



Développement d'applications pour Android



M. Dalmau – IUT de Bayonne – Pays Basque

Positionnement d'Android



L'iPhone (Apple)



Nokia (Symbian OS)



Window Mobile (Microsoft)



BlackBerry

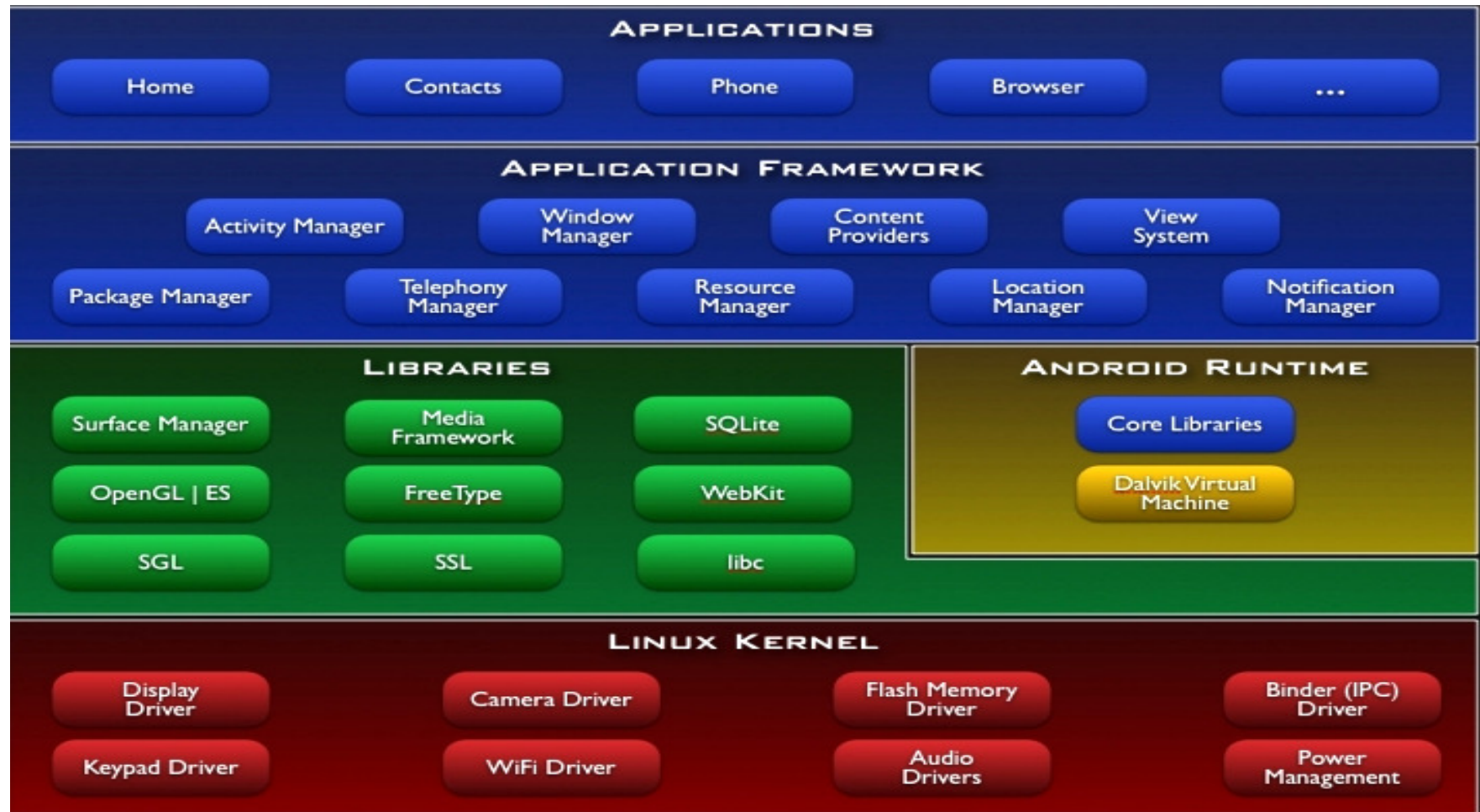


Palm (WebOS)



Google (Android)

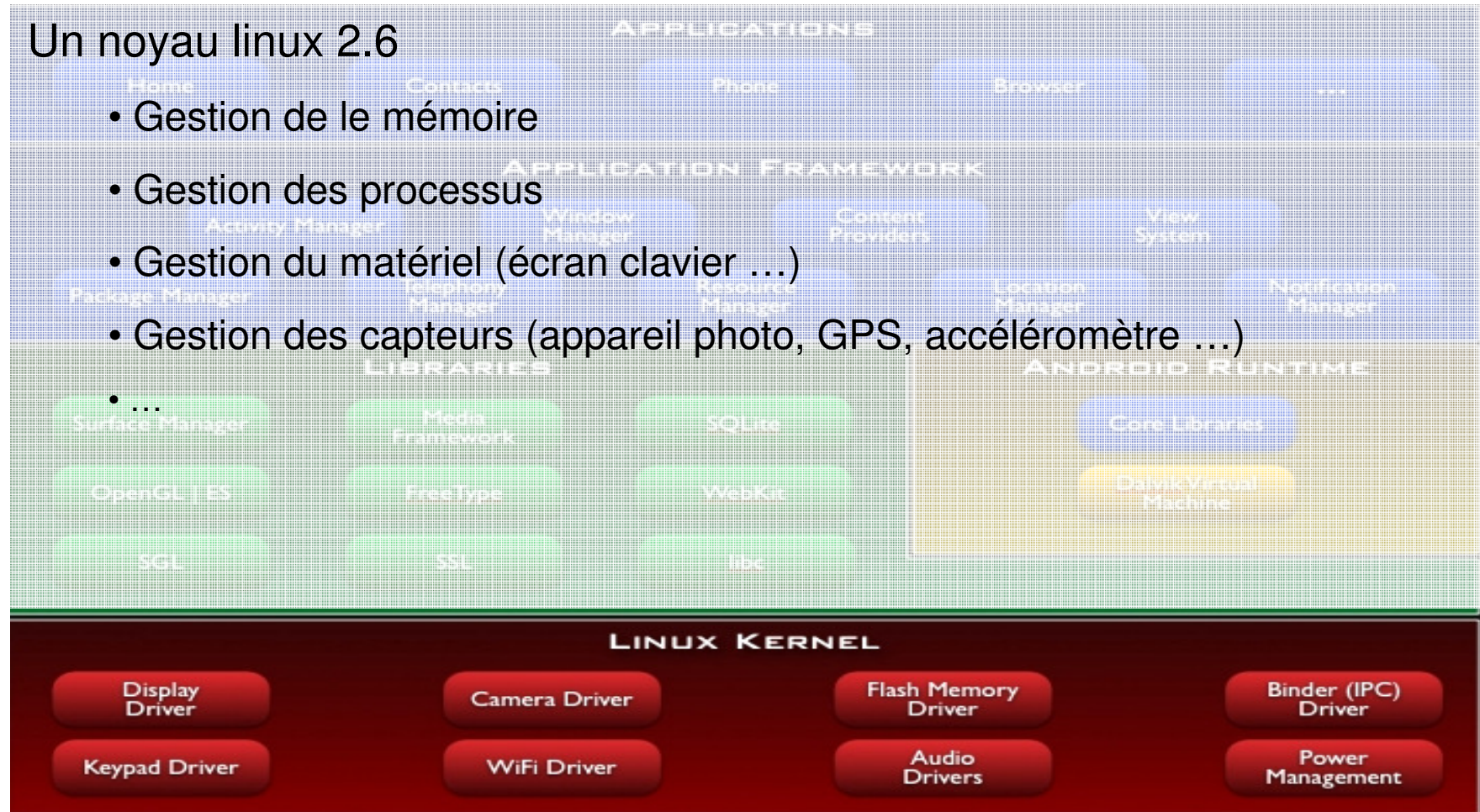
Architecture d'Android



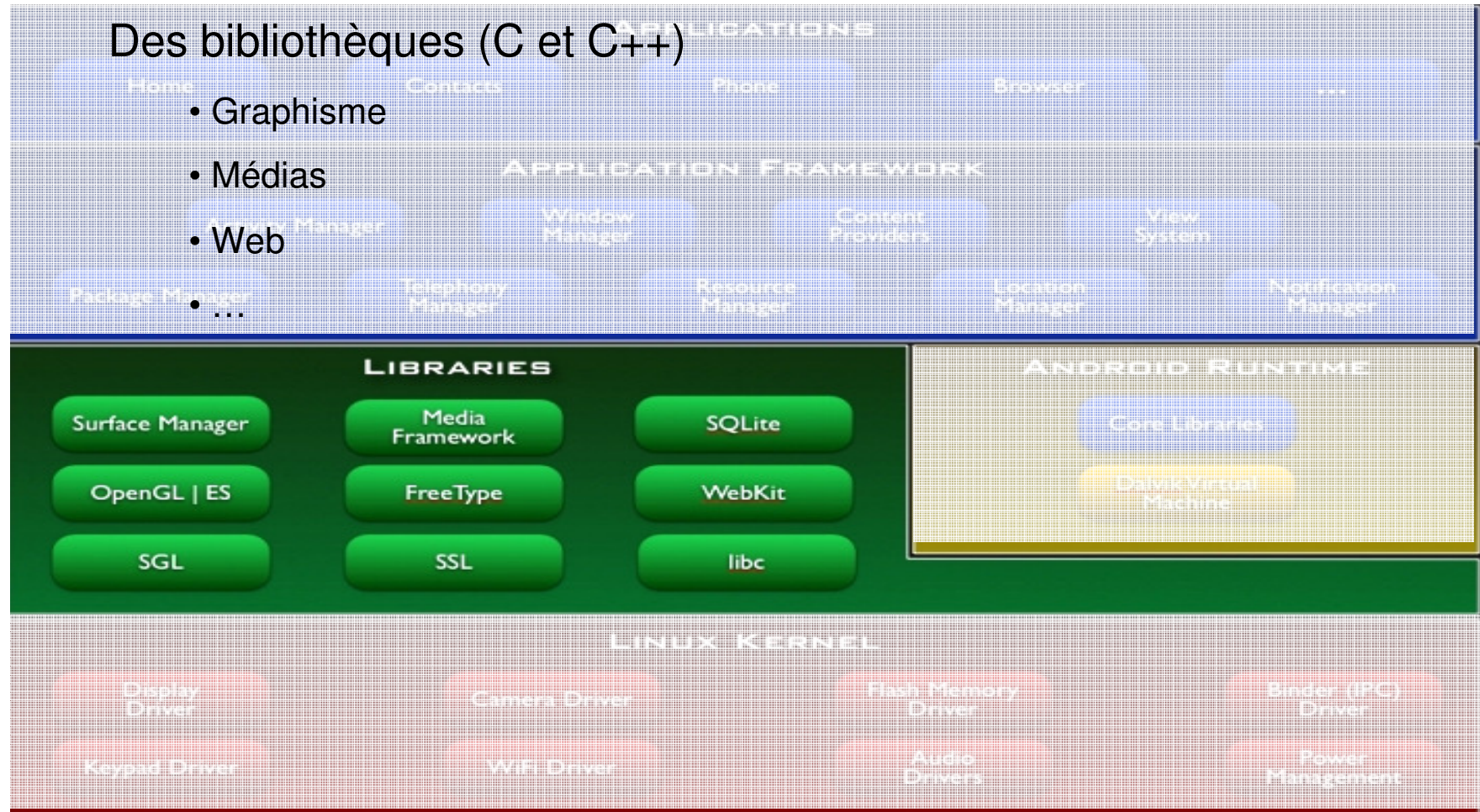
Architecture d'Android

Un noyau linux 2.6

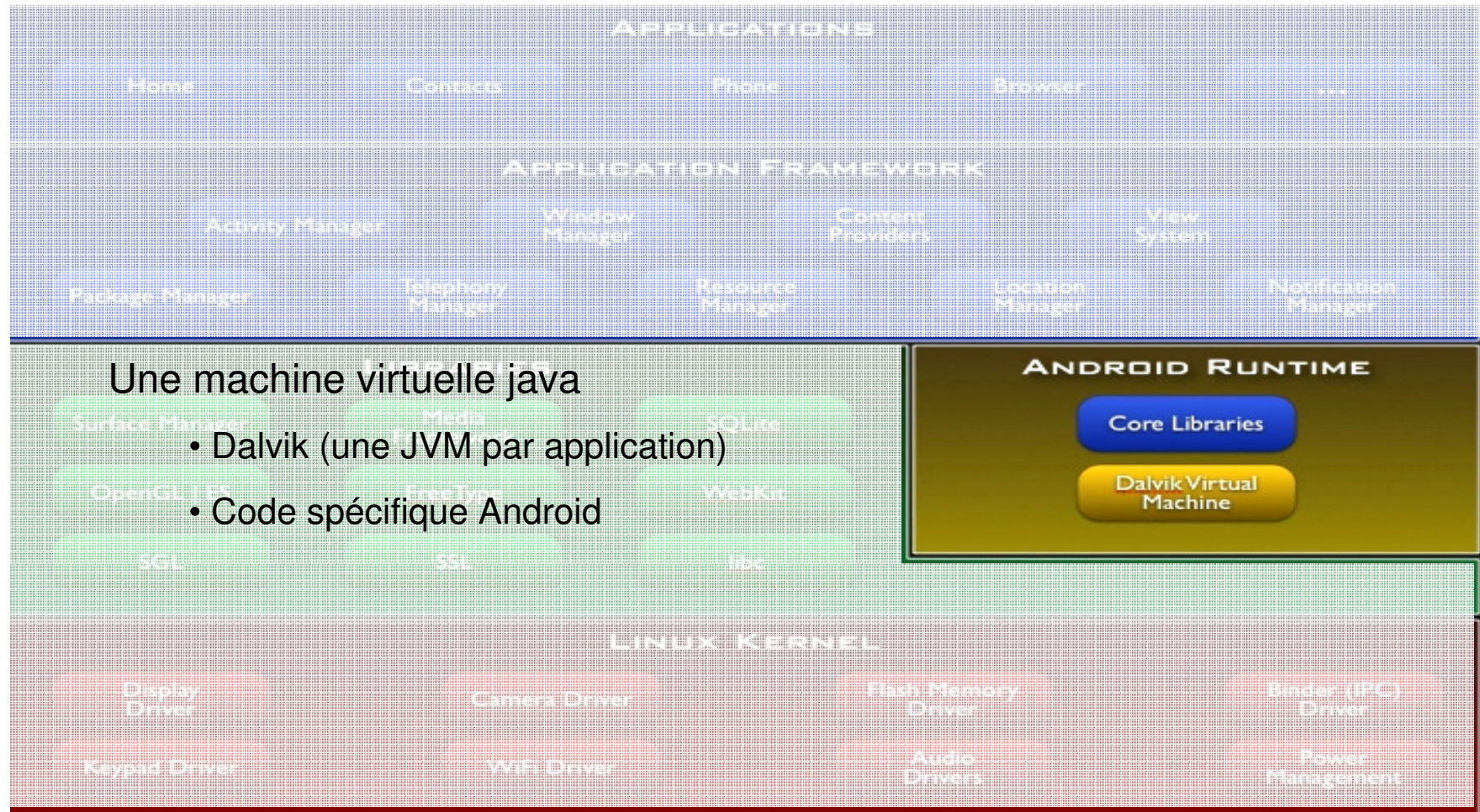
- Gestion de la mémoire
- Gestion des processus
- Gestion du matériel (écran clavier ...)
- Gestion des capteurs (appareil photo, GPS, accéléromètre ...)



Architecture d'Android



Architecture d'Android



Architecture d'Android

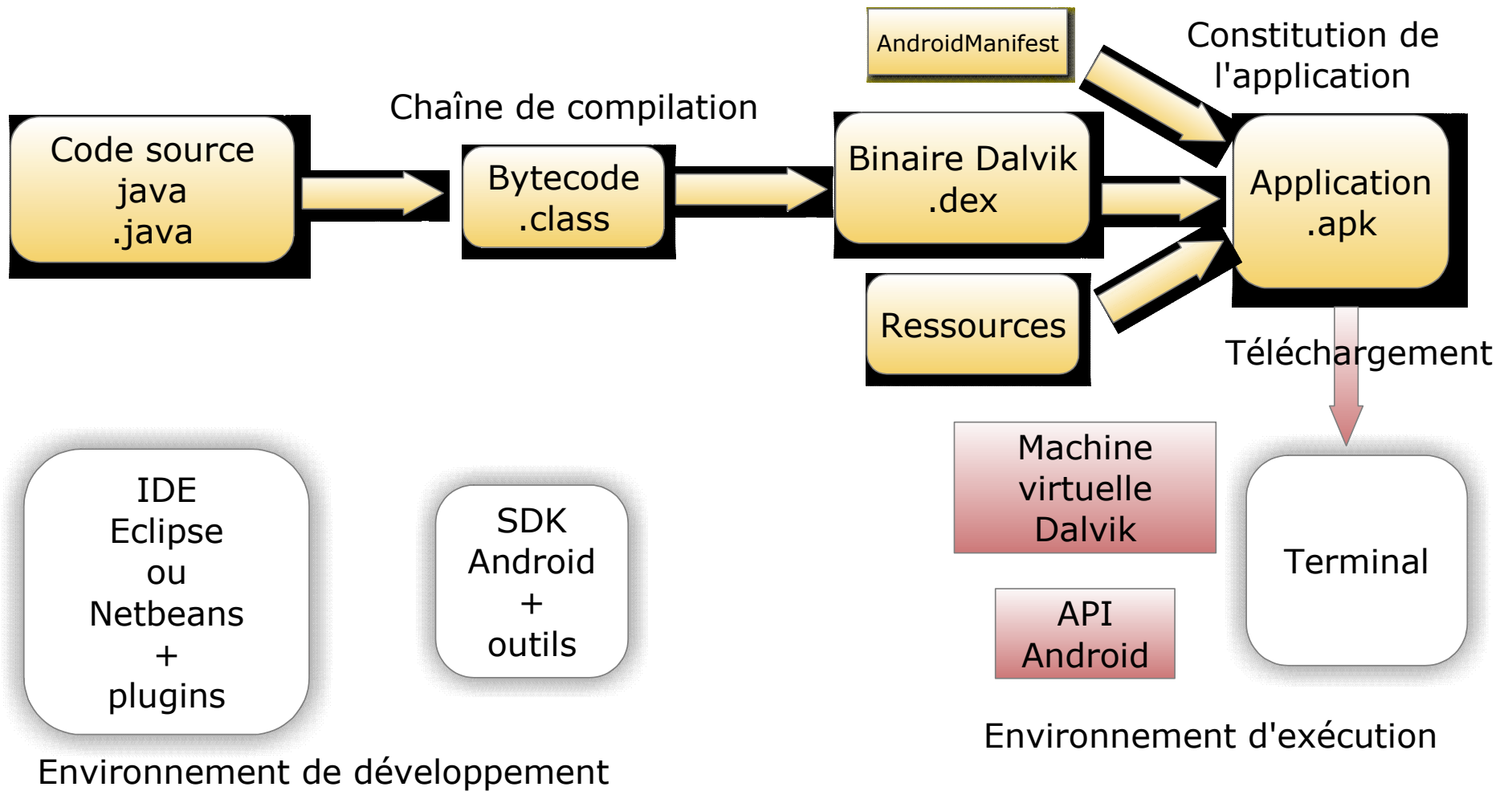


Architecture d'Android



Développement d'applications pour Android

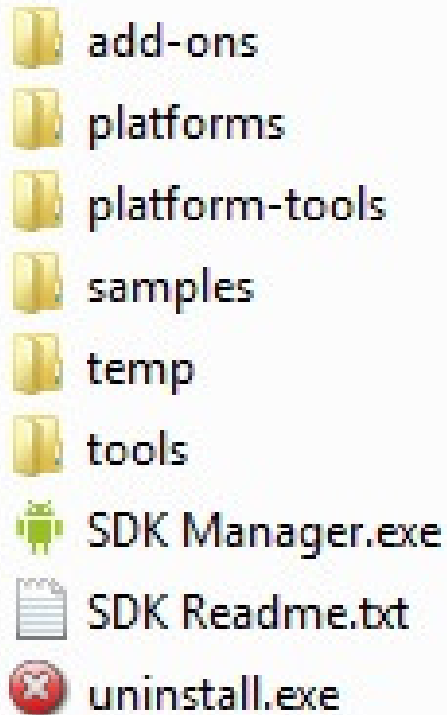
Production de logiciel



Développer pour Android

- Les interfaces et les constantes sont définies dans des fichiers XML
 - Facilite la modification
 - Statique
- Les ressources sont téléchargées avec l'application
- Les fonctionnalités sont dans le code
 - Lien avec ce qui est défini en XML
 - Accès aux ressources
- L'API n'est pas totalement celle de java (classes redéfinies (par exemple Color), interfaces, écouteurs ...)
- La syntaxe des fichiers XML est extensible ⇒ difficile de savoir ce qui est prédéfini
- Les propriétés définies en XML peuvent être contradictoires
- L'interface ne peut être utilisée que par l'activité qui l'a créée
- Difficile de développer sans un environnement adéquat (Eclipse ou Netbeans) en raison des fichiers générés
- La pré-visualisation des interfaces par Eclipse n'est pas toujours conforme (ascenseurs, contenu défini dans le code ...)

Le SDK Android



- Téléchargeable sur :
developer.android.com/sdk
sous la forme d'un zip ou d'un fichier d'installation
- Propose le **SDKManager** qui permet de télécharger les plateformes et outils :
 - Android versions xx
 - Google API versions xx
 - Outils (tools et platform-tools)
 - ...

Quelques outils du SDK Android

Accessibles à partir d'une ligne de commande (fenêtre DOS)

- **adb** permet la connexion au terminal (smartphone ou simulateur) pour :
 - Transférer des fichiers (push / pull)
 - Travailler en ligne de commande unix (shell)
 - Installer une application (install)
 - Paramétrer le réseau (forward)
 - Déboguer une application (logcat)
- **dx** transforme le bytecode java en code Dalvik
- **apkbuilder** constitue un fichier .apk téléchargeable sur le terminal

Remarque : Eclipse utilise ces outils directement mais on est parfois obligé d'y recourir (transfert de fichiers, installations directes d'applications ...)

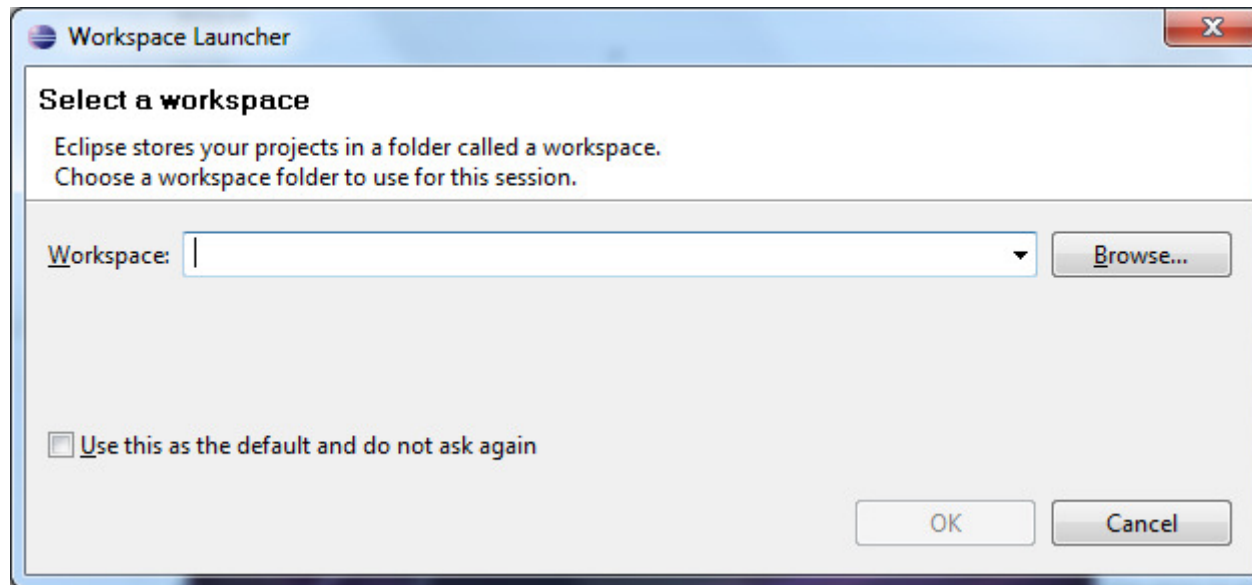
Eclipse



- Téléchargeable sur www.eclipse.org
- Ne nécessite pas d'installation (simple copie)
- Le plugin pour Android est téléchargé et installé depuis Eclipse (Menu Help → Install New Software) sur <https://dl-ssl.google.com/android/eclipse>
- Il suffit ensuite d'indiquer à Eclipse où se trouve le SDK Android (Menu Windows → Preferences → Android)

Développer avec Eclipse

Espace de travail



Remarque : les paramètres d'Eclipse sont liés à l'espace de travail
⇒ nouvel espace nouveaux paramètres

Développer avec Eclipse

New Android Project

Project name must be specified

Project name:

Contents

☒ Create new project in workspace

☐ Create project from existing source

☒ Use default location

Location:

☐ Create project from existing sample

Samples:

Build Target

| Target Name | Vendor | Platform | API Level |
|----------------------------------------------|-----------------------------|-------------|-----------|
| <input type="checkbox"/> Android 2.1-update1 | Android Open Source Project | 2.1-update1 | 7 |
| <input type="checkbox"/> Google APIs | Google Inc. | 2.1-update1 | 7 |

Properties

Application name:

Package name:

☒ Create Activity:

Min SDK Version:

Working sets

☐ Add project to working sets

Working sets:

Projet Android

(Menu File → New →
Android Project)

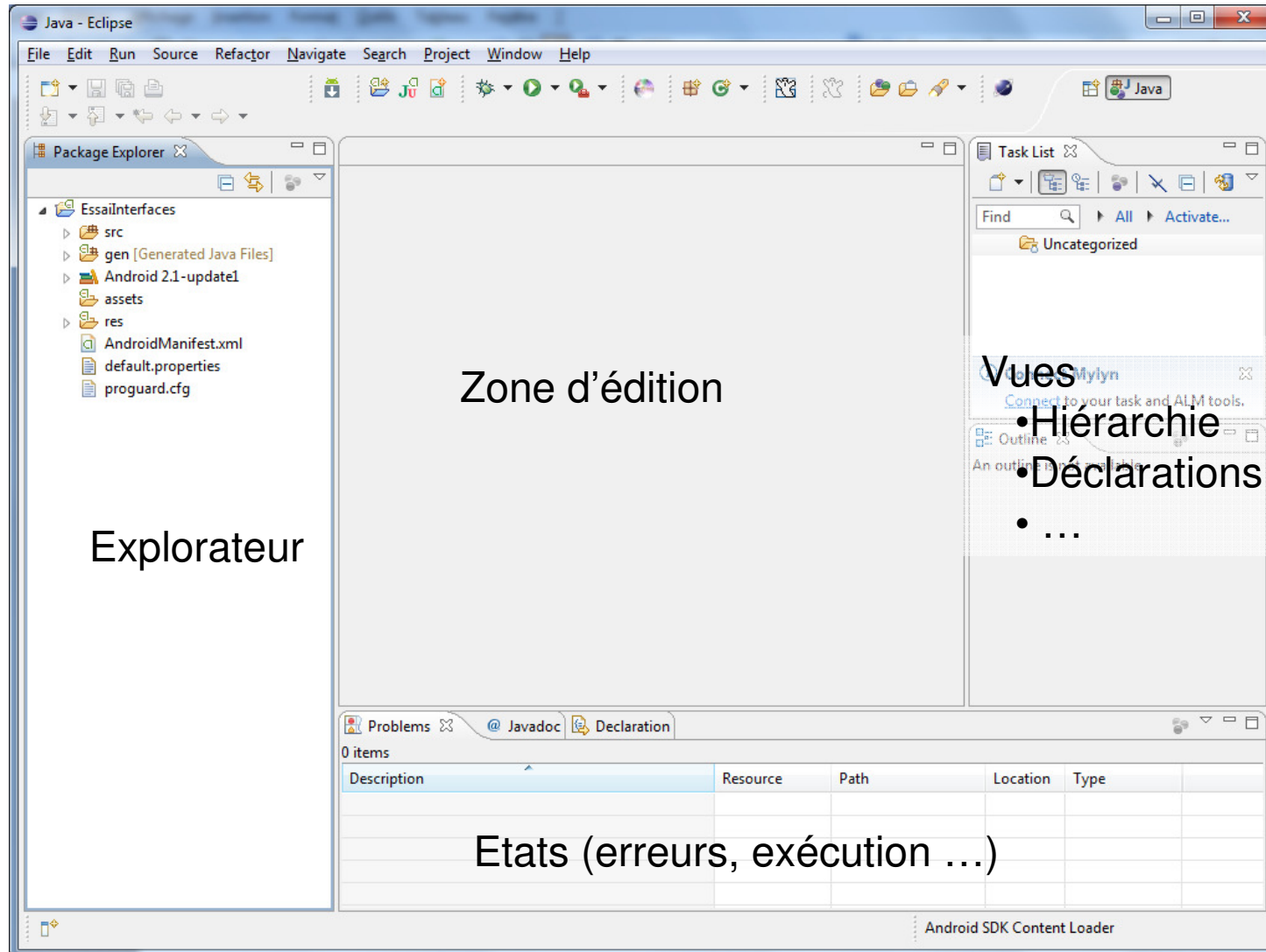
Nom

SDK

Paquetage

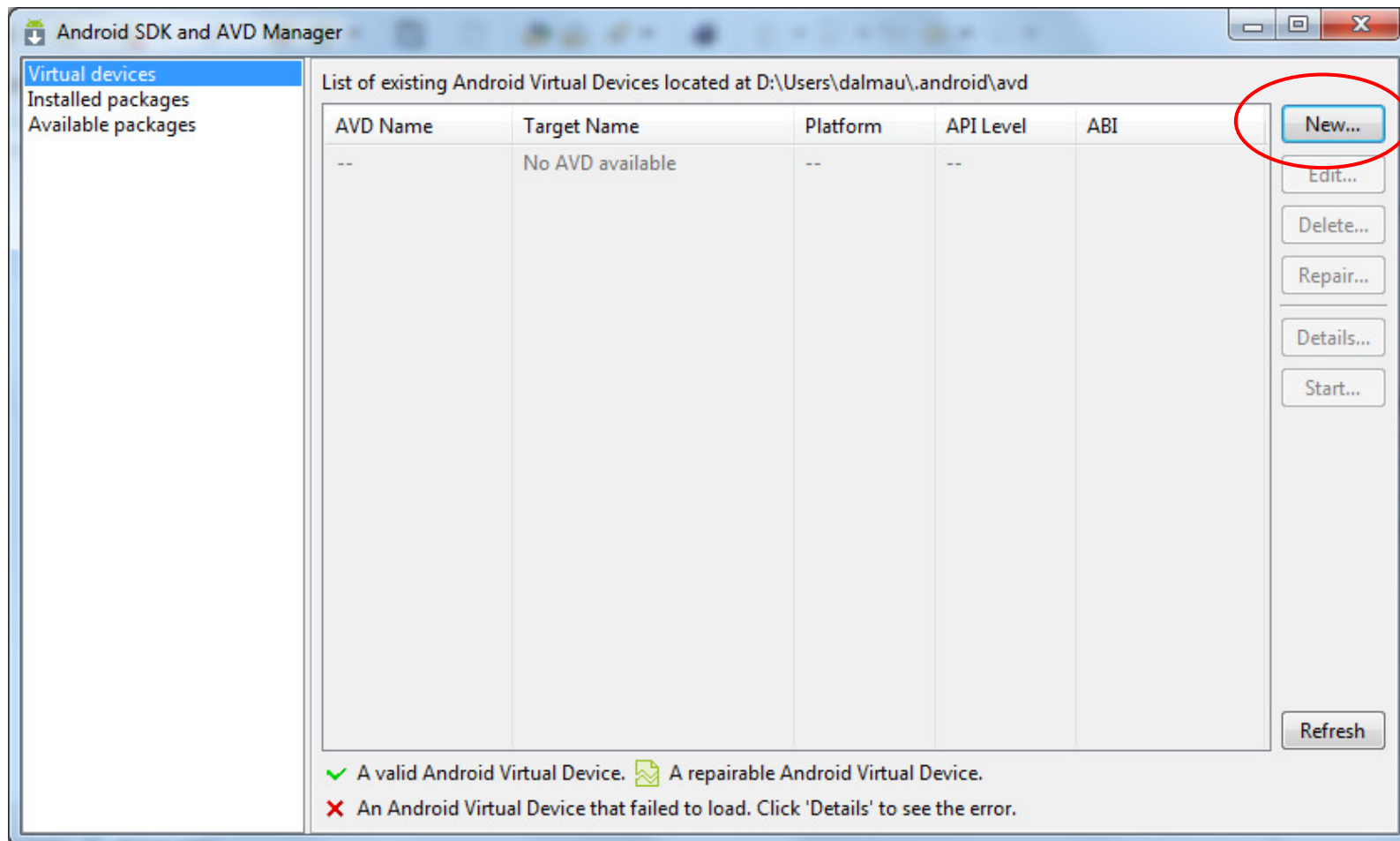
(au moins 2 niveaux)

Développer avec Eclipse



Créer un terminal virtuel (simulateur)

(Menu Windows → Android SDK and AVD Manager)



Créer un terminal virtuel (simulateur)

Create new Android Virtual Device (AVD)

Name:

Target:

ABI:

SD Card:

☒ Size:

☐ File:

Snapshot:

☐ Enabled

Skin:

☒ Built-in:

☐ Resolution: x

Hardware:

| Property | Value | |
|---------------------------|-------|---------------------------------------|
| Accelerometer | yes | <input type="button" value="New..."/> |
| Abstracted LCD density | 240 | <input type="button" value="Delete"/> |
| Max VM application hea... | 24 | |
| Camera support | no | |

☐ Override the existing AVD with the same name

Paramétrage:

Nom

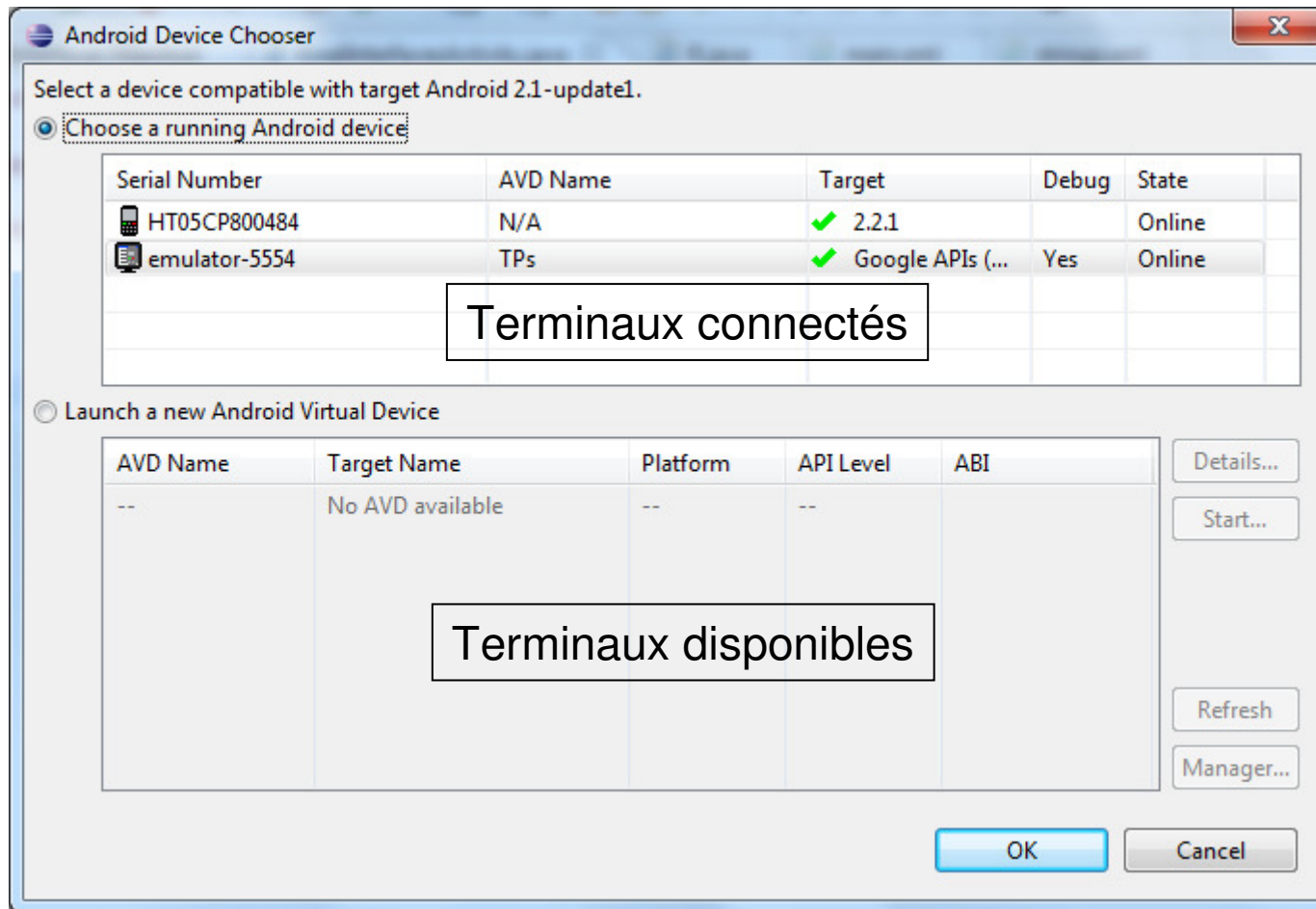
SDK

Taille carte SD

Format d'écran

Matériel

Terminal virtuel ou smartphone



Le terminal virtuel



Contrôle du terminal (DDMS)

DDMS - EssaïInterfaces/src/ut/tp1/EssaïInterfacesActivity.java - Eclipse

File Edit Run Source Navigate Search Project Refactor Window Help

DDMS Java

Devices

| Name | State | Version |
|------------------------|--------|---------------|
| HT05CP800484 | Online | 2.2.1 |
| com.google.android.app | 379 | 8600 |
| emulator-5554 | Online | TPs [2.1, ... |
| system_process | 64 | 8602 |
| com.android.phone | 119 | 8604 |
| android.process.acore | 143 | 8606 |
| com.android.alarmclock | 168 | 8601 |
| android.process.media | 184 | 8608 |
| com.svox.pico | 210 | 8610 |
| com.android.email | 228 | 8612 |
| com.android.inputmethc | 242 | 8603 |
| com.google.android.app | 249 | 8613 |
| com.android.mms | 260 | 8614 |
| com.android.settings | 286 | 8615 |
| iut.tp1 | 339 | 8611 |

Processus

File Explorer

| Name | Size | Date | Time | Permissions | Info |
|--------|------|------------|-------|-------------|------|
| data | | 2011-07-12 | 09:50 | drwxrwx--x | |
| sdcard | | 2011-07-12 | 09:50 | d---rwxr-x | |
| system | | 2009-12-19 | 01:17 | drwxr-xr-x | |

Etats (mémoire, fichiers ...)

Console LogCat

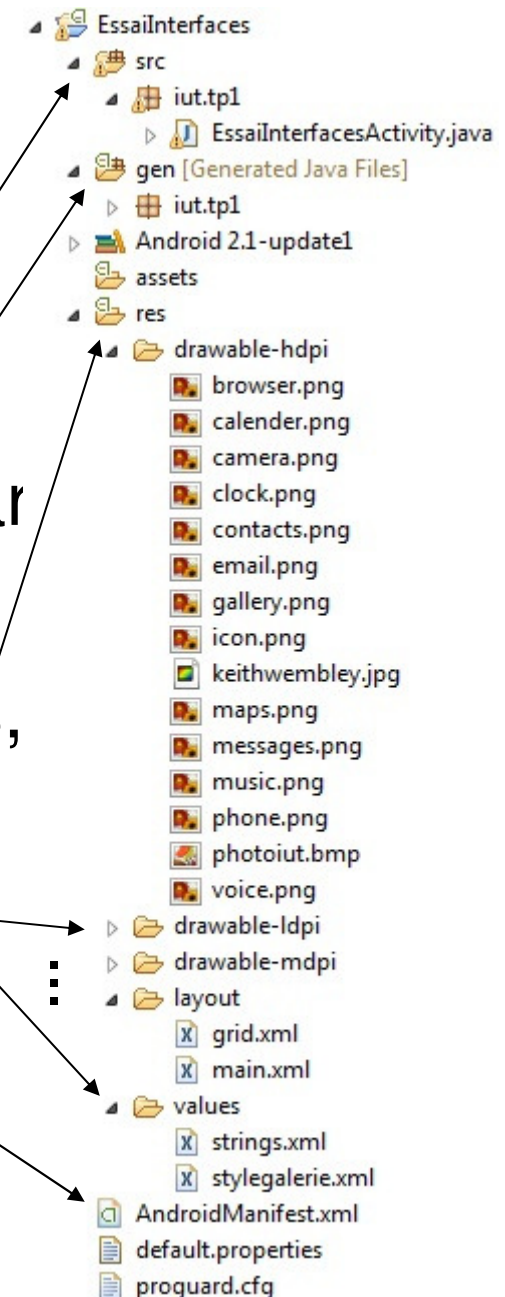
| Time | pid | tag | Message |
|----------------|-------|----------|--------------------------------------|
| 07-12 10:04... | D 389 | Andro... | --- registering native functions --- |
| 07-12 10:04... | D 389 | ddm-heap | Got feature list request |
| 07-12 10:04... | D 389 | Andro... | Shutting down VM |
| 07-12 10:04... | D 389 | dalvikvm | DestroyJavaVM waiting for non-dae... |
| 07-12 10:04... | D 389 | dalvikvm | DestroyJavaVM shutting VM down |
| 07-12 10:04... | D 389 | dalvikvm | HeapWorker thread shutting down |
| 07-12 10:04... | D 389 | dalvikvm | HeapWorker thread has shut down |
| 07-12 10:04... | D 389 | jdwp | JDWP shutting down net... |

Traces et erreurs d'exécution

Android SDK Content Loader

Structure d'un projet Eclipse

- **src** contient les fichiers source
- **gen** contient les fichiers générés par Eclipse
- **res** contient les ressources (images, descriptions d'interfaces, valeurs)
 - Organisé en sous dossiers par types
- **AndroidManifest** décrit les propriétés de l'application



Le fichier AndroidManifest

- Généré par Eclipse, contient la description de l'application

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="iut.tp1"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".DemoCoursActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- On modifiera ce fichier pour déclarer les éléments de l'application, les permissions, etc.

Le Fichier AndroidManifest

Modifiable directement ou par des formulaires

Android Manifest Application

▼ **Application Toggle**

The [application](#) tag describes application-level components contained in the package, as well as general application attributes.

☒ Define an <application> tag in the AndroidManifest.xml

▼ **Application Attributes**

Defines the attributes specific to the application.

| | | | | |
|------------------------|-----------------------------------------------|------------------------------------------|---------------------------|---------------------------------------------------------------|
| Name | <input type="text"/> | <input type="button" value="Browse..."/> | Persistent | <input type="text"/> |
| Theme | <input type="text"/> | <input type="button" value="Browse..."/> | Enabled | <input type="text"/> |
| Label | <input type="text" value="@string/app_name"/> | <input type="button" value="Browse..."/> | Debuggable | <input type="text"/> |
| Icon | <input type="text" value="@drawable/icon"/> | <input type="button" value="Browse..."/> | Manage space activity | <input type="text"/> <input type="button" value="Browse..."/> |
| Description | <input type="text"/> | <input type="button" value="Browse..."/> | Allow clear user data | <input type="text"/> |
| Permission | <input type="text"/> | | Test only | <input type="text"/> |
| Process | <input type="text"/> | <input type="button" value="Browse..."/> | Backup agent | <input type="text"/> <input type="button" value="Browse..."/> |
| Task affinity | <input type="text"/> | <input type="button" value="Browse..."/> | Allow backup | <input type="text"/> |
| Allow task reparenting | <input type="text"/> | | Kill after restore | <input type="text"/> |
| Has code | <input type="text"/> | | Restore needs application | <input type="text"/> |

Application Nodes S P A R M U Az

Manifest Application Permissions Instrumentation AndroidManifest.xml

Rubriques du fichier AndroidManifest

- Manifest
 - Nom du paquetage
 - Versions
 - SDK
- Application
 - Nom
 - Icône
 - Éléments constituant l'application (activités, services, ...)
- Permissions
 - Accès aux capteurs
 - Accès au réseau
 - Accès à la téléphonie
 - ...
- Instrumentation (pour les tests)

Les ressources

Les ressources

- Application embarquée \Rightarrow tout doit être dans le fichier .apk téléchargé
- Le répertoire **res** contient toutes les ressources qui seront mises dans le apk :
 - **drawable-hdpi** (images en haute définition)
 - **drawable-ldpi** (images en basse définition)
 - **drawable-mdpi** (images en moyenne définition)
 - **layout** (description en XML des interfaces)
 - **values** (définitions en XML de constantes : chaînes, tableaux, valeurs numériques ...)
 - **anim** (description en XML d'animations)
 - **menus** (description en XML de menus pour l'application)
 - **xml** (fichiers XML utilisés directement par l'application)
 - **raw** (tous les autres types de ressources : sons, vidéos, ...)

On peut ajouter d'autres sous répertoires

Créer des ressources valeurs

- Les ressources de type valeur sont décrites 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="mon_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>
```

Type
Nom
Valeur

**Les noms (identificateurs)
servent à les désigner :**

- Dans d'autres fichiers XML
- Dans le code

La classe R

- C'est une classe générée par Eclipse qui permet à l'application d'accéder aux ressources
- Elle contient des classes internes dont les noms correspondent aux types de ressources (id, drawable, layout ...)
- Elle est constituée à partir des fichiers placés dans les sous répertoires du répertoire **res**
- Une propriété est créée pour :
 - Chaque image placé dans drawable-xxxx
 - Chaque identificateur défini dans des fichiers XML (objets d'interface, constantes)
 - Chaque fichier placé dans les répertoires xml , raw ...

Utilisation des ressources

- Référencement d'une ressource dans un fichier xml. La forme générale est : "**@type/identificateur**"

Par exemple : *@string/machaine*

Fait référence à une chaîne contenue dans un fichier XML placé dans le répertoire **res/values** et définie comme suit :

```
<resources>
...
<string name="machaine">Contenu de cette chaîne</string>
...
</resources>
```

- Référencement d'une ressource dans le code. La forme générale est : **R.type.identificateur**

Par exemple : *R.string.machaine*

Fait référence à la même chaîne

La classe Resources

- Permet l'accès aux ressources répertoriées dans **R**
- On obtient une instance de cette classe par `getResources()` de l'activité
- Principales méthodes de la classe **Resources** (le paramètre est un identifiant défini dans **R** de la forme **R.type.nom**) :
 - boolean `getBoolean(int)`
 - int `getInteger(int)`
 - int[] `getArray(int)`
 - String `getString(int)`
 - String[] `getStringArray(int)`
 - int `getColor(int)`
 - float `getDimension(int)`
 - Drawable `getDrawable(int)`
- Exemple : `String titre = getResources().getString(R.string.ma_chaine);`

Utilisation des ressources

Accès aux ressources dans l'application

- Mise en place de l'interface principale
`setContentView(R.layout.nom_du_fichier_xml);`
- Mise en place d'interfaces supplémentaires
Par les classes `LayoutInflater` ou `MenuInflater`
- Accès direct à une valeur ou à une ressource :

```
String titre = getResources().getString(R.string.texte_titre);
```

```
Drawable monImage =  
    getResources().getDrawable(R.drawable.nom_de_l_image)
```

Uri (Uniform Resource Identifiers)

Désigne des ressources locales ou distantes
(plus général que les URL car non lié à un
protocole réseau)

- Création

- Ressource locale

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

- Ressource distante

- ```
Uri.parse("http://domaine.sous_domaine/chemin/nom_du_fichier");  
Uri.fromFile(File)
```

- Codage en chaîne de caractères

- ```
toString()
```

# Les applications



# Structure d'une application

- **Activité** ([android.app.Activity](#))  
Programme qui gère une interface graphique
- **Service** ([android.app.Service](#))  
Programme qui fonctionne en tâche de fond sans interface
- **Fournisseur de contenu** ([android.content.ContentProvider](#))  
Partage d'informations entre applications
- **Ecouteur d'intention diffusées** ([android.content.BroadcastReceiver](#)) :  
Permet à une application de récupérer des informations générales (réception d'un SMS, batterie faible, ...)

## Éléments d'interaction

- **Intention** ([android.content.Intent](#)) : permet à une application d'indiquer ce qu'elle sait faire ou de chercher un savoir-faire
- **Filtre d'intentions** ([<intent-filter>](#)) : permet de choisir la meilleure application pour assurer un savoir-faire

# Déclaration des éléments dans AndroidManifest.xml

- **Activité**

`<activity>`

`<intent-filter>`

*... les savoir-faire*

`</intent-filter>`

`</activity>`

- **Service**

`<service>`

`<intent-filter>`

*... les savoir-faire*

`</intent-filter>`

`</service>`

- **Ecouteur d'intention  
diffusée**

`<receiver>`

`<intent-filter>`

*... les savoir-faire*

`</intent-filter>`

`</receiver>`

- **Fournisseur de contenu**

`<provider>`

`<grant-uri-permission .../>`

`<path-permission .../>`

`</provider>`

# Application Android

- Une activité = un programme + une interface
- Un service = un programme sans interface
- Une application =
  - Une activité principale
  - Eventuellement une ou plusieurs activités secondaires
  - Eventuellement un ou plusieurs services
  - Eventuellement un ou plusieurs écouteurs d'intentions diffusées
  - Eventuellement un ou plusieurs fournisseurs de contenu

# Contenu du fichier AndroidManifest

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

```
<manifest>
```

```
<uses-sdk />
```

```
<uses-permission />
```

Général

```
<application>
```

```
<activity>
```

```
<intent-filter>
```

```
<action />
```

```
<category />
```

```
<data />
```

```
</intent-filter>
```

```
</activity>
```

Pour chaque activité

Pour chaque service

```
<service>
```

```
<intent-filter>
```

```
...
```

```
</intent-filter>
```

```
</service>
```

```
<receiver>
```

```
<intent-filter>
```

```
...
```

```
</intent-filter>
```

```
</receiver>
```

```
<provider>
```

```
<grant-uri-
permission />
```

```
</provider>
```

```
<uses-library />
```

```
</application>
```

```
</manifest>
```

# Activité Android

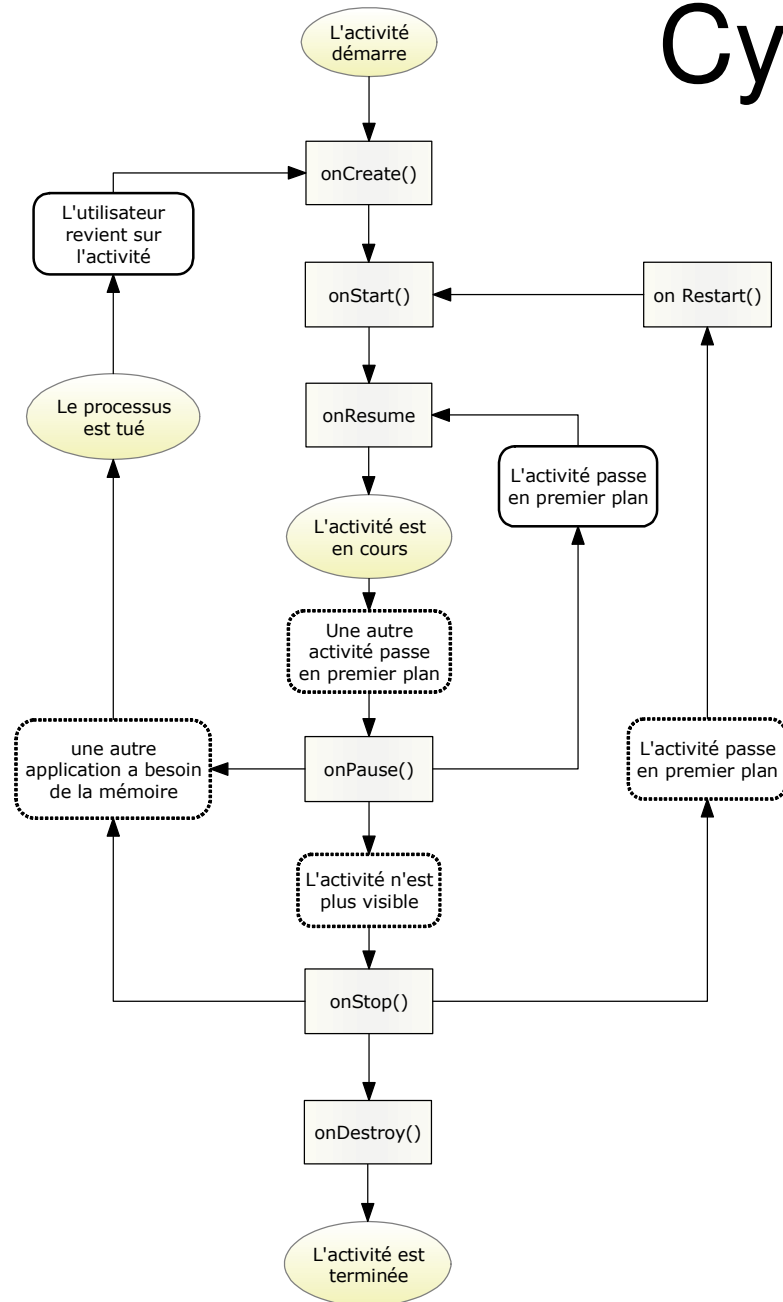
- Classe qui hérite de **Activity** ou d'une classe dérivée de Activity (par exemple de **MapActivity** pour utiliser Google maps, **ListActivity** ou **TabActivity** pour des interfaces particulières)
- On surcharge certaines méthodes qui sont appelées par Android pour définir le comportement (même principe que les applets) :
  - **onCreate**            lors de la création
  - **onDestroy**        lorsque l'activité se termine
  - **onStart**            lorsque l'activité démarre ou redémarre
  - **onPause**           lorsque l'activité n'est plus en premier plan
  - **onResume**        lorsque l'activité revient en premier plan
  - **onStop**            lorsque l'activité n'est plus visible
  - **onRestart**        lorsque l'activité redevient visible



# Cycle de vie d'une activité

- Android se réserve le droit de **tuer le processus unix d'une activité** s'il n'y a plus assez de ressources (mémoire). Les règles sont les suivantes :
  - Une activité en premier plan n'est tuée que si c'est elle qui consomme trop de ressources.
  - Une activité en arrière plan ou non visible peut être tuée.
- Lorsqu'une activité a été tuée, si on revient dessus elle est relancée (**onCreate**)
  - On peut sauvegarder l'état (c'est-à-dire les propriétés) d'une activité (dans **onPause**) pour le retrouver lorsqu'elle est recréée par le paramètre transmis à **onCreate**

# Cycle de vie d'une activité



- Etats principaux :
  - Active  
Après l'exécution de `onResume`
  - Suspendue  
Après l'exécution de `onPause`
  - Arrêtée  
Après l'exécution de `onStop`
  - Terminée  
Après l'exécution de `onDestroy`

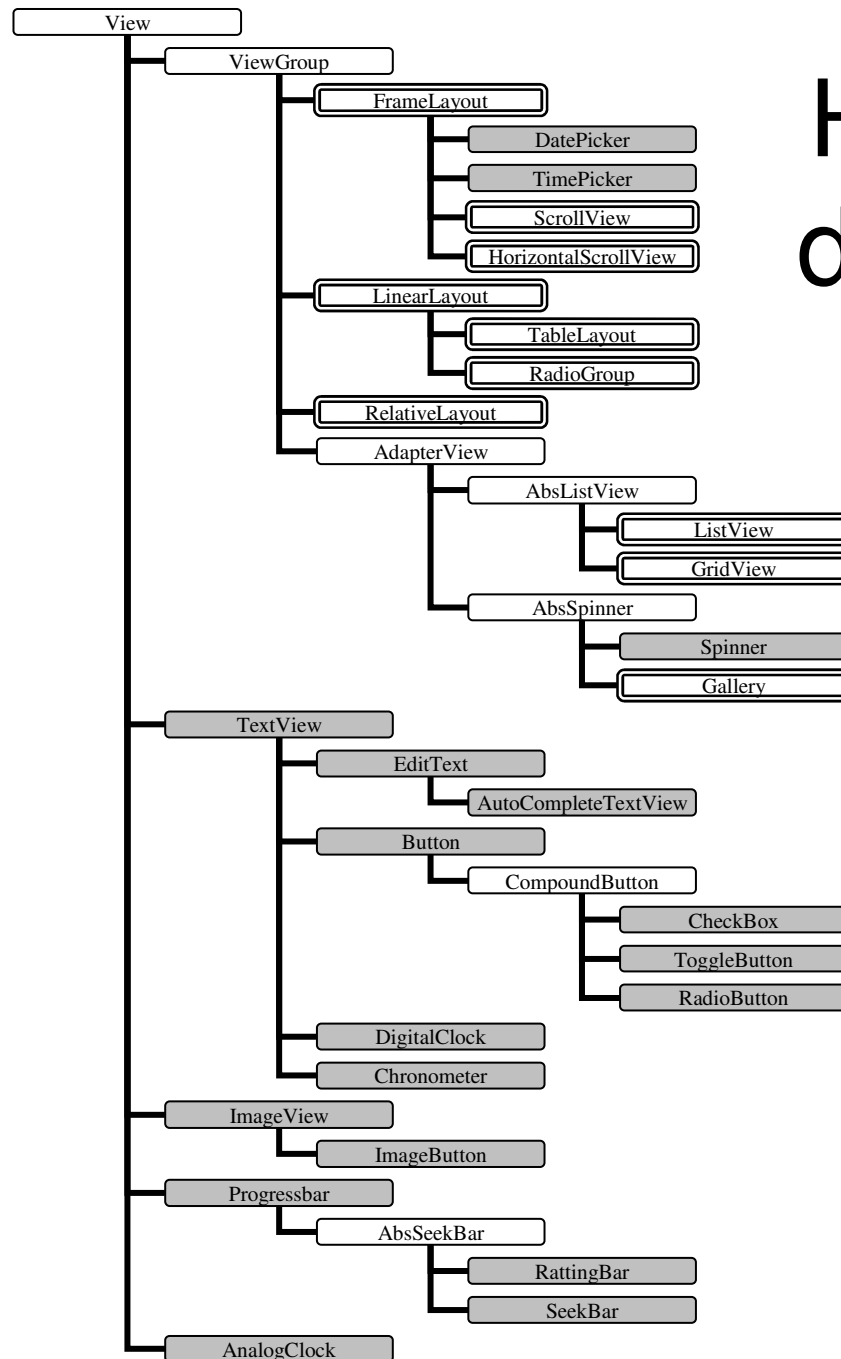
# Les interfaces

# Pensez vos interface pour un smartphone

- Ecran tactile de petite taille
  - Eviter les interfaces **trop touffues** (on ne peut pas agrandir l'écran comme on agrandit une fenêtre)
  - Eviter les éléments cliquables **trop petits** (il faut pouvoir cliquer avec le doigt même si on a des gros doigts)
  - Eviter les élément cliquables **trop tassés** (il faut pouvoir cliquer sur le bon élément même si on vise mal)
- Le défilement se fait par touché/glissé
  - **Pas trop d'ascenseurs** (on ne peut pas faire défiler un conteneur entier ET des éléments de ce conteneur dans le même sens)
  - Pas d'ascenseurs **mal placés** (si tous les éléments sont cliquables comment faire défiler sans cliquer ?)
- L'écran peut être tourné
- Tous les smartphones n'ont pas la même définition d'écran

# Création d'interfaces

- Par programme (comparable à java swing) mais avec des classes propres à Android
  - Définition de conteneurs (un conteneur = un conteneur + un mode de placement  $\equiv$  JPanel + Layout)
  - Définition d'éléments d'interaction (widgets) + placement et ajout dans les conteneurs
- Par description dans des fichiers xml (forme déclarative statique)
- Une interface est un arbre dont la racine est l'écran et les feuilles les éléments de l'interface (boutons, textes, cases à cocher, ...)



# Hiérarchie partielle de classes pour les interfaces

- View
- ViewGroup
- TextView

## Légende

Trait double = conteneurs ou groupes

Grisé = éléments d'interaction (widgets)

# Définir une interface en XML

## Définition de l'interface

```
<?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>
```

Espace de  
nommage  
d'Android  
(imposé)

Lorsqu'il s'agit  
d'un conteneur il  
doit être décrit  
avec son contenu



# Définir une interface en XML

## Description d'un conteneur de l'interface

**<Classe\_de\_conteneur**  
propriétés du conteneur  
**>**

Pour chaque  
conteneur

**<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>**

# Créer une interface à partir d'un fichier XML

- Dans l'activité principale

```
setContentView(R.layout.nom_du_fichier_xml)
```

- Ailleurs

```
LayoutInflater decodeur = LayoutInflater.from(contexte);
```

```
View vue = decodeur.inflate(R.layout.nom_du_fichier_xml, parent, false);
```

- *contexte* est l'activité qui gère cette interface
- *parent* est le contenant dans lequel doit se placer la vue constituée à partir du fichier XML

Il ne reste plus qu'à ajouter cette vue dans le conteneur.

# Unités de mesure dans les fichiers XML

- Dans les fichiers XML, les dimensions des éléments d'interface (taille, marges, ...) peuvent être exprimées en diverses unités :
  - Pixels (**px**)
  - Pouces (**in**)
  - Millimètres (**mm**)
  - Points (**pt**) = 1/72 pouce
  - Pixel à densité indépendante (**dp**) 1 dp = 1 pixel pour un écran de 160 dpi
  - Pixel à taille indépendante (**sp**) relatif à la taille des polices de caractères
- Dans les fichiers XML les unités sont exprimées sous la forme : “24.5mm” ou “65px” ...

# Couleurs dans les fichiers XML

- Dans les fichiers XML, les couleurs sont exprimées sous la forme d'une chaîne de caractères codant les composantes en hexadécimal : "#ARRVVB"
  - AA est l'opacité (00 totalement transparent, FF opaque)
  - RR est la composante rouge (00 à FF)
  - VV est la composante verte (00 à FF)
  - BB est la composante bleue (00 à FF)Si AA est omis la couleur est opaque

# Les conteneurs

- **FrameLayout** (un seul élément)
- **AbsoluteLayout** (plusieurs éléments placés par leur coordonnées)
- **LinearLayout** (plusieurs éléments placés horizontalement ou verticalement sans ascenseurs)
- **TableLayout** (plusieurs éléments en tableau sans ascenseurs)
- **RelativeLayout** (plusieurs éléments placés relativement les uns aux autres)
- **ScrollView** (un seul élément avec ascenseur vertical)
- **HorizontalScrollView** (un seul élément avec ascenseur horizontal)

# Les groupes

Regrouper des éléments participant à un choix

- **ListView** (plusieurs éléments organisés en liste verticale avec séparateurs). Souvent utilisé pour des listes de mots (type menu).
- **GridView** (plusieurs éléments organisés en table). Souvent utilisé pour des tables de mots (type menu).
- **RadioGroup** (groupe de boutons radio dont un seul peut être coché à la fois)
- **Gallery** (plusieurs éléments organisées horizontalement avec défilement). Souvent utilisé pour des images

# Propriété communes aux éléments d'interface (conteneurs et widgets)

## ***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 ou de le référencer dans d'autres fichiers XML. Les éléments ne devant pas être référencés peuvent ne pas avoir d'identifiant.

[android:id](#)="@+id/*mon\_ident*" permettra de retrouver cet élément par `findViewById(R.id.mon_ident)`.

Méthode correspondante : [setId\(int\)](#)



# Propriété communes aux éléments d'interface (conteneurs et widgets)

## **Visibilité**

[android:visibility](#)

Rend l'élément [visible](#), [invisible](#) ou [absent](#) (avec [invisible](#) la place est conservée, avec [absent](#) la place n'est pas conservée).

## **Fond**

[android:background](#) couleur ou une image de fond

## **Taille**

[android:minHeight](#) et [android:minWidth](#) dimensions minimales

## **Placement des éléments contenus (défini pour chaque élément)**

[android:layout\\_height](#) et [android:layout\\_width](#) place prise par l'élément dans le conteneur :

- [FILL\\_PARENT](#) remplit toute la place
- [WRAP\\_CONTENT](#) occupe la place nécessaire

[android:layout\\_gravity](#) positionnement de l'élément dans le conteneur

top, bottom, left, right, center\_vertical, fill\_vertical, center\_horizontal, fill\_horizontal, center, fill

# Propriété communes aux éléments d'interface (conteneurs et widgets)

## ***Ascenseurs (s'il y en a)***

[android:fadeScrollbars](#) Pour choisir de faire disparaître ou pas les ascenseurs lorsqu'ils ne sont pas utilisés

[android:scrollbarDefaultDelayBeforeFade](#) Définit le délai avant que les ascenseurs non utilisés disparaissent

[android:scrollbarFadeDuration](#) Définit la durée d'effacement des ascenseurs

## ***Marges internes (défini pour chaque élément)***

[android:layout\\_paddingBottom](#) , [android:layout\\_paddingLeft](#) ,  
[android:layout\\_paddingRight](#) , [android:layout\\_paddingTop](#)

## ***Marges externes (défini pour chaque élément)***

[android:layout\\_marginBottom](#) , [android:layout\\_marginLeft](#) ,  
[android:layout\\_marginRight](#) , [android:layout\\_marginTop](#)

# Propriété communes aux éléments d'interface (conteneurs et widgets)

## ***Prise en compte des événements***

- *Prise en compte des clics sur l'élément*  
[android:clickable](#) Autorise ou interdit la prise en compte des clics

Méthode correspondante : [setClickable\(boolean\)](#)

- *Prise en compte des clics longs sur l'élément*  
[android:longClickable](#) **Autorise** ou interdit la prise en compte des clics longs

Méthode correspondante : [setLongClickable\(boolean\)](#)

***On ajoutera ensuite un écouteur d'événements  
pour les traiter***

# Exemple d'interface simple

Un LinearLayout contenant 2 éléments placés l'un sous l'autre

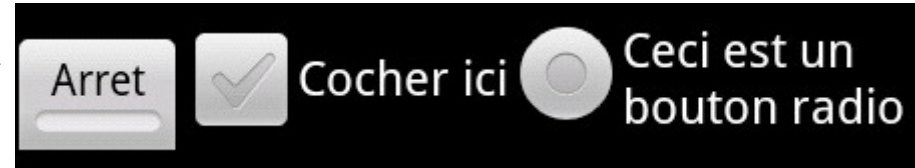
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 >
 <Spinner android:id="@+id/planetes"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 />
 <ProgressBar android:id="@+id/attente"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 />
</LinearLayout>
```



# Exemple d'interface simple

Un LinearLayout contenant 3 éléments placés l'un à coté de l'autre

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="horizontal"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 >
 <ToggleButton ...
 />
 <CheckBox ...
 />
 <RadioButton...
 />
</LinearLayout>
```

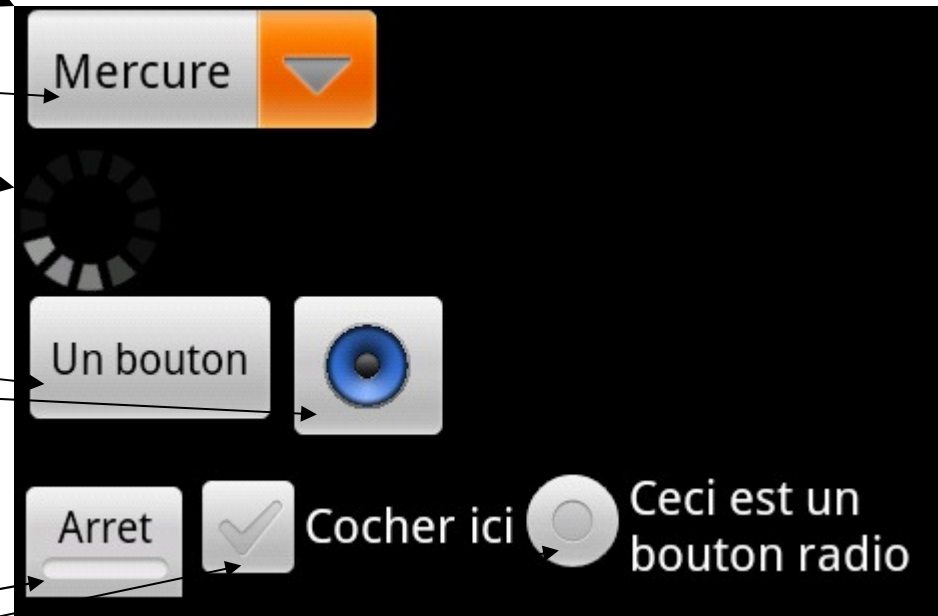


The diagram illustrates the mapping between the XML code and the resulting UI layout. The XML code defines a horizontal LinearLayout with three children: a ToggleButton, a CheckBox, and a RadioButton. The UI layout shows these elements in sequence: a button labeled 'Arret', a checked checkbox, and a radio button. Arrows indicate the mapping: 'horizontal' points to the layout orientation, 'ToggleButton' points to the 'Arret' button, 'CheckBox' points to the checked checkbox, and 'RadioButton' points to the radio button. The text 'Ceci est un bouton radio' is also present next to the radio button.

# Exemple d'interface complexe

Un LinearLayout vertical contenant 2 éléments + 2 LinearLayout horizontaux

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 >
 <Spinner ... />
 <ProgressBar ... />
 <LinearLayout
 android:orientation="horizontal"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 >
 <Button .../>
 <ImageButton .../>
 </LinearLayout>
 <LinearLayout
 android:orientation="horizontal"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 >
 <ToggleButton .../>
 <CheckBox .../>
 <RadioButton ... />
 </LinearLayout>
</LinearLayout>
```



# Les Contenants



# FrameLayout

- Ne contient qu'un seul élément (si on en met plusieurs ils se superposent)
- Propriétés supplémentaires :

## Contenu

[android:foreground](#) Pour définir une couleur ou une image.

[android:foregroundGravity](#) Pour positionner l'image

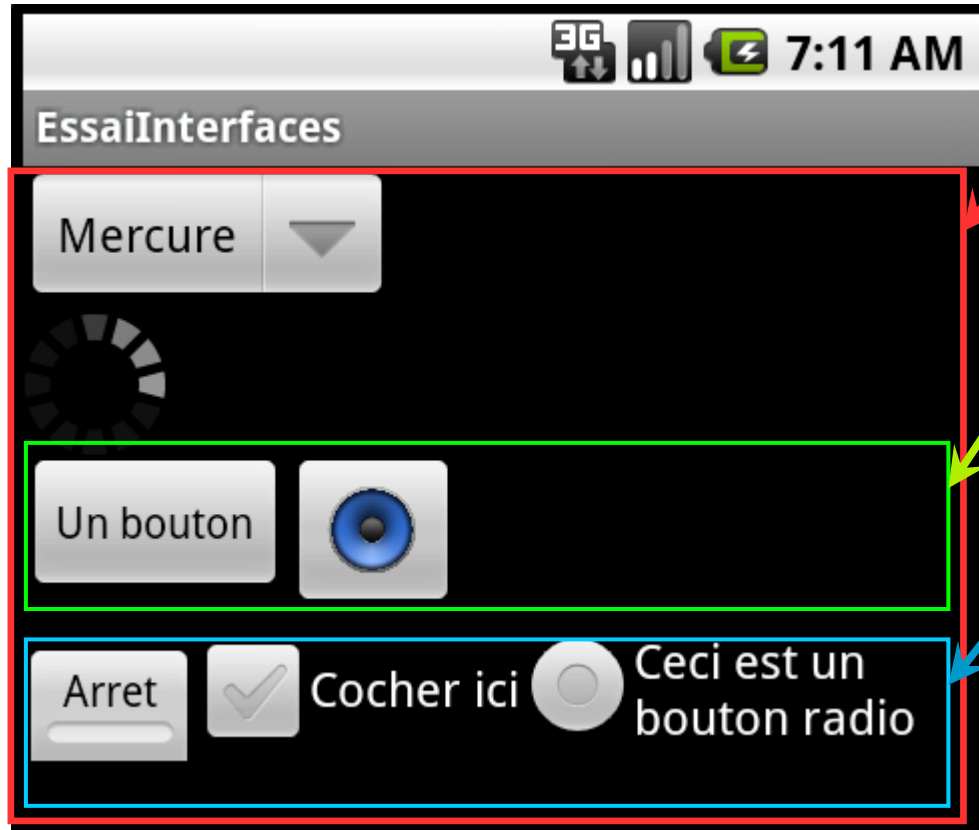
# LinearLayout

- Pour placer plusieurs éléments en ligne ou en colonne sans ascenseur (utiliser ScrollView et/ou HorizontalScrollView).
- Propriétés supplémentaires :  
[android:orientation](#) Pour en définir le sens du LinearLayout (vertical ou horizontal)

[android:layout\\_weightSum](#) Un paramètre de type : [android:layout\\_weight](#) peut être associé à chacun des éléments placés dans le LinearLayout pour indiquer leur poids de redimensionnement relatif à la valeur de layout\_weightSum.

Par exemple : [android:layout\\_weightSum](#) = "100" permettra de placer 2 éléments ayant [android:layout\\_weight](#) = "60" et [android:layout\\_weight](#) = "40"

# Exemple avec LinearLayout



LinearLayout  
vertical

LinearLayout  
horizontal

LinearLayout  
horizontal

# AbsoluteLayout

- Pour placer les éléments par positions fixes.
- Chaque élément ajouté dans un AbsoluteLayout indiquera sa position en mettant dans ses propriétés :
  - `android:layout_x="unité"`
  - `android:layout_y="unité"`

# TableLayout

- Pour placer des éléments en tableau sans ascenseurs (pour en avoir le mettre dans un ScrollView et/ou un HorizontalScrollView).
- Propriétés supplémentaires :
  - `android:collapseColumns` Pour définir les numéros de colonnes à cacher
  - `android:shrinkColumns` Pour définir les numéros de colonnes qui peuvent être rétrécies en fonction de la place disponible
  - `android:stretchColumns` Pour définir les numéros de colonnes qui peuvent être agrandies en fonction de leur contenu
- Chaque élément ajouté dans un `TableLayout` indiquera le nombre de colonnes qu'il occupe en mettant dans ses propriétés :
  - `android:layout_span` (par défaut 1)

# Exemple avec TableLayout

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_height="fill_parent"
 android:layout_width="fill_parent">
```

```
 <TableRow>
```

```
 <TextView
```

```
 ...
```

```
 />
```

```
 <EditText android:id="@+id/nom"
```

```
 ...
```

```
 />
```

```
 <Button android:id="@+id/envoyer"
```

```
 ...
```

```
 />
```

```
 </TableRow>
```

```
 <TableRow>
```

```
 <ProgressBar android:id="@+id/attente"
```

```
 android:layout_span="2"
```

```
 ...
```

```
 />
```

```
 <Button android:id="@+id/annuler"
```

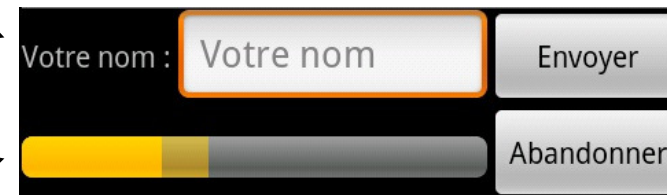
```
 ...
```

```
 android:text="Abandonner"
```

```
 />
```

```
 </TableRow>
```

```
</TableLayout>
```



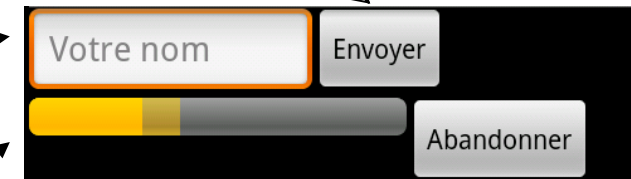
# RelativeLayout

- Permet de placer des éléments les uns relativement aux autres
  - Placement par rapport au conteneur
    - `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)
  - Placement par rapport aux autres éléments
    - `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"/`



# Exemple avec RelativeLayout

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_height="fill_parent"
 android:layout_width="fill_parent">
 <EditText android:id="@+id/nom"
 ...
 android:layout_alignParentTop="true"
 android:layout_alignParentLeft="true"
 />
 <Button android:id="@+id/envoyer"
 android:layout_toRightOf="@+id/nom"
 ...
 />
 <ProgressBar android:id="@+id/attente"
 android:layout_below="@+id/nom"
 ...
 />
 <Button android:id="@+id/annuler"
 android:layout_toRightOf="@+id/attente"
 android:layout_below="@+id/nom"
 ...
 />
</RelativeLayout>
```



# ScrollView et HorizontalScrollView

- En général utilisés pour ajouter des ascenseurs à un conteneur.
- Ne peuvent contenir qu'un seul élément (le plus souvent un conteneur).
- Propriétés supplémentaires :
  - [android:fillViewport](#)="b" (où b vaut true ou false) indique si le contenu doit être étiré pour occuper la place disponible ou pas
- **ATTENTION** : En raison de l'écran tactile le défilement porte sur l'élément le plus externe (le plus haut dans l'arbre de l'interface)

# Les Groupes

# ListView

- Place les éléments en liste verticale et ajoute un ascenseur si nécessaire
  - **Séparateurs**
    - `android:divider` Pour définir la couleur des séparateurs ou pour utiliser une image comme séparateur.
    - `android:dividerHeight`="unité" Pour définir la hauteur des séparateurs (même s'ils contiennent une image)
  - **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).

# ListView (contenu)

- **En XML (texte seulement)**

`android:entries="@array/maliste"` définit le contenu de la liste à partir du contenu d'un fichier xml placé dans `res/values/` et qui a la forme :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <string-array name="maliste">
 <item>premier élément</item>
 <item>deuxième élément</item>
 ...
 <item>dernier élément</item>
 </string-array>
</resources>
```

- **Dans le code (éléments quelconques)**

*On utilise un gestionnaire de contenu (Adapter)*

- `setAdater(Adapter)` pour associer à la ListView
- **Soit de classe prédéfinie (`ArrayAdapter` , `SimpleAdapter`, `CursorAdapter`)**
  - `ArrayAdapter(Context, type)` le second paramètre est une type prédéfini :  
`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  
(une case à cocher apparaît à coté de chaque élément de la liste)
  - `ArrayAdapter.add(Object)` pour remplir la liste
- **Soit de classe personnalisée (héritage de `BaseAdapter`)**

# Exemple de ListView

- En XML

- Dans le XML d'interface

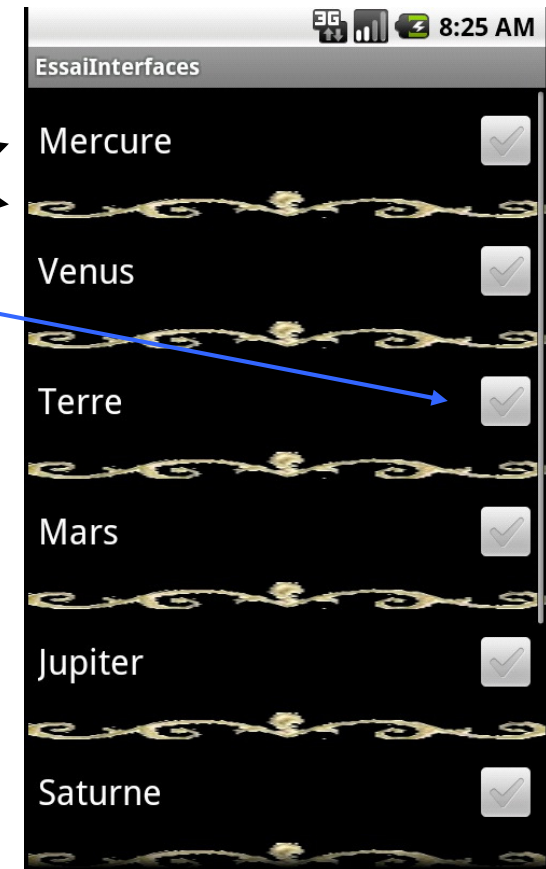
```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+id/liste_de_planetes"
 android:entries="@array/planetes"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:divider="@drawable/separateur"
 android:dividerHeight="25px"
 android:choiceMode="multipleChoice"
/>
```

- Dans le XML de valeurs

```
<string-array name="planetes">
 <item>Mercure</item>
 ...
 <item>Neptune</item>
</string-array>
```

- Dans le code

```
ListView liste = (ListView) findViewById(R.id.liste_de_planetes);
String[] elements = getResources().getStringArray(R.array.planetes);
ArrayAdapter<String> adaptateur = new ArrayAdapter<String>(this,
 android.R.layout.simple_list_item_multiple_choice);
for (int i=0; i<elements.length; i++) adaptateur.add(elements[i]);
liste.setAdapter(adaptateur);
```



# GridView

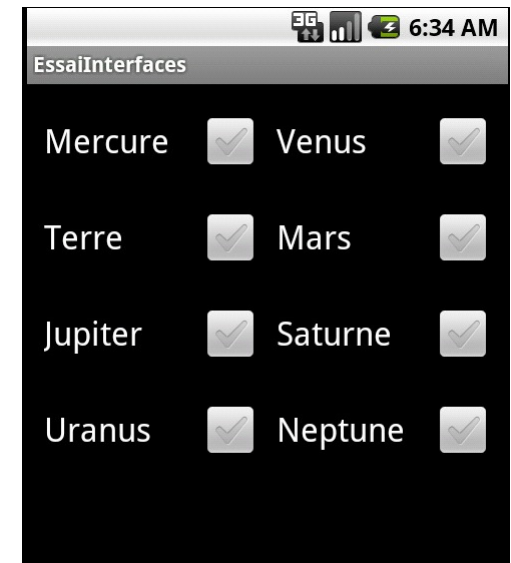
- Fonctionne comme ListView mais permet une présentation en plusieurs colonnes
- Exemple

- Dans le XML d'interface

```
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+id/liste_de_planetes"
 android:entries="@array/planetes"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:numColumns="2"
 android:stretchMode="columnWidth"
 android:columnWidth="60dp"
 android:gravity="fill_horizontal"
 android:choiceMode="multipleChoice"
/>
```

- Dans le code (même principe)

```
GridView table = (GridView) findViewById(R.id.liste_de_planetes);
String[] elements = getResources().getStringArray(R.array.planetes);
ArrayAdapter<String> adaptateur = new ArrayAdapter<String>(this,
 android.R.layout.simple_list_item_multiple_choice);
for (int i=0; i<elements.length; i++) adaptateur.add(elements[i]);
table.setAdapter(adaptateur);
```



# RadioGroup

- Pour grouper des boutons radio (ajoute un ascenseur si nécessaire)
- Un seul bouton peut être coché à la fois (attention à l'état initial qui n'est pris en compte par le RadioGroup que s'il est fait par la méthode **check** du RadioGroup)
- Exemple de fichier XML :

```
<RadioGroup
 android:id="@+id/groupe_de_boutons"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:orientation="horizontal"
>
 <RadioButton
 ...
 />
 <RadioButton
 ...
 />
</RadioGroup>
```





# Gallery

- Normalement utilisé pour faire des galeries d'images avec défilement horizontal
- Propriétés supplémentaires
  - `android:animationDuration` Pour définir la durée de la transition (en ms) lorsque l'on passe d'un élément à l'autre
  - `android:unselectedAlpha` Pour définir la transparence des éléments non sélectionnés
- Pour remplir une galerie il faut un **Adapter** (comme pour ListView) mais que l'on doit écrire par héritage de la classe **BaseAdapter** puis l'associer à la galerie par la méthode `setAdapter`

Exemple de fichier XML :

```
<Gallery android:id="@+id/magalerie"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:unselectedAlpha="0.5"
>
```



# Les Composants d'interface (Widgets)

# Placement des composants d'interface

- Placement dans le conteneur
  - Lorsqu'ils sont placés dans un `AbsoluteLayout` leur position est définie par :  
`android:layout_x="unité"`  
`android:layout_y="unité"`
  - Lorsqu'ils sont placés dans un `RelativeLayout` leur position est définie par rapport à ce contenant et/ou aux autres composants
  - Lorsqu'ils sont placés dans un autre contenant leur position est définie par ce contenant
- Taille
  - `android:layout_height="t"` (où t peut être une unité ou prendre les valeurs `FILL_PARENT` ou `WRAP_CONTENT`)
  - `android:layout_width="t"` (où t peut être une unité ou prendre les valeurs `FILL_PARENT` ou `WRAP_CONTENT`)
- 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

# Placement des composants d'interface

- Occupation du conteneur (sauf Relative et Absolute Layout)  
`android:layout_gravity="g"` (où g peut prendre les valeurs : top, bottom, left, right, center\_vertical, fill\_vertical, center\_horizontal, fill\_horizontal, center, fill)

On peut combiner (si ça a une sens) plusieurs valeurs par |

Par exemple : `android:layout_gravity="top|right"`

- Dans le cas d'un LinearLayout ce paramètre ne permet pas de modifier le placement implicite (les uns à côté des autres ou les uns sous les autres selon l'orientation du LinearLayout)
  - Comme un traitement de texte on ne peut pas faire une ligne dont le début est cadré à gauche et la fin est cadrée à droite !
  - Pour obtenir ce type de placement il faut encapsuler l'élément dans un FrameLayout et le placer dans celui-ci

# ImageView

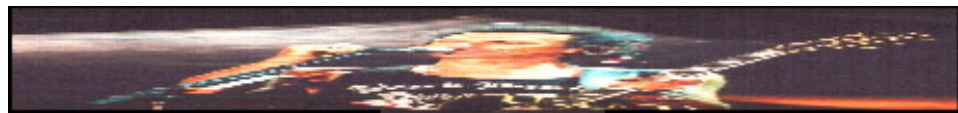
- Permet d'afficher des images
- Propriétés :
  - Contenu
    - `android:src` Pour définir une couleur ou une image.
    - `android:tint` Pour définir une couleur qui teinte l'image
  - Position et dimensions de l'image
    - `android:adjustViewBounds` La taille de l'ImageView sera ou pas modifiée
    - `android:baselineAlignBottom` Cadrage ou pas de l'image en bas de la zone
    - `android:cropToPadding` L'image sera ou pas coupée si elle est plus grande que la taille disponible
    - `android:scaleType` Pour définir le mode de redimensionnement de l'image avec ou sans déformation. (voir exemples transparent suivant)
  - Taille
    - `android:maxHeight` Pour définir la hauteur maximale
    - `android:maxWidth` Pour définir la largeur maximale

# ImageView Examples

```
<ImageView android:id="@+id/image"
 android:src="@drawable/keithwembley"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:maxHeight="200px"
 android:adjustViewBounds="true"
 android:scaleType="centerCrop"
>
```



```
<ImageView android:id="@+id/image"
 android:src="@drawable/keithwembley"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:maxHeight="200px"
 android:adjustViewBounds="true"
 android:scaleType="fitXY"
>
```



# TextView et EditText

- **TextView** est normalement utilisé pour afficher un texte tandis que **EditText** l'est pour saisir du texte
- Propriétés :
  - Dimensions de texte
    - android:ems* Pour définir la taille du texte en caractères
    - android:maxems* Pour définir le nombre de caractères des lignes du texte
    - android:height*="unité"
    - android:maxheight*="unité"
    - android:minheight*="unité"
    - android:width*="unité"
    - android:maxwidth*="unité"
    - android:minwidth*="unité"

# TextView et EditText

- **Contenu**

`android:text`="texte a afficher"

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

- **Taille et aspect du texte**

`android:textSize`="unité" utiliser de préférence l'unité sp qui est liée aux polices

`android:textScaleX` Pour définir l'échelle horizontale du texte

`android:textStyle`="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` Pour définir le nombre de lignes du texte

`android:maxlines` Pour définir le nombre maximal de lignes du texte

`android:minlines` Pour définir le nombre minimal de lignes du texte

`android:lineSpacingExtra` Pour définir l'espace supplémentaire entre les lignes

`android:scrollHorizontally` Pour autoriser ou interdire le défilement horizontal du texte



# TextView et EditText

- **Comportement du texte**

`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` Pour valider ou pas le mode correction du texte

`android:capitalize="c"` (où c peut être : `none`, `sentences`, `words`, `characters`) indique le type de saisies que le texte mémorise et peut re-proposer.

`android:digits` Pour indiquer si la saisie n'accepte que du numérique ou pas

`android:numeric="x"` (où x peut être `integer`, `signed`, `decimal`) définit le mode de saisie numérique

`android:password` pour cacher ou pas le texte lors d la saisie

`android:phoneNumber` Pour indiquer si la saisie n'accepte que des n<sup>os</sup> de téléphone

`android:inputType` Pour définir le mode de saisie (`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`)

# TextView et EditText

- **Affichage**

`android:cursorVisible` Pour rendre visible ou non le curseur

`android:editable` Pour autoriser 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` Pour rendre ou pas les liens cliquables

`android:textIsSelectable` pour autoriser/interdire la sélection de texte

- **Couleurs et images**

`android:textColor` Pour définir une couleur au texte

`android:textColorHighlight` Pour définir une couleur de surlignage du texte

`android:textColorHint` Pour définir une couleur au texte par défaut

`android:textColorLink` Pour définir une couleur aux liens du texte

`android:drawableBottom` Pour définir une couleur ou une image de fond au  
texte

# AutoCompleteTextView

- C'est une spécialisation de EditText pour apporter l'auto complétion



- Propriétés supplémentaires:

`android:completionHint`="texte" texte affiché en titre du menu déroulant

`android:completionThreshold` Pour définir 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 hauteur du menu déroulant

`android:dropDownHorizontalOffset` Pour définir le décalage horizontal du menu déroulant

`android:dropDownVerticalOffset` Pour définir le décalage vertical du menu déroulant

# Les boutons

- Button

Mêmes paramètres que TextView



- ImageButton

Mêmes paramètres que ImageView càd :



`android:src="couleur"` pour définir une couleur ou une image

`android:adjustViewBounds` Pour indiquer si la taille du bouton doit ou pas être ajustée à celle de l'image

`android:baselineAlignBottom` Pour indiquer que l'image est placée ou pas en bas de la zone

`android:cropToPadding` Pour indiquer si l'image sera coupée ou pas si elle est plus grande que la taille disponible

`android:scaleType="s"` (où s peut prendre les valeurs : matrix, fitXY, fitStart, fitCenter, fitEnd, center, centerCrop, centerInside) permet de redimensionner ou pas l'image à la taille disponible et/ou de la déformer.

`android:maxHeight` Pour définir la hauteur disponible

`android:maxLength` Pour définir la largeur disponible

`android:tint` Pour définir une couleur qui teinte l'image

# Les éléments à deux états

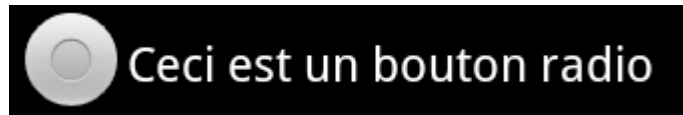
Ils ont les mêmes paramètres que TextView auxquels vient s'ajouter la définition de l'état initial :

`android:checked="b"` où b vaut true ou false Pour définir l'état initial

- CheckBox



- RadioButton



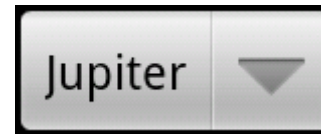
- ToggleButton



`android:disabledAlpha` pour définir la transparence appliquée lorsque le bouton est inactif  
`android:textOff` Pour définir le texte quand le bouton n'est pas allumé  
`android:textOn` Pour définir le texte quand le bouton n'est pas allumé

# Liste de choix (Spinner)

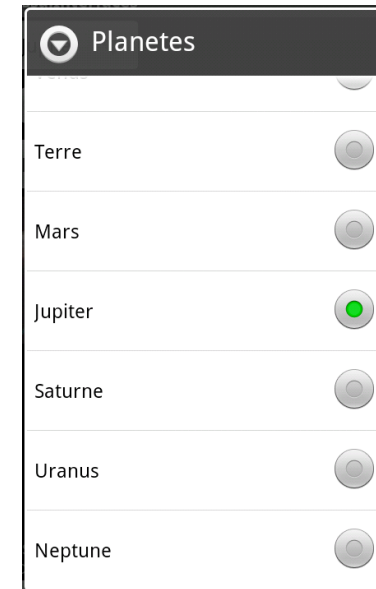
- Affiche le choix actuel et affiche un RadioGroup quand on clique dessus pour le changer



- Propriétés :

`android:prompt` Pour définir 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 du contenu d'un fichier xml placé dans `res/values/` qui a la forme suivante :

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



# Choix de date et d'heure

- DatePicker



`android:startYear` Pour définir l'année de départ du calendrier affiché

`android:endYear` Pour définir l'année de fin du calendrier affiché

`android:minDate` Pour définir la date affichée de départ du calendrier  
sous la forme mm/jj/aaaa

`android:maxDate` Pour définir la date affichée de fin du calendrier  
sous la forme mm/jj/aaaa

- TimePicker



# ProgressBar

- Deux comportements selon que l'on connaît ou pas la valeur maximale
  - `android:indeterminate` Pour définir le type de progressBar (true=indéterminé, false=déterminé).
- Animation (si indéterminé)
  - `android:indeterminateBehavior="i"` (où i peut être : `repeat` ou `cycle`) définit le comportement de l'animation pour le type circulaire (repeat=recommence l'animation, cycle=changer le sens de l'animation)
- Dimensions
  - `android:maxHeight="unité"`
  - `android:minHeight="unité"`
  - `android:maxWidth="unité"`
  - `android:minWidth="unité"`
- Valeurs (si déterminé)
  - `android:max` Pour définir la valeur maximale
  - `android:progress` Pour définir la valeur initiale
  - `android:secondaryProgress` Pour définir une valeur secondaire (par exemple celle d'un buffer comme on le voit sur des vidéos en streaming)



# Formes des ProgressBar

- En l'absence de paramètre `style` la forme est circulaire
- Pour obtenir d'autres forme on utilise le paramètre `style` :
  - `style="?android:attr/s"` où s peut être :
    - `progressBarStyleHorizontal`
    - `progressBarStyleSmall`
    - `progressBarStyleLarge`



- On ne peut pas changer la couleur

## SeekBar



C'est un ProgressBar sous forme de barre horizontale dotée d'un curseur permettant de modifier la valeur si on a choisi

`android:indeterminate="false"`

sinon le curseur ne marche pas et la barre bouge sans arrêt.

# RatingBar



- Paramètres :

[android:isIndicator](#) Pour indiquer si l'utilisateur peut modifier la valeur ou pas (true= non modifiable)

[android:numStars](#) Pour définir le nombre d'étoiles affichées

[android:rating](#) Pour définir la position initiale

[android:stepSize](#) Pour définir le pas de progression (on peut colorier des  $\frac{1}{4}$  d'étoiles par exemple)

# Horloges et Chronomètres

- AnalogClock



- DigitalClock

4:58:17 pm

- Chronometer

Ce cours a commencé depuis 51:45 déjà

[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)

Les événements (interaction)

# Traitement des événements

- Tous les éléments d'interface (conteneurs et widgets) possèdent les méthodes suivantes :
  - [setOnClickListener\(View.OnClickListener\)](#) associe un écouteur d'événements aux **clics** sur la vue
  - [setOnLongClickListener\(View.OnLongClickListener\)](#) associe un écouteur d'événements aux **clics longs** sur la vue
  - [setOnKeyListener\(View.OnKeyListener\)](#) associe un écouteur d'événements aux **actions clavier** sur la vue
  - [setOnTouchListener\(View.OnTouchListener\)](#) associe un écouteur d'événements aux **touchés** sur la vue

qui permettent de leur associer des écouteurs d'événements

- Certains éléments ont des écouteurs spécifiques

# Traitement des événements (les bonnes habitudes)

- Quand un widget est modifié la méthode correspondante de l'écouteur d'événements associé est exécutée
- Ceci est vrai que le widget soit modifié par l'utilisateur ou par programme.
- Il est donc préférable de ne mettre en place les écouteurs d'événements qu'**après** avoir totalement initialisé les widgets pour éviter qu'ils ne s'exécutent au cours de ces initialisations

# Événements généraux

| Événement | Association<br>Classe                                              | Méthode<br>• Paramètres                                                                                   |
|-----------|--------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| Clic      | setOnClickListener<br><a href="#">View.OnClickListener</a>         | <a href="#">onClick(View)</a><br>• Élément concerné                                                       |
| Clic long | setOnLongClickListener<br><a href="#">View.OnLongClickListener</a> | <a href="#">onLongClick(View)</a><br>• Élément concerné                                                   |
| Clavier   | setOnKeyListener<br><a href="#">View.OnKeyListener</a>             | <a href="#">onKey(View, int, KeyEvent)</a><br>• Élément concerné<br>• Code clavier<br>• Événement clavier |
| Touché    | setOnTouchListener<br><a href="#">View.OnTouchListener</a>         | <a href="#">onTouch(View, MotionEvent)</a><br>• Élément concerné<br>• Événement de touché                 |

# Evénements spécifiques

- ListView , GridView et Gallery

| Evénement<br>sur un<br>élément | Association<br>Classe                                                             | Méthode<br>• Paramètres                                                                                                                                                                                                                                              |
|--------------------------------|-----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Clic                           | setOnItemClickListener<br><a href="#">AdapterView.OnItemClickListener</a>         | <a href="#">onItemClick</a> ( <a href="#">AdapterView</a> , <a href="#">View</a> ,<br>int, long) <ul style="list-style-type: none"><li>• Gestionnaire de contenu</li><li>• Élément concerné</li><li>• Rang de l'élément</li><li>• Identifiant de l'élément</li></ul> |
| Clic long                      | setOnItemLongClickListener<br><a href="#">AdapterView.OnItemLongClickListener</a> | <a href="#">onItemLongClick</a> ( <a href="#">AdapterView</a> ,<br><a href="#">View</a> , int, long) <ul style="list-style-type: none"><li>• Idem</li></ul>                                                                                                          |
| Sélection                      | setOnItemSelectedListener<br><a href="#">AdapterView.OnItemSelectedListener</a>   | <a href="#">onItemSelected</a> ( <a href="#">AdapterView</a> ,<br><a href="#">View</a> , int, long) <ul style="list-style-type: none"><li>• Idem</li></ul> <a href="#">onNothingSelected</a> ( <a href="#">AdapterView</a> )                                         |



# Événements spécifiques

- Spinner et AutoCompleteTextView

| Événement<br>sur un<br>élément | Association<br>Classe                                                           | Méthode<br>• Paramètres                                                                                                                                                                                                                                                                                                                       |
|--------------------------------|---------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Sélection                      | setOnItemSelectedListener<br><a href="#">AdapterView.OnItemSelectedListener</a> | <a href="#">onItemSelected</a> ( <a href="#">AdapterView</a> ,<br><a href="#">View</a> , int, long) <ul style="list-style-type: none"><li>• Élément permettant le choix</li><li>• Élément concerné</li><li>• Rang de l'élément</li><li>• Identifiant de l'élément</li></ul> <a href="#">onNothingSelected</a> ( <a href="#">AdapterView</a> ) |

# Événements spécifiques

- TextView et EditText

| Événement     | Association<br>Classe                                         | Méthode<br>• Paramètres                                                                                                                                                                                                                                                                                                                                                 |
|---------------|---------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Fin de saisie | setOnEditorActionListener<br><a href="#">TextWatcher</a>      | <a href="#">onEditorAction</a> ( <a href="#">TextView</a> , int, <a href="#">KeyEvent</a> ) <ul style="list-style-type: none"><li>• Élément concerné</li><li>• EditorInfo.IME_NULL (si touche Entrée)</li><li>• Événement clavier (si touche Entrée)</li></ul>                                                                                                          |
| Modification  | addTextChangedListener<br><a href="#">TextChangedListener</a> | <a href="#">beforeTextChanged</a> ( <a href="#">CharSequence</a> , int, int, int)<br><a href="#">afterTextChanged</a> ( <a href="#">CharSequence</a> , int, int, int) <ul style="list-style-type: none"><li>• Texte</li><li>• Point de départ de la modification</li><li>• Nombre de cars remplacés</li><li>• Nombre de cars de remplacement</li></ul>                  |
| Saisie        | setKeyListener<br><a href="#">KeyListener</a>                 | <a href="#">onKeyDown</a> ( <a href="#">View</a> , <a href="#">Editable</a> , int, <a href="#">KeyEvent</a> )<br><a href="#">onKeyUp</a> ( <a href="#">View</a> , <a href="#">Editable</a> , int, <a href="#">KeyEvent</a> ) <ul style="list-style-type: none"><li>• Élément concerné</li><li>• Texte</li><li>• Code de la touche</li><li>• Événement clavier</li></ul> |

# Événements spécifiques

- DatePicker

| Événement de choix | Association<br>Classe                                   | Méthode<br>• Paramètres                                                                                                                                                                         |
|--------------------|---------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Choix              | init<br><a href="#">DatePicker.OnDateChangeListener</a> | <a href="#">onDateChanged</a> ( <a href="#">DatePicker</a> ,<br>int, int, int) <ul style="list-style-type: none"><li>• Élément concerné</li><li>• Année</li><li>• Mois</li><li>• Jour</li></ul> |

- TimePicker

| Événement de choix | Association<br>Classe                                                      | Méthode<br>• Paramètres                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Choix              | setOnTimeChangeListener<br><a href="#">TimePicker.OnTimeChangeListener</a> | <a href="#">onTimeChanged</a> ( <a href="#">TimePicker</a><br>, int, int) <ul style="list-style-type: none"><li>• Élément concerné</li><li>• Heure</li><li>• Minutes</li></ul> |

# Evénements spécifiques

- SeekBar

| Événement            | Association<br>Classe                                                         | Méthode<br>• Paramètres                                                                                                                                    |
|----------------------|-------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Curseur déplacé      | setOnSeekBarChangeListener<br><a href="#">SeekBar.OnSeekBarChangeListener</a> | <a href="#">onProgressChanged</a> (<br><a href="#">SeekBar</a> , int, boolean)<br>• Élément concerné<br>• Position du curseur<br>• Action de l'utilisateur |
| Début de déplacement |                                                                               | <a href="#">onStartTrackingTouch</a> (<br><a href="#">SeekBar</a> )<br>• Élément concerné                                                                  |
| Fin de déplacement   |                                                                               | <a href="#">onStopTrackingTouch</a> (<br><a href="#">SeekBar</a> )<br>• Élément concerné                                                                   |

# Événements spécifiques

- RatingBar

| Événement       | Association<br>Classe                                                               | Méthode<br>• Paramètres                                                                                                                                                                                     |
|-----------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Valeur modifiée | setOnRatingBarChangeListener<br><a href="#">RatingBar.OnRatingBarChangeListener</a> | <a href="#">onRatingChanged</a> ( <a href="#">RatingBar</a> , float, boolean) <ul style="list-style-type: none"><li>• Élément concerné</li><li>• Valeur choisie</li><li>• Action de l'utilisateur</li></ul> |

- Chronometer

| Événement      | Association<br>Classe                                                                 | Méthode<br>• Paramètres                                                                                                              |
|----------------|---------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| Incrémentation | setOnChronometerTickListener<br><a href="#">Chronometer.OnChronometerTickListener</a> | <a href="#">onChronometerTick</a> ( <a href="#">Chronometer</a> ) <ul style="list-style-type: none"><li>• Élément concerné</li></ul> |

# Notifications

# La classe Toast

Texte qui apparaît en premier plan puis disparaît au bout d'un temps donné



Répertoire illisible

- **Création d'un Toast**

**Toast.makeText**(Context, String, int) renvoie l'objet de classe Toast créé.

- Le premier paramètre est l'activité
- 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) appelée avant l'affichage par show 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 (en pixels).

- **Affichage d'un Toast**

**show**() affiche le message pour la durée définie lors de sa création.

Menus



# Menus

- Deux types
  - Menu général de l'activité
  - Menu contextuel associé à un élément d'interface
- Contiennent des rubriques sous la forme texte et/ou image
- Décrits par un fichier XML placé dans res/menu (répertoire à créer) de la forme :

```
<?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>
```

# Sous menus

- Chaque élément d'un menu peut proposer des sous menus
- Décrits dans le fichier XML sous la forme :

```
<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

- Apparaît par appui de la touche Menu
- Création dans la méthode `onCreateOptionsMenu` de l'activité à partir du fichier xml de description du menu sous la forme :

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

# Menu général

## Réactions aux choix

- Dans la méthode `onOptionsItemSelected` de l'activité qui est appelée lorsque l'utilisateur fait un choix dans un menu ou un sous menu général :

```
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);
 }
}
```

# Menu contextuel

- Apparaît par appui long sur l'élément d'interface
- Associé à l'élément d'interface par la méthode :  
`registerForContextMenu(element_associe_au_menu_contextuel);`
- Création dans la méthode `onCreateContextMenu` de l'activité à partir du fichier xml de description du menu 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);
}
```

# Menu contextuel

## Réactions aux choix

- Dans la méthode `onContextItemSelected` de l'activité qui est appelée lorsque l'utilisateur fait un choix dans un menu ou un sous menu contextuel :

```
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);
 }
}
```

# Couleurs et images

# La classe Color

- **Différente de celle de java**

- `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**

- `Color.BLACK`, `Color.WHITE`
- `Color.LTGRAY`, `Color.GRAY`, `Color.DKGRAY`
- `Color.RED`, `Color.GREEN`, `Color.BLUE`
- `Color.CYAN`, `Color.MAGENTA`, `Color.YELLOW`
- `Color.TRANSPARENT`



# La classe Drawable

Classe de tout ce qui peut se dessiner (dont les images)

- Quelques types :
  - **Bitmap** : image PNG ou JPEG
  - **Nine Patch** : extension de PNG permettant d'indiquer comment la déformer. Le SDK propose un utilitaire **draw9path** placé dans le répertoire **tools**
  - **Shape** : dessin
  - **Layers** : calques
  - **States** : image ayant plusieurs états (aspects) par exemple pour avoir un aspect différent quand sélectionné, actif ...
  - ...

# La classe BitmapDrawable

- Spécialisation de Drawable pour les images

# La classe BitmapFactory

Permet de créer des images depuis diverses sources

- Un tableau d'octets (`decodeByteArray`)
- Un fichier (`decodeFile`)
- Une ressource (`decodeResource`)
- Un flux (`decodeStream`)

Ces créations utilisent des options (`BitmapFactory.Options`)

- `inSampleSize` pour réduire l'image
- `inScaled` pour redimensionner l'image
- `inDither` pour autoriser ou interdire le tramage
- `inPurgeable` pour libérer la mémoire occupée par l'image
- `outHeight`, `outWidth` pour définir la taille

Navigation entre activités

# Navigation entre activités

## Démarrer une activité

- Mode explicite :

On indique la classe de l'activité à lancer (par exemple dans une application multi interfaces)

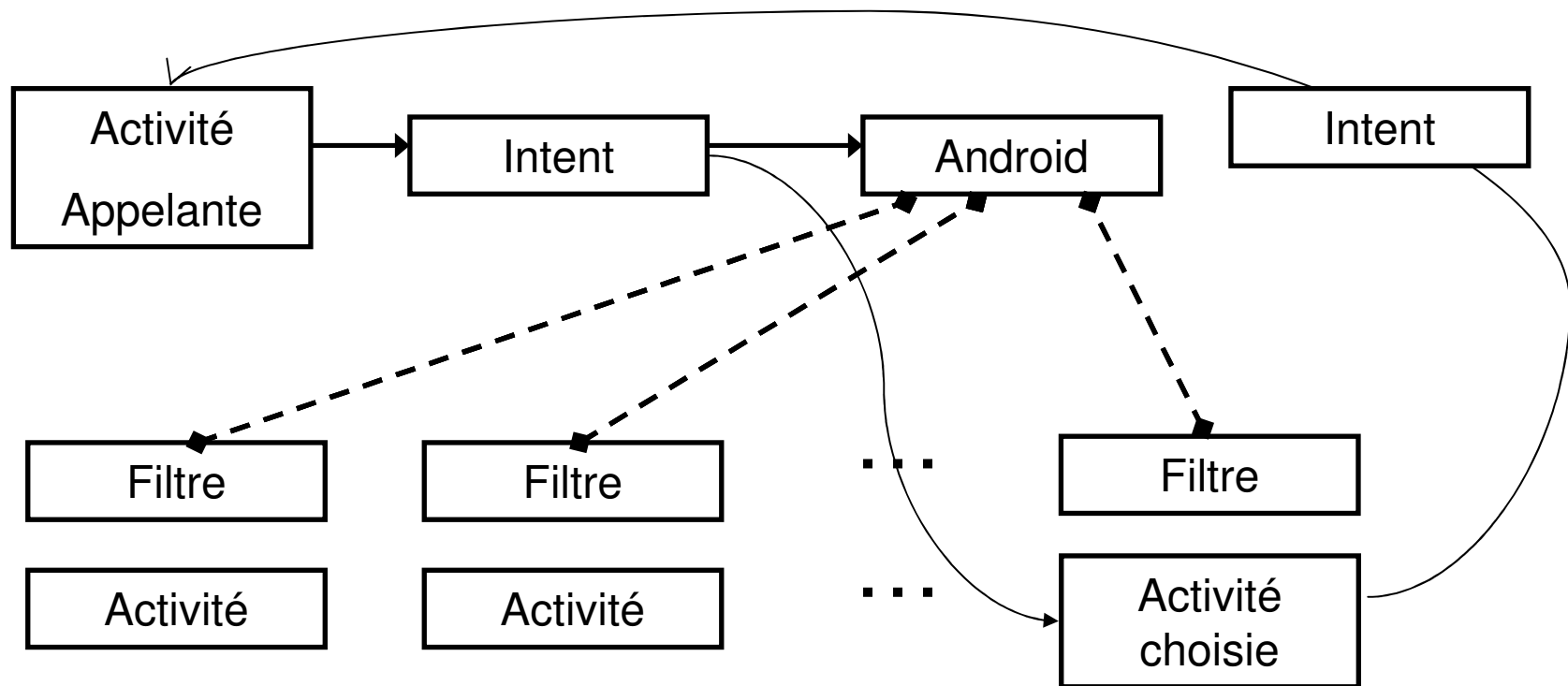
Cette activité doit être déclarée dans le fichier [AndroidManifest.xml](#) de l'application par une balise `<activity android:name=".classe">`

- Mode implicite :

On décrit l'activité à lancer et Android recherche une activité correspondant à cette description (par exemple un navigateur web)

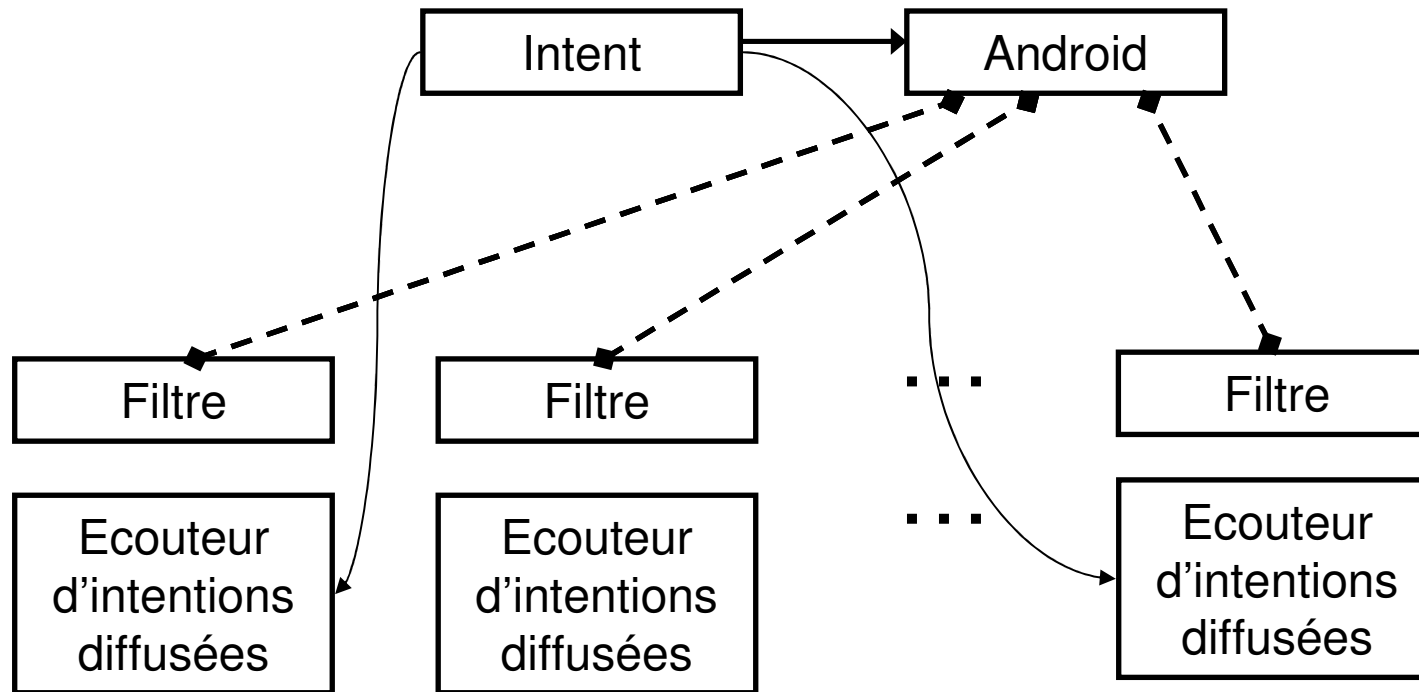
On peut éventuellement passer des paramètres et récupérer des valeurs de retour de l'activité lancée

# Invocation implicite



- Android choisit l'activité à lancer en fonction de l'**Intent** émis par l'activité appelante et des **filtres** associés aux activités présentes
- L'activité choisie reçoit cet Intent
- L'activité choisie renvoie un Intent à l'appelante lorsqu'elle se termine

# Intention diffusée



- Android envoie l'**Intent** à toutes les applications ayant un écouteur d'intentions diffusées associé à des **filtres** correspondants à cet **Intent**

# L'Intent

- Informe sur les intentions de l'activité ou du service recherché par 3 éléments :
  - Action  
Chaîne de caractères qui indique le type d'action recherché (par exemple appel téléphonique)
  - Catégorie  
Chaîne de caractères qui indique la catégorie de l'activité recherchée (par exemple CATEGORY\_LAUNCHER indique une activité proposée comme exécutable par Android (icône de lancement))
  - Donnée ou Type  
Chaîne de caractères et Uri qui indiquent la donnée transmise à l'activité lancée (par exemple le n° de téléphone) ou le type de réponse attendu
- Peut contenir les paramètres passés à l'activité lancée
- Peut contenir les valeurs renvoyées par l'activité lancée

# Méthodes de la classe Intent

- Construction
  - [Intent\(String\)](#) : avec action
  - [Intent\(String, Uri\)](#) : avec action et Uri
- Ajout de catégorie
  - [addCategory\(String\)](#) ajoute une catégorie
  - [setDataAndType\(Uri, String\)](#) définit l'Uri et le type mime des données
- Comparaison
  - [filterEquals\(Intent\)](#) renvoie true si le paramètre correspond au même filtre
- Contenu
  - [getAction\(\)](#) renvoie l'action (String)
  - [getCategories\(\)](#) renvoie les catégories (collection de String)
  - [getData\(\)](#) renvoie l'Uri correspondant aux données (Uri)
  - [getType\(\)](#) renvoie le type mime des données (String)
- Paramètres
  - [putExtra\(nom, valeur\)](#) ajoute un paramètre associé à un nom
  - [getxxxExtra\(nom\)](#) renvoie le paramètre correspondant au nom (xxx dépend du type de paramètre : Int, String, StringArray ...)



# Filtres d'intentions

- Dans le **AndroidManifest.xml** à chaque activité ou service est associé un ou plusieurs filtres d'intentions qui permettent à Android de choisir une activité (en mode implicite)

- 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://hôte:port/chemin" />
 />
 </intent-filter>
 ...
 <intent-filter>
 ...
 </intent-filter>
</activity>
```

# Filtrage d'intentions

- En mode explicite il n'y a aucun filtrage
  - L'objet `Intent` de l'appelant est transmis à l'appelé
- En mode implicite Android utilise les informations contenues dans l'objet `Intent` de l'appelant pour les confronter aux filtres définis par les activités connues dans leur fichiers `AndroidManifest`. Ces filtres définissent les capacités de l'activité en termes :
  - D'actions : **l'une des actions** indiquées dans l'`Intent` doit correspondre à **l'une de celles du filtre**
  - De catégorie : **toutes les catégories** indiquées dans l'`Intent` doivent apparaître dans le filtre
  - De données : le type de données indiqué dans l'`Intent` doit correspondre à celui du filtre

# Quelques valeurs prédéfinies

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

En gras le cas d'une activité principale d'application

La plupart des valeurs prédéfinies correspondent à des activités disponibles sur Android (appel téléphonique, navigateur ...)

# Exemple

- On a écrit une application contenant une activité de traduction français/basque
- Cette activité affiche 2 zones de texte et un bouton : on tape le texte à traduire dans la 1<sup>ère</sup> zone quand on clique le bouton la traduction s'affiche dans la 2<sup>ème</sup> zone
- On prévoit que l'activité puisse démarrer avec un texte à traduire qu'elle reçoit dans un Bundle (on verra plus tard comment faire)
- On la dote d'un filtre avec :
  - Action = `android.intent.action.VIEW`
  - Catégorie = "Traduction FR-BSQ"
  - Type mime de données = "text/plain"
- Un développeur ayant cette application installée pourra lancer une traduction depuis une application en préparant un `Intent` réunissant ces informations et en y ajoutant la chaîne à traduire : Android la trouvera pour lui et la lancera
- Si cette application a prévu de renvoyer la chaîne traduite quand elle se termine, l'autre application pourra la récupérer et l'utiliser.

# Lancer une activité

- Lancer explicitement une activité

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

- Lancer implicitement une activité

– Exemple : lancer un navigateur sur une page :

```
Uri chemin = Uri.parse("http://www.google.fr");
Intent naviguer = new Intent(Intent.ACTION_VIEW, chemin);
startActivity(naviguer);
```

– Exemple : appeler un n° de téléphone :

```
Uri numero = Uri.parse("tel:0559574320");
Intent appeler = new Intent(Intent.ACTION_CALL, numero);
startActivity(appeler);
```

# Lancer une activité et obtenir un retour

- Lancement (dans l'activité A)

```
static final int MON_CODE = 1; // code servant à identifier l'activité qui répond
Intent démarre = new Intent(this, NomDeLaClasseDeLActiviteB.class);
// ajouter les paramètres passés à B dans l'Intent démarre
startActivityForResult(intention, MON_CODE); // lancement de l'activité B
```

- Renvoi du code et des valeurs de retour (dans 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
setResult(code, intent_retour); // renvoyer un code de retour (entier) et l'Intent de retour
finish(); // terminer l'activité B
```

- Traitement du code de retour (dans l'activité A)

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

# Passer des paramètres à l'activité appelée

- La classe `Intent` permet de passer des paramètres à l'activité appelée et d'en récupérer les valeurs en retour
- Ajouter des paramètres (types simples ou tableaux)  
Objet\_intent.putExtra(String, val)  
Le 1<sup>er</sup> paramètre est un nom (clé)  
Le second paramètre est la valeur :
  - De type simple (boolean, int, short, long, float, double, char)
  - Tableau de types simples
  - String et tableau de String
- L'activité appelée pourra récupérer ces paramètres par leur nom :

# Récupérer les paramètres dans l'activité appelée

- L'activité lancée récupère l'objet de classe `Bundle` contenant les paramètres par :  
`Bundle params = getIntent().getExtras()`
- Les paramètres sont récupérés dans ce `Bundle` par ses méthodes :
  - `getBoolean(String)`
  - `getInt(String)`
  - ...
  - `getBooleanArray(String)`
  - ...auxquelles on passe la clé en paramètre



# Placer des valeurs de retour dans l'activité appelée

- Le méthode `setResult(int, Intent)` permet de renvoyer un code de retour et un Intent de retour
- L'activité appelée place les valeurs de retour dans cet Intent par `putExtra(String, val)` comme déjà vu pour les paramètres
- L'activité appelante récupère cet Intent comme dernier paramètre de la méthode :  
`onActivityResult(int req, int code_retour, Intent retour)`
- Elle en extrait les paramètres par les méthodes de la classe Intent :
  - `getBooleanExtra(String)`
  - `getIntExtra(String)`
  - ...
  - `getBooleanArrayExtra(String)`
  - ...

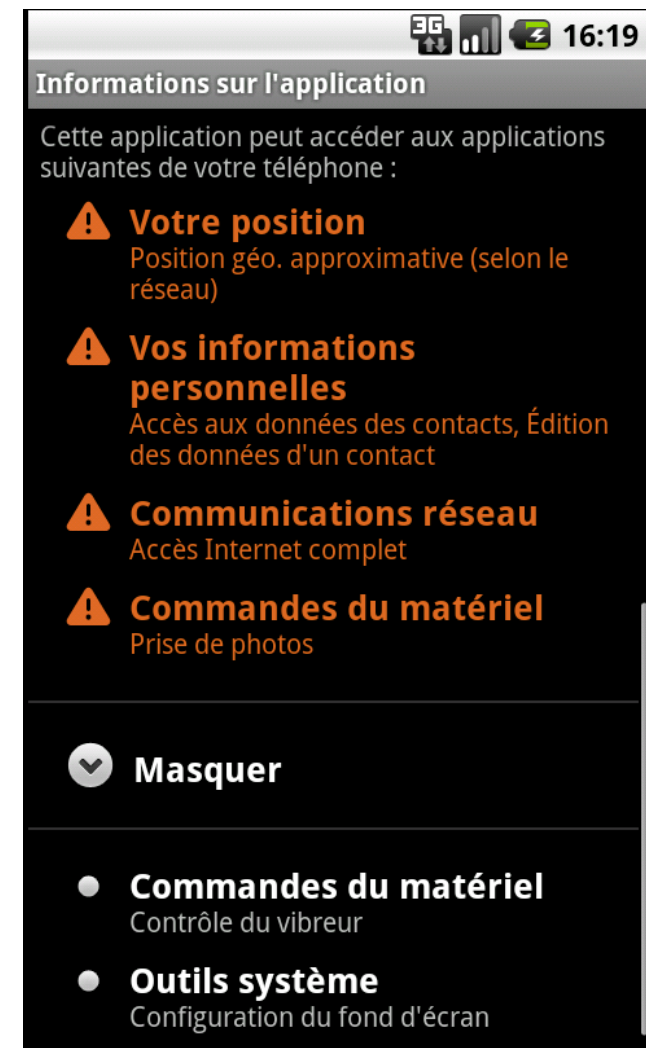
Permissions d'une activité

# Permissions

- Une activité ne peut accéder à certaines ressources matérielles qu'avec une permission
- Les permissions apparaissent dans le `AndroidManifest`
- Elles sont visibles par l'utilisateur (sécurité)
- Elles concernent :
  - La géolocalisation (GPS)
  - Les accès aux contacts et à l'agenda du téléphone
  - Les modifications de paramètres (orientation, fond d'écran ...)
  - Les appels téléphoniques
  - L'envoi et réception de SMS/MMS
  - L'audio
  - Le réseau (dont l'accès à Internet)
  - Le matériel (bluetooth, appareil photo, ...)

# Surveiller vos applications

- Il est prudent de regarder **quelles permissions demande une application**
- On peut le faire par Paramètres → Applications → Gérer les applications puis clic sur l'application
- Certaines permissions peuvent être **dangereuses** pour :
  - Votre forfait (envoi de SMS/MMS, appels)
  - Votre vie privée (consultation/modification des données personnelles, position)
  - Votre appareil (modifications de paramètres)



# Permissions dans AndroidManifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
 <uses-sdk />
 <uses-permission android:name="android.permission.CALL_PHONE" />

 <uses-permission android:name="android.permission.INTERNET " />
 <application>
 <activity>
 <intent-filter>
 <action />
 <category />
 <data />
 </intent-filter>
 </activity>
 ...
 </application>
</manifest>
```

## ATTENTION

**L'oubli de permissions provoquera  
une erreur d'exécution**

# Le matériel et les capteurs

# Téléphonie (appel)

- Permettre à l'utilisateur d'appeler un n° composé

```
Uri numero = Uri.parse("tel:0559574320");
Intent composer = new Intent(Intent.ACTION_DIAL, numero);
startActivity(composer);
```

Il faut avoir positionné la permission :

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

- Appeler directement un n°

```
Uri numero = Uri.parse("tel:0559574320");
Intent appeler = new Intent(Intent.ACTION_CALL, numero);
startActivity(appeler);
```

Il faut avoir positionné la permission :

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

# Téléphonie (envoi de SMS)

- Classe `SmsManager` dont une instance est obtenue par : `SmsManager.getDefault()`
- Envoi d'un message par : `sendTextMessage` en précisant le n° et le texte

Il faut avoir positionné la permission :

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



# Téléphonie (Réception de SMS)

1. Déclaration d'un écouteur d'intentions diffusées

```
<receiver android:name=".receiver.monRecepteur" android:enabled="true">
 <intent-filter>
 <action android:name="android.provider.Telephony.SMS_RECEIVED" />
 </intent-filter>
</receiver>
```

2. Ecriture de l'écouteur d'intention diffusées par héritage de `BroadcastReceiver` et surcharge de la méthode `onReceive`
3. L'Intent reçu en paramètre de `onReceive` contient les messages reçus sous forme brute désignés par la clé "pdu"
4. Ces message bruts peuvent être convertis en objets de classe `SmsMessage` par la méthode `createFromPdu` de cette classe.
5. Un `SmsMessage` permet de récupérer l'expéditeur, le corps du message ainsi que l'expéditeur et le corps d'un mail

Il faut avoir positionné la permission :

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

# Géolocalisation par GPS

1. Accés au GPS par :  
`getSystemService(Context.LOCATION_SERVICE)`
2. Associer un écouteur d'événements par :  
`requestLocationUpdate` en précisant :
  - Le mode (GPS ou réseau)
  - Le rythme
  - La distance minimale
3. Récupérer les informations dans l'écouteur
  - Latitude, longitude, altitude
  - Précision
4. Possibilité de calculer une distance

Il faut avoir positionné les permissions :

```
<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" />
```

# Appareil photo

- La classe `Camera` permet la prise de photo par `takePicture` en associant un écouteur d'événement pour récupérer la photo en raw ou JPEG. La méthode `onPictureTaken` de cet écouteur est appelé quand la photo est faite, l'image est reçue en tableau d'octets.
- Nécessite une prévisualisation par un objet de classe `SurfaceView` dans l'interface auquel on associe un écouteur d'événements pour :
  - Démarrer/arrêter l'appareil photo (méthodes `open` et `release` de la classe `Camera`)
  - Lancer/arrêter la prévisualisation (méthodes `startPreview` et `stopPreview` de la classe `Camera`)
- On peut enregistrer le tableau d'octets reçu par `onPictureTaken` dans un fichier ou utiliser `BitmapFactory` pour le convertir en image

Il faut avoir positionné la permission :

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

# Vibreur (classe `Vibrator`)

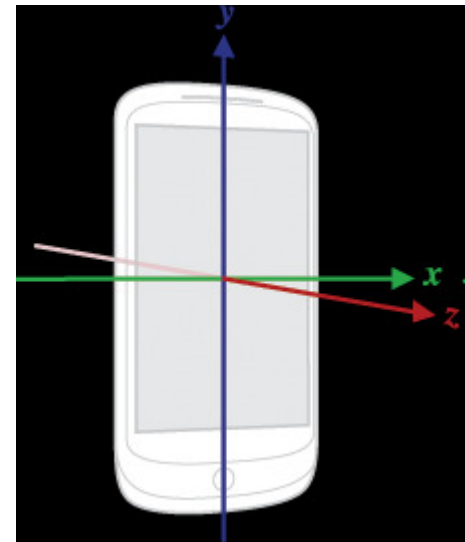
- Peut être utilisé pour alerter l'utilisateur (par exemple dans un service ou un écouteur d'intentions diffusées qui n'a pas d'interface)
- Accès au vibreur par :  
`getSystemService(Context.Vibrator_SERVICE)`
- Faire vibrer par :  
La méthode `vibrate(int)` de la classe `Vibrator`  
En donnant la durée en ms

Il faut avoir positionné la permission :

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

# Capteurs

- Les types de capteurs disponibles sont les suivants (selon le modèle certains capteurs peuvent ne pas être disponibles) :
  - Accéléromètre (accélération du périph sur 3 axes)
  - Gravité (composante de la gravité selon les 3 axes)
  - Gyroscope (vitesse angulaire de rotation du périph sur 3 axes)
  - Lumière (luminosité ambiante)
  - Champ magnétique (champ magnétique ambiant sur 3 axes)
  - Orientation (angles du périph par rapport aux 3 axes)
  - Pression (pression exercée sur l'écran tactile)
  - Proximité (détection de proximité souvent binaire)
  - Température (température ambiante)



# Utilisation des capteurs

- Classe `SensorManager` dont une instance est obtenue par :  
`SensorManager gestionnaireCapteurs =  
 (SensorManager) getSystemService(Context.SENSOR_SERVICE);`
- Récupération d'un capteur par :  
`Sensor monCapteur =  
 gestionnaireCapteurs.getDefaultSensor(type_de_capteur);`

Avec `type_de_capteur` :

- `Sensor.TYPE_ACCELEROMETER`
- `Sensor.TYPE_GRAVITY`
- `Sensor.TYPE_GYROSCOPE`
- `Sensor.TYPE_LIGHT`
- `Sensor.TYPE_MAGNETIC_FIELD`
- `Sensor.TYPE_ORIENTATION`
- `Sensor.TYPE_PRESSURE`
- `Sensor.TYPE_PROXIMITY`
- `Sensor.TYPE_TEMPERATURE`

# Mesures par capteurs

- Association d'un écouteur d'événements par :  
gestionnaireCapteurs.registerListener([SensorEventListener](#),  
[Sensor](#), int)
  - Interface [SensorEventListener](#) surcharge des méthodes :
    - void [onSensorChanged\(SensorEvent\)](#) exécutée chaque fois que le capteur effectue une nouvelle mesure.
    - void [onAccuracyChanged\(Sensor, int\)](#) exécutée si la précision du capteur change.
  - Capteur auquel est associé l'écouteur
  - Rythme des mesures :
    - [SENSOR\\_DELAY\\_NORMAL](#)
    - [SENSOR\\_DELAY\\_UI](#) (adapté pour interfaces)
    - [SENSOR\\_DELAY\\_GAME](#) (adapté pour jeux)
    - [SENSOR\\_DELAY\\_FASTEST](#).

# Récupération des mesures

- Par le paramètre de classe [SensorEvent](#) de [onSensorChanged](#)
- [SensorEvent](#) a une propriété [values](#) dont on fait une copie par [values.clone\(\)](#) le résultat est un tableau de 3 éléments (float)

Selon le cas une seule ou les 3 valeurs sont significatives

Remarque : la copie permet d'éviter que les valeurs ne soient modifiées par une nouvelle mesure

- [SensorEvent](#) a également :
  - une propriété [timeStamp](#) qui indique l'instant de la mesure
  - une propriété [accuracy](#) qui indique la précision de la mesure :
    - [SENSOR\\_STATUS\\_ACCURACY\\_HIGH](#)
    - [SENSOR\\_STATUS\\_ACCURACY\\_LOW](#)
    - [SENSOR\\_STATUS\\_ACCURACY\\_MEDIUM](#)



# Multimédia

# Audio

- Créer un **MediaPlayer** :  
`MediaPlayer lecteur = MediaPlayer.create(Context, int)`  
Le premier paramètre est l'activité elle-même  
Le second paramètre est l'identificