

Accès à l'appareil photo

On déclare l'appareil photo comme une propriété de l'activité par :

```
private Camera photo;
```

Prévisualisation

L'appareil photo ne peut fonctionner que s'il dispose d'une zone d'écran pour prévisualiser la photo qu'il va prendre. On va donc lui associer un élément d'interface de classe `SurfaceView` qui sera déclaré dans le fichier xml de description de l'interface par :

```
<SurfaceView
      android:name="@id/visualisation"
      …
/>
```

Auquel on associera un écouteur d'événements pour gérer la prévisualisation par :

```
SurfaceView ecran = (SurfaceView)findViewById(R.id.visualisation);
SurfaceHolder surface = ecran.getHolder();
surface.addCallback(new GestionPrevisualisation());
surface.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

Et on créera cet écouteur d'événements par :

```
private class GestionPrevisualisation implements
                                        SurfaceHolder.Callback {
   public void surfaceCreated(SurfaceHolder surf) {
      photo = Camera.open();
      try {
        photo.setPreviewDisplay(surf);
      }
      catch (IOException ioe) {
        // erreur lors de l'association de la prévisualisation
        photo.stopPreview();
        photo.release();
        }
   }

   public void surfaceChanged(SurfaceHolder surf, int format,
                                    int largeur, int hauteur) {
   }

   public void surfaceDestroyed(SurfaceHolder holder) {
      if (photo != null) {
        photo.stopPreview();
        photo.release();
      }
   }
}
```

Prise de photo

A chaque prise de photo on doit lancer la prévisualisation et faire la photo en associant un écouteur d'événement qui récupèrera l'image par :

```
photo.startPreview();
photo.takePicture(Camera.ShutterCallback, Camera.PictureCallback,
                                          Camera.PictureCallback);
```

Les paramètres de la méthode `takePicture` sont les suivants :

- Les deux premiers permettent d'associer des écouteurs d'événements pour l'instant de prise de vue et pour la capture d'image non compressée (raw). Si l'on souhaite seulement récupérer la photo en jpeg on met null dans ces paramètres.
- Le dernier est un objet d'une classe qui implémente l'interface `PictureCallback` et dont on surcharge la méthode `onPictureTaken` qui sera exécutée lorsque la photo est disponible en jpeg. Cette méthode reçoit deux paramètres :
  - un tableau d'octets (byte[]) contenant l'image au format jpeg
  - l'appareil photo lui-même (objet de classe `Camera`)

Affichage de photo

```
import android.graphics.drawable.BitmapDrawable
import  android.graphics.BitmapFactory
import  android.graphics.BitmapFactory.Options
```

On peut transformer le tableau d'octets reçu en paramètre de `onPictureTaken` en objet de classe `BitmapDrawable` qui hérite de `Drawable` par le code suivant :

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inPurgeable = true; // pour libérer la mémoire prise par l'image
options.inSampleSize = x; // pour réduire l'image dans un rapport 1/x
BitmapDrawable image = new BitmapDrawable(
   BitmapFactory.decodeByteArray(tableau, 0, tableau.length, options));
```

Vibreur

import android.os.Vibrator
<uses-permission android:name="android.permission.VIBRATE" />

Pour accéder au vibreur on utilise la classe `Vibrator` dont une instance est obtenue par :

```
Vibrator vibreur = (Vibrator)getSystemService(Context.VIBRATOR_SERVICE);
```

Puis on peut faire vibrer le téléphone pour une durée donnée par la méthode :

vibreur.`vibrate`(int) où le paramètre est la durée de vibration en ms.

Capteurs

import android.hardware.Sensor
import android.hardware.SensorManager
import android.hardware.SensorEvent

Les types de capteurs sont les suivants (selon le modèle certains capteurs peuvent ne pas être disponibles) :

- `Sensor.TYPE_ACCELEROMETER`
- `Sensor.TYPE_AMBIENT_TEMPERATURE`
- `Sensor.TYPE_GRAVITY`
- `Sensor.TYPE_GYROSCOPE`
- `Sensor.TYPE_LIGHT`
- `Sensor.TYPE_LINEAR_ACCELERATION`
- `Sensor.TYPE_MAGNETIC_FIELD`
- `Sensor.TYPE_PRESSURE`
- `Sensor.TYPE_PROXIMITY`
- `Sensor.TYPE_RELATIVE_HUMIDITY`
- `Sensor.TYPE_ROTATION_VECTOR`
- `Sensor.TYPE_GAME_ROTATION_VECTOR`
- `Sensor.TYPE_GEOMAGNETIC_ROTATION_VECTOR`

Pour accéder à un capteur on utilise la classe `SensorManager` dont une instance est obtenue par :

```
SensorManager gestionnaireCapteurs =
(SensorManager)getSystemService(Context.SENSOR_SERVICE);
```

Puis on récupère un capteur particulier par :

```
Sensor monCapteur = gestionnaireCapteurs.getDefaultSensor(type_de_capteur);
```

Le paramètre type_de_capteur est l'un des types indiqués ci-dessus.

On lui associe un écouteur d'événements par la méthode :

gestionnaireCapteurs.`registerListener`(`SensorEventListener`, `Sensor`, int)

- Le premier paramètre est l'écouteur d'événements qui est un objet d'une classe à définir par implémentation de l'interface `SensorEventListener` et par surcharge des méthodes :
  - `void onSensorChanged(SensorEvent)` qui est exécutée chaque fois que le capteur effectue une nouvelle mesure. Le paramètre contient les valeurs mesurées dans sa propriété `values`. On peut donc récupérer ces valeurs par `nomDuParametre.values.clone()` qui renvoie une copie du tableau de réels contenant les mesures (voir ci-dessous).
  - `void onAccuracyChanged(Sensor, int)` qui est exécutée si la précision du capteur change. Le premier paramètre est le capteur dont la précision a changé, le second indique la nouvelle précision.
- Le deuxième paramètre est le capteur auquel on associe l'écouteur d'événements.
- Le dernier paramètre est le rythme de mesure (en microsecondes) ou l'une des valeurs prédéfinies suivantes : `SENSOR_DELAY_NORMAL`, `SENSOR_DELAY_UI`, `SENSOR_DELAY_GAME`, ou `SENSOR_DELAY_FASTEST`.

`SENSOR_DELAY_UI` correspond à la valeur la plus adéquate pour de l'interface tandis que `SENSOR_DELAY_GAME` correspond à la valeur la plus adéquate pour des jeux.

On peut supprimer un écouteur associé à un capteur par :

gestionnaireCapteurs.`unregisterListener(SensorEventListener, Sensor)` le premier paramètre est l'écouteur d'événements, le deuxième paramètre est le capteur auquel était associé cet écouteur d'événements.

Le tableau de valeurs récupéré par la méthode `values.clone()` contient 1 ou 3 valeurs qui doivent être interprétées selon le type de capteur (sa taille est toujours de 3 mais certaines valeurs ne sont pas significatives) :

Type de capteurs	valeurs	unité
`Sensor.TYPE_ACCELEROMETER`	3 valeurs : accélération en X, en Y et en Z Inclut la gravité	m/s²
`Sensor.TYPE_LINEAR_ACCELERATION`	3 valeurs : accélération en X, en Y et en Z N'inclut pas la gravité	m/s²
`Sensor.TYPE_GRAVITY`	3 valeurs : gravité en X, en Y et en Z	m/s²
`Sensor.TYPE_GYROSCOPE`	3 valeurs : vitesse de rotation angulaire en X, en Y et en Z	radians/s
`Sensor.TYPE_LIGHT`	1 valeur : luminosité	lux
`Sensor.TYPE_MAGNETIC_FIELD`	3 valeurs : champ magnétique en X, en Y et en Z	microtesla
`Sensor.TYPE_ROTATION_VECTOR`(2)	3 valeurs : – rotation par rapport à l'axe X – rotation par rapport à l'axe Y – rotation par rapport à l'axe Z	Sinus de la moitié de l'angle Y est orienté nord
`Sensor.TYPE_GAME_ROTATION_VECTOR`(2)	3 valeurs : – rotation par rapport à l'axe X – rotation par rapport à l'axe Y – rotation par rapport à l'axe Z	Sinus de la moitié de l'angle Pas de référence d'orientation
`Sensor.TYPE_GEOMAGNETIC_ROTATION_VECTOR`(2)	3 valeurs : – rotation par rapport à l'axe X – rotation par rapport à l'axe Y – rotation par rapport à l'axe Z	Sinus de la moitié de l'angle Y est orienté nord
`Sensor.TYPE_PRESSURE`	1 valeur : pression atmosphérique	hPa
`Sensor.TYPE_PROXIMITY`(1)	1 valeur : distance capteur/objet (mesuré par IR)	cm
`Sensor.TYPE_AMBIENT_TEMPERATURE`	1 valeur : température	°C
`Sensor.TYPE_RELATIVE_HUMIDITY`	1 valeur : humidité	%

(1) Sur certains modèles le capteur de proximité se contente de détecter une présence. La valeur mesurée n'est que 0 ou la valeur maximale (généralement 5 à 10 cm).

(2) Les mesures faites avec le TYPE_GEOMAGNETIC_ROTATION_VECTOR sont identiques à celles faites avec le TYPE_ROTATION_VECTOR seule la façon de les faire est différente : le 1er est moins précis mais consomme moins). Les mesures faites avec le TYPE_GAME_ROTATION_VECTOR sont, elles aussi, identiques à la différence que l'axe Y n'est pas orienté vers le nord. Le TYPE_GAME_ROTATION_VECTOR est le plus précis des 3.

Le multimédia

Android peut utiliser de nombreux formats audio et vidéo.

- Pour le son : 3gp , mp3 , mp4 , ogg , mid et wav
- Pour la vidéo : 3gp et mp4

Jouer un son

import android.media.MediaPlayer

Le fichier contenant le son doit être placé dans le répertoire **res/raw**. Il suffit alors de créer un lecteur audio par :

`MediaPlayer` lecteur = `MediaPlayer.create`(Context, int)

1. Le premier paramètre est le contexte de l'activité que l'on peut, par exemple, obtenir par `getApplicationContext()`
2. Le second paramètre est l'identificateur du fichier son obtenu par : `R.raw.nom_du_fichier_son`

Les opérations possibles sont alors :

lecteur.`start`() pour jouer le son
lecteur.`pause`() pour suspendre le son, il sera repris par `start`()
lecteur.`stop`() pour arrêter le son

Pour rejouer un son arrêté ou terminé il faut faire les 3 opérations suivantes :

lecteur.`reset`();
lecteur.`prepare`();
lecteur.`start`();

La classe MediaPlayer

Les principales méthodes de MediaPlayer sont :

Configuration

- `prepare()` initialisation du player
- `release()` libère les ressources du player (à faire dans la méthode `onDestroy` de l'activité)
- `reset()` réinitialisation du player
- `setDataSource(String)` définit le média par un chemin de fichier ou une URL
- `setDataSource(Context, Uri)` définit le média par une `Uri`, le premier paramètre est l'activité, le second l'`Uri` désignant le média.
- `setLooping(boolean)` met le player en mode boucle
- `setVolume(float, float)` définit le volume (le 1er paramètre est la voie gauche, le second la voie droite)

Controle

- `pause()` met en pause
- `seekTo(int)` déplacement dans le média en ms (en + ou en -)
- `start()` lancement
- `stop()` arrêt

Etat

- `getCurrentPosition()` renvoie la position actuelle dans le média (en ms)
- `getDuration()` renvoie la durée du média (en ms)
- `isPlaying()` renvoie true si le media est en cours
- `isLoopPlaying()` renvoie true si le media est en mode boucle

Evénements

- `setOnCompletionListener(MediaPlayer.OnCompletionListener)` associe un écouteur d'événements dont la méthode `onCompletion(MediaPlayer)` est appelée lorsque le média se termine
- `setOnBufferingUpdateListener(MediaPlayer.OnBufferingUpdateListener)` associe un écouteur d'événements dont la méthode `onBufferingUpdate(MediaPlayer, int)` est appelée lors de la mise à jour du buffer. Le second paramètre est le pourcentage de remplissage du buffer.
- `setOnPreparedListener(MediaPlayer.OnPreparedListener)` associe un écouteur d'événements dont la méthode `onPrepared(MediaPlayer)` est appelée lorsque le `MediaPlayer` est prêt.

- `setOnSeekCompleteListener(MediaPlayer.OnSeekCompleteListener)` associe un écouteur d'événements dont la méthode `onSeekComplete(MediaPlayer)` est appelée lorsqu'un déplacement dans le média est terminé.

Afficher une video

import android.widget.VideoView
import android.net.Uri

Remarque : en général l'affichage de vidéos ne fonctionne pas correctement sur l'émulateur (on a le son mais l'image est figée ou saccadée) pour des raisons d'émulation d'accélération graphique.

Le fichier contenant la vidéo doit être placé dans le répertoire **res/raw** ou accessible sur une URI. Ce dernier cas sera décrit plus loin.
L'interface doit comporter une zone d'affichage pour la vidéo qui sera déclarée dans le fichier xml de définition d'interface par :

```
<VideoView android:id="@+id/ecran_video"
android:layout_width="fill-parent"
android:layout_height="fill-parent" />
```

Le chemin d'accès à la vidéo est décrit par une Uri :

```
Uri chemin =
Uri.parse("android.resource://paquetage_de_l_application/"
                                    +R.raw.nom_du_fichier_video);
```

On va ensuite associer ce média à la zone d'affichage par :

```
VideoView lecteur = (VideoView)findViewById(R.id.ecranVideo);
lecteur.setVideoURI(chemin);
lecteur.requestFocus();
```

La classe VideoView

Les principales méthodes de VideoView sont :

Configuration

- `setMediaController(MediaController)` associe un contrtôleur de média qui apparaîtra par un clic long sur la vidéo et affiche les boutons de contrôle.
- `setVideoPath(String)` définit le média par un chemin de fichier
- `setVideoURI(Uri)` définit le média par une Uri

Controle

- `start()` lancement
- `pause()` mise en pause, reprise par `start()`
- `seekTo(int)` déplacemant dans le média, le paramètre est un temps en ms à partir du début
- `stopPlayback()` arrêt définitif ne sera pas relancé par `start()`

Etat

- `canPause()` renvoie true si le media peut être mis en pause
- `canSeekBackward()` renvoie true si le media peut être reculé
- `canSeekForward()` renvoie true si le media peut être avancé
- `getBufferPercentage()` renvoie le pourcentage d'occupation du buffer de média
- `getCurrentPosition()` renvoie la position actuelle dans le média (en ms)
- `getDuration()` renvoie la durée du média (en ms)
- `isPlaying()` renvoie true si le media est en cours

Evénements

- `setOnCompletionListener(MediaPlayer.OnCompletionListener)` associe un écouteur d'événements dont la méthode `onCompletion(MediaPlayer)` est appelée lorsque le média se termine.

Synthèse de parole

import android.speech.tts.TextToSpeech
import android.speech.tts.TextToSpeech.OnInitListener

Android permet de lire à haute voix un texte par synthèse de parole. Il faut, bien entendu, que le synthétiseur connaisse la langue du texte. On peut installer des applications permettant d'ajouter des langues.

Pour créer et préparer le synthétiseur :

- `parle = new TextToSpeech(getApplicationContext(), new SynthPret());`

Pour attendre que le synthétiseur soit prêt on utilise l'écouteur d'événements qui lui a été associé à la création. Cet écouteur peut se contenter de positionner la langue quand le synthétiseur est prêt. Son code est le suivant :

```
private class SynthPret implements TextToSpeech.OnInitListener {
      public void onInit(int status) {
            if (status == TextToSpeech.SUCCESS){
                  parle.setLanguage(Locale.FRANCE); // langue
            }
      }
}
```

Autres réglages :

- `setSpeechRate(float)` permet de régler la vitesse de synthèse (1 normal, <1 plus lent, >1 plus rapide)
- `setPitch(float)` permet de régler la tonalité (1 normal, <1 plus grave, >1 plus aigu)

Pour synthétiser un texte :

```
String message = new String("texte a dire");
parle.speak(message, TextToSpeech.QUEUE_FLUSH, null);
```

Pour ajouter du texte à synthétiser pendant la synthèse :

```
String ajout = new String("texte a ajouter");
parle.speak(ajout, TextToSpeech.QUEUE_ADD, null);
```

Pour arrêter la synthèse de son :

```
parle.stop(); // arrêt de la synthèse
```

Pour arrêter le synthétiseur de son :

```
parle.shutdown(); // arrêt du synthétiseur
```

On doit s'assurer de fermer le synthétiseur à la fin de l'application, ceci peut se faire dans la méthode `onDestroy` de l'activité par :

```
public void onDestroy() {
    if (parle != null) {
        parle.stop();
        parle.shutdown();
    }
    super.onDestroy();
}
```

Internet

```
import java.net.URL
import java.io.InputStreamReader
import java.io.BufferedReader
import java.io.BufferedInputStream
import java.net.HttpURLConnection

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

Récupérer un fichier texte depuis une URL

Ce type d'accès se fait de façon classique par un flux ouvert sur l'URL :

```
try {
    URL url = new URL("mettre ici l'URL du fichier texte");
    BufferedReader in = new BufferedReader(new
                                InputStreamReader(url.openStream()));
    String str;
    while ((str = in.readLine()) != null) {
        // traiter la ligne lue (qui est dans str)
    }
    in.close();
}
catch (MalformedURLException e) {}
catch (IOException e) {
```

Récupérer une image depuis une URL

On peut fabriquer un `Drawable` qui pourra être mis dans un `ImageView` ou un `ImageButton` à partir d'une image prise sur une `URL` par :

```
BitmapDrawable img;
HttpURLConnection connexion = (HttpURLConnection)new URL("mettre ici
        l'url de l'image").openConnection();
connexion.connect();
BufferedInputStream lecture = new BufferedInputStream
                                        (connexion.getInputStream());
BitmapFactory.Options opts = new BitmapFactory.Options();
opts.inSampleSize = x; // pour réduire la taille en 1/x
img = BitmapFactory.decodeStream(lecture, null, opts);
```

Jouer un son depuis une URL

Le procédé est exactement le même que pour un son placé dans le répertoire **res/raw** la seule différence porte sur la construction du `MediaPlayer`. Au lieu de :

`MediaPlayer` lecteur = `MediaPlayer.create`(Context, int)

Il suffit de faire :

```
MediaPlayer mp = new MediaPlayer();
try {
  mp.setDataSource("http://domaine.sous_domaine/chemin/nom_son.mp3");
  mp.prepare();
}
catch (IllegalArgumentException e) {
// Le paramètre de setDataSource est incorrect
}
catch (IllegalStateException e) {
// Le MediaPlayer n'est pas dans l'état initial
}
catch (IOException e) {
```

```
// L'acces à l'URL provoque une erreur
}
```

Jouer une vidéo depuis une URL

Le procédé est exactement le même que pour une vidéo placée dans le répertoire **res/raw** la seule différence porte sur l'Uri utilisée. Au lieu de :

```
Uri chemin =
Uri.parse("android.resource://paquetage_de_l_application/"
                                    +R.raw.nom_du_fichier_video);
```

Il suffit de faire :

```
Uri chemin = Uri.parse("http://domaine.sous_domaine
                                              /rep1/nom_video.3gp");
```

Le widget WebView

WebView permet d'afficher des pages Web comme un navigateur. Elle peut également exécuter du javascript.

```
<WebView
    android:id="@+id/vueweb"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>
```

Méthodes de la classe WebView

android.webkit.WebView

Construction

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

Etats

- `canGoBack()` renvoie true si la vue a un historique permettant de revenir en arrière
- `canGoForward()` renvoie true si la vue a un historique permettant d'aller en avant
- `goBack()` revenir en arrière
- `goForward()` aller en avant
- `clearCache()` efface le cache
- `clearHistory()` efface l'historique
- `getTitle()` renvoie le titre de la page
- `getURL()` renvoie l'URL de la page

Contenu

- `loadData(String data, String typeMime, String codage)` affiche le contenu du premier paramètre interprêté comme étant du type mime indiqué par le seuxième paramètre. Le dernier paramètre indique le type de codage ("UTF-8" …), pour du HTML ce paramètre peut être ***null***.
- `loadURL(String url)` affiche le contenu de l'URL indiquée.
- `pageDown()` descend d'une page
- `pageUp()` monte d'une page
- `reload()` recharge la page
- `stopLoading()` arrête le chargement de la page
- `zoomIn()` zoom en avant de la page
- `zoomOut()` zoom en arrière de la page

Comportement

Le contrôle du comportement de la WebWiew se fait au trvers de l'objet de classe **Websettings** associé à la WebView et obtenu par sa méthode `getSettings()`. Cette classe possède les méthodes suivantes :

- `setJavaScriptEnabled(boolean)` qui autorise ou interdit javascript selon le paramère (true/false).
- `setJavaScriptCanOpenWindowsAutomatically(boolean)` qui autorise ou interdit javascript à ouvrir de nouvelles fenêtres selon le paramère (true/false).
- `setBlockNetworkImage(boolean)` qui autorise ou interdit le chargement des images selon le paramère (true/false).
- `setBlockNetworkLoads(boolean)` qui autorise ou interdit le chargement de pages selon le paramère (true/false).
- `setGeolocationEnabled(boolean)` qui autorise ou interdit la géolocalisation selon le paramère (true/false).

Fragments

Les fragments permettent de définir des zones dans l'interface qui seront gérées par des sous activités. Une zone est en fait un conteneur dans lequel la sous activité placera sa propre interface.

Les fragments peuvent être gérés de façon statique ou dynamique. La gestion statique se contente de définir des fragments dont la sous activité associée sera lancée dès que la zone correspondante sera affichée. La gestion dynamique permet à l'activité d'ajouter, d'enlever et de remplacer des fragments en cours de fonctionnement. Ainsi, par exemple, un fragment apparaîtra lorsque l'utilisateur cliquera sur un bouton et sera remplacé par un autre s'il clique sur un autre bouton. Cette méthode permet d'optimiser l'affichage disponible en ne faisant apparaître que ce qui est utile à un moment donné. De plus, selon la taille de l'écran (téléphone ou tablette par exemple) l'application choisira de laisser certains fragments visibles ou pas.

Ajouter un fragment à l'interface

Le fragment est placé dans le fichier XML de l'interface comme n'importe quel widget :

```
<fragment android:name =  ".MonFragment"
        android:id = "@+id/nom_du_fragment"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        … />
```

Le parametre *android:name* définit le nom de la classe de la sous activité gérant le fragment.

Contenu d'un fragment

Il est décrit par un fichier XML classique comme toute interface.

Sous activité de gestion du fragment

C'est une classe qui hérite de *Fragment* et non de *Activity*.

Elle met en place l'interface et en gère les événements. Pour ce faire il faut surcharger les méthodes de son cycle de vie. La création correspond à la méthode *onCreateView* et non à *onCreate* c'est là que l'interface est créée à partir du fichie XML. Cette méthode renvoie l'interface créée :

```
public View onCreateView(LayoutInflater decode, ViewGroup conteneur, Bundle sauve) {
   // Mise en place de l'interface du fragment
          View contenu = decodeur.inflate(R.layout.fichr_xml_du_fragment, conteneur, false);
   return contenu; // Renvoie le fragment créé
}
```

La méthode onViewCreated est appelée lorsque l'interface a été créée, elle permet de récupérer les widgets par findViewById :

```
public void onViewCreated(View vue, Bundle sauve) {
  super.onViewCreated(vue, sauve);
```

```
        // Récupération des widgets
        titre = (TextView) vue.findViewById(R.id.titre);
    }
```

La classe Fragment

Les principales méthodes de Fragment sont :

Environnement du fragment

- `Activity getActivity()` renvoie l'activité principale (peut être null si elle n'est pas encore créée, voir plus loin la surcharge de onAttach)
- `Context getContext()` renvoie le contexte du fragment (pas celui de l'activité principale)
- `FragmentManager getFragmentManager()` renvoie le gestionnaire de Fragments
- `LayoutInflater getLayoutInflater()` renvoie l'objet permettant la lecture du fichier XML d'interface et sa création
- `Resources getResources()` accès aux ressources comme pour une activité classique

Etat du fragment

- `isAdded()` true si le Fragment a été ajouté à l'interface
- `isInLayout()` true si le Fragment est dans l'interface
- `isDetached()` true si le Fragment est détaché de l'interface
- `isHidden()` true si le Fragment est invisible
- `isVisible()` true si le Fragment est visible

Cycle de vie du Fragment

- `onActivityCreated(Bundle savedInstanceState)` lorsque l'activité principale est créée
- `onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)` lorsque le Fragment doit être créé
- `onViewCreated(View view, Bundle savedInstanceState)` lorsque le Fragment est créé
- `onAttach(Context context)` lorsque le fragment est attaché à l'activité
- `onDetach()` lorsque le fragment est détaché à l'activité
- `onPause()` comme pour une activité classique
- `onResume()` comme pour une activité classique
- `onStart()` comme pour une activité classique
- `onStop()` comme pour une activité classique
- `onDestroy()` comme pour une activité classique

Gestion des menus du Fragment

Ces méthodes fonctionnent comme celles d'une activité classique.

- `onCreateOptionsMenu(Menu menu, MenuInflater inflater)`
- `onContextItemSelected(MenuItem item)`
- `onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)`
- `onOptionsItemSelected(MenuItem item)`
- `registerForContextMenu(View view)`
- `unregisterForContextMenu(View view)`

Gestion des sous activités du Fragment

Ces méthodes fonctionnent comme celles d'une activité classique.

- `startActivity(Intent intent)`
- `startActivityForResult(Intent intent, int requestCode)`
- `onActivityResult(int requestCode, int resultCode, Intent data)`

Activité contenent des fragments

L'activité contenant des Fragments doit hériter de ***FragmentActivity*** ou, si l'on souhaite conserver la compatibilité avec les versions antérieures d'Android, de ***AppCompatActivity***.

Elle peut récupérer un Fragment créé dans son interface par son identifiant comme elle le fait pour récupérer ses widgets :

```
MonFragment monFragment = (MonFragment) getSupportFragmentManager().
                                        findFragmentById(R.id.monfragment);
```

Communication entre activité et fragments

L'activité peut vouloir accéder au Fragment (par exemple pour y faire afficher quelque chose). Ceci peut être réalisé en utilisant une méthode du Fragment.

Dans la classe du Fragment on ajoute une méthode : void affiche(String t) qui affiche le texte passé en paramètre dans un widget du Fragment.
Dans l'activité il suffit de faire :

```
MonFragment monFragment = (MonFragment) getSupportFragmentManager().
                                        findFragmentById(R.id.monfragment);
monFragment.affiche(message);
```

De même une action déclenchée par un widget peut vouloir accéder à l'activité par exemple pour y faire afficher quelque chose. Pour cela il faut que l'activité propose une méthode : void trace(String t) qui affiche le texte passé en paramètre dans un widget de l'interface.

Afin que le Fragment sache que cette méthode existe on utilisera une interface qui la définit :

```
public interface ITrace {
   public void trace(String t) ;
}
```

L'activité implémente cette interface ITrace et définit la méthode trace :

```
class MonActivite extends AppCompatActivity implements ITrace {
        public void onCreate(Bundle sauve) {
            …
        }
        …
        public void trace(String t) {
            …
        }
}
```

Pour que le Fragment accède à cette méthode il faut :

- Déclarer une propriété dans le Fragment : private ITrace activite ;
- Récupérer l'activité dans onAttach(Context contexte) par :
  activite = (ITrace) contexte ;

Dans le Fragement il suffit de faire : activite.trace(message);

Gestion dynamique des fragments

Les Fragments peuvent être ajoutés, enlevés ou remplacés dans l'interface dynmiquement. Afin d'éviter l'apparition d'états intyermédiaires de l'interface pendant ces changements ils sont inclus dans une transaction.
Les transactions sont gérées pat un objet de classe **FragmentTransaction** obtenu après du **FragmentManager** par :

```
FragmentTransaction transaction = getFragmentManager().beginTransaction();
```

L'objet FragmentTransaction ainsi obtenu offre les méthodes :

- `transaction.add(R.id.nomDuContenant, fragment);`
- `transaction.remove(fragment);`
- `transaction.replace(R.id.nomDuContenant, nouveauFragment);`

- `transaction.hide(fragment);`
- `transaction.show(fragment);`
- `transaction.attach(fragment);`
- `transaction.detach(fragment);`

Lorsque toutes les opérations sur les fragments ont été faites, la transaction est fermée (et donc exécutée) par :
- `transaction.commit();`

Défilement de Fragments (ViewPager)

Le ViewPager permet d'organiser des Fragments comme des pages d'un livre et de passer de l'un à l'autre.
Il est défini dans l'interface comme un conteneur :

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pageapage"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Le changement de page est géré par un objet de classe FragmentStatePagerAdapter associé à ce ViewPager dans l'activité :

```
private ViewPager pages;
private PagerAdapter gestionPages;

protected void onCreate(Bundle savedInstanceState)
      super.onCreate(savedInstanceState);
      setContentView(R.layout.page_activite);
      …
      pages = (ViewPager) findViewById(R.id. pageapage);
      gestionPages = new ChangePage (getSupportFragmentManager());
      pages.setAdapter(gestionPages);
}
```

Le changement de page est assuré par une classe spécifique héritant de FragmentStatePagerAdapter :

```
private class ChangePage extends FragmentStatePagerAdapter {

        public ChangePage (FragmentManager fm) {
            super(fm);
        }

        public Fragment getItem(int position) {
            // renvoie le fragment correspondant à la position
        }

        public int getCount() {
            return NUM_PAGES; // nombre de pages
        }
}
```

La méthode getItem doit récupérer le Fragment correspondant à la position demandée. Les Fragments ont été crées auparavant (par exemple dans le onCreate de l'activité) et placés dans une collection (Vector).
La méthode getCount se contente d'indiquer combien de pages (donc de Fragments) propose le ViewPager.

Le défilement des pages est géré par le Viewpager. Toutefois on peut également prendre en compte le bouton « Back » du périphérique pour revenir en arrière dans le parcours. Pour cela il suffit de surcharger la méthode onBackPressed() de l'activité :

```
public void onBackPressed() {
    if (pages.getCurrentItem() == 0) {
            super.onBackPressed();
    }
```

```
        else {
                pages.setCurrentItem(pages.getCurrentItem() - 1);
        }
    }
```

Remarque : Un Viewpager peut être lié à un TabLayout pour disposer d'onglets.

Utiliser des services WEB

Pour accéder aux service WEB, l'accès à internet doit être autorisé par l'ajout dans le fichier **AndroidManifest** d'une balise :

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

Trouver les coordonnées géographiques de lieux

Android offre une classe `Geocoder` permettant de trouver les coordonnées d'un lieu décrit par une adresse ou une adresse à partir de ses coordonnées géographiques.

La classe Geocoder

```
import android.location.Geocoder
import java.util.Locale
import java.util.List
import android.location.Address
```

Construction

- `Geocoder(Context, Locale)` le premier paramètre est le contexte de l'activité que l'on peut, par exemple, obtenir par `getApplicationContext()`. Le second paramètre indique la zone géographique concernée, il peut prendre les valeurs (`Locale.FRANCE`, `Locale.CANADA`, `Locale.UK`, `Locale.US` …). Il peut être omis si l'on ne souhaite pas limiter la localisation.

Recherches

Toutes ces méthodes renvoient une liste (classe `List` de java) contenant des objets de classe `Address` (ATTENTION : cette liste peut être vide si la recherche ne donne aucun résultat) :

- `getFromLocation`(double, double, int) renvoie les adresses connues proches du point défini par ses coordonnées géographiques passées par les 2 premier paramètres (latitude et longitude exprimées en degrés). Le dernier paramètre permet de limiter la taille de cette liste.
- `getFromLocationName`(String, int) renvoie les adresses connues proches du point défini par le nom passé en premier paramètre (c'est une chaîne du type "3 Allée du parc Montaury, 64600, Anglet"). Le second paramètre permet de limiter la taille de cette liste.
- `getFromLocationName`(String, int, double, double, double, double) cette méthode fonctionne comme la précédente mais permet de limiter la zone de recherche à un rectangle. Les 3ème et 4ème paramètres indiquent la longitude et la latitude du coin inférieur gauche de ce rectangle tandis que les 5ème et 6ème paramètres en indiquent la longitude et la latitude du coin supérieur droit.

La classe Address

```
import android.location.Address
```

Un objet de classe Address correspond à une localisation géographique et contient des chaînes de caractères correspondant aux lignes d'adresse de cette localisation (voir exemple ci-dessous).

Construction

- `Address(Locale)` le paramètre indique la zone géographique concernée, il peut prendre les valeurs (`Locale.FRANCE`, `Locale.CANADA`, `Locale.UK`, `Locale.US` …). Ce dernier paramètre peut être omis si l'on ne souhaite pas limiter la localisation.

Contenu

- `getLatitude()` renvoie la latitude en degrés (réel type double)

- `getLongitude()` renvoie la longitude en degrés (réel type double)
- `getFeatureName()` renvoie le nom du lieu
- `getLocality()` renvoie le nom de la ville
- `getPostalCode()` renvoie le code postal
- `getCountryName()` renvoie le nom du pays
- `getAddressLine(int)` renvoie la ligne d'adresse désignée par son index passé en paramètre (en commençant à 0). Renvoie **null** s'il n'y a rien correspondant à l'index. On peut connaître le nombre de lignes d'adresse disponibles par la méthode : `getMaxAddressLineIndex()`

Exemple :

Le code suivant :

```
Geocoder localisateur = new Geocoder(getApplicationContext(), Locale.FRANCE);
List<Address> liste = localisateur.getFromLocationName("Parc Montaury, Anglet", 10);
```

Renvoie une liste ne contenant qu'un seul objet de classe `Address` dont le contenu est le suivant :

- latitude : 43,4800424
- longitude : -1,5093202
- nom de lieu : Allée du Parc Montaury
- nom de ville : Anglet
- code postal : 64600
- nom de pays : France
- Deux lignes d'adresse qui sont :
  - Allée du Parc Montaury
  - 64600 Anglet

GoogleMaps

Préparation du projet

Pour pouvoir utiliser GoogleMaps il faut que le projet intégre la bibliothèque Google Maps : com.google.android.gms:play-services-maps:9.0.1

Le widget d'affichage de la carte doit être placé dans un Fragment (voir ci-dessus).
L'activité doit donc hériter de `FragmentActivity` ou de `AppCompatActivity`.

Dans onCreate l'activité récupère le fragment et le connecte à la carte en lui associant un écouteur :

```
MapFragment monFragment = (MapFragment) getFragmentManager()
                                                .findFragmentById(R.id.nomDeMonFragment);
monFragment.getMapAsync(new Ecouteur());
```

L'écouteur hérite de OnMapReadyCallback et surcharge la méthode

```
public void onMapReady(GoogleMap googleMap)
```

qui sera appelée quand la carte sera disponible

Clé d'utilisation

L'accès à Google maps nécessite une clé que l'on obtient comme suit :

- Avoir un compte Google ou en créer un en allant sur **https://www.google.com/accounts/**
- Obtenir l'empreinte md5 du certificat de l'application. Pour ce faire :
  - Ouvrir une fenêtre DOS et se mettre dans le répertoire du compilateur java (par exemple : C:\Program Files\Java\jdk1.6.0_07\bin)
  - Taper la commande :
    keytool –list –alias androiddebugkey –keystore *chemin* **–storepass android –keypass android**
    dans laquelle *chemin* désigne le fichier **debug.keystore** qui se trouve dans le répertoire **.android** de l'utilisateur
    Cette commande affiche un certificat MD5 de la forme :
    androiddebugkey, 2 oct. 2010, PrivateKeyEntry, Empreinte du certificat (MD5) :
    CA:78:06:9E:1A:27:20:4D:28:78:6B:F3:A5:D2:BF:78
- Obtenir la clé en allant sur la page :

http://code.google.com/intl/fr/android/maps-api-signup.html
Le code précédent est saisi dans "My certificate's Fingerprint" et on clique sur le bouton "Generate API Key". Après s'être authentifié on récupère une clé qui est une chaîne de caractères (faites un copier/coller dans un fichier texte pour ne pas la perdre).

Cette clé fonctionnera pour toutes vos applications. Elle doit être mise dans le fichier manifest de l'application sous la forme :

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="mettre ici la clé"/>
```

De plus ce fichier doit contenir les permissions :

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Et autoriser l'utilisation de OpenGL :

```
<uses-feature android:glEsVersion="0x00020000" android:required="true"/>
```

Afficher des cartes dans une application

Les cartes sont affichées dans un widget `MapView` placé dans un fragment

La classe MapView

Cette classe hérite de `ViewGroup`. Elle a donc les mêmes propriétés que `ViewGroup`

Exemple de fichier XML :

```
<com.google.android.maps.MapView android:id="@+id/carte"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"/>
```

Méthodes de la classe MapView

import com.google.android.maps.MapView

Apparence

- `displayZoomControls(boolean)` affiche le contrôle de zoom. Le paramètre permet de donner le focus à ce contrôle (true)
- `setBuiltInZoomControls(boolean)` autorise (true) ou invalide (false) le contrôle de zoom
- `setSatellite(boolean)` affichage en mode satellite ou plan
- `setStreetView(boolean)` affichage avec ou sans visualisation des rues
- `setTraffic(boolean)` affichage avec ou sans visualisation du traffic

Contenu

- `getLatitudeSpan()` renvoie l'angle de latitude visible (entre le haut et le bas de la carte affichée) en millionièmes de degrés
- `getLongitudeSpan()` renvoie l'angle de longitude visible (entre la gauche et la droite de la carte affichée) en millionièmes de degrés
- `getMapCenter()` renvoie un objet de classe `GeoPoint` indiquant le centre de la carte. Cet objet possède les méthodes `getLatitudeE6`() et `getLongitudeE6`() qui renvoient ses coordonnées géographiques en millionièmes de degrés.
- `getMaxZoomLevel()` renvoie un entier indiquant la valeur maximale de zoom possible
- `getZoomLevel()` renvoie un entier indiquant la valeur actuelle de zoom
- `isSatellite()` renvoie un booléen indiquant si la carte est en mode satellite (true)
- `isStreetView()` renvoie un booléen indiquant si la carte est en mode rues (true)
- `isTraffic()` renvoie un booléen indiquant si la carte est en mode trafic (true)

Projection

Lorsqu'une carte est affichée il est possible de récupérer les coordonnées géographiques d'un point par rapport à sa position à l'écran (par exemple lorsque l'utilisateur touche la carte). Ceci peut se faire par la méthode `fromPixels(int, int)` de l'objet de classe `Projection` renvoyé par la méthode `getProjection()` de l'objet de classe `MapView` affiché. Cette méthode prend en paramètres les coordonnées (x et y) d'un pixel à l'écran et renvoie un objet de classe `GeoPoint` correspondant à la position. . Cet objet possède les méthodes `getLatitudeE6`() et `getLongitudeE6`() qui renvoient ses coordonnées géographiques en millionièmes de degrés.

La classe MapController

Cette classe sert à piloter la carte affichée. On l'obtient par la méthode `getController`() de l'objet `MapView`.

Méthodes de la classe MapController

import com.google.android.maps.MapController

Déplacements

- `animateTo`(`GeoPoint`) positionne la carte sur le point passé en paramètre en faisant une animation. La classe `GeoPoint` possède un constructeur acceptant en paramètres ses coordonnées géographiques en millionièmes de degrés; le 1er paramètre est la latitude, le second la longitude.
- `setCenter(GeoPoint)` positionne la carte sur le point passé en paramètre sans faire d'animation. La classe `GeoPoint` possède un constructeur acceptant en paramètres ses coordonnées géographiques en millionièmes de degrés
- `scrollBy(int , int)` déplace la carte. Le premier paramètre exprime le déplacement horizontal (en pixels), le second le déplacement vertical.
- `setZoom(int)` définit le niveau de zoom
- `zoomIn()` zoome d'un niveau
- `zoomOut()` dé-zoome d'un niveau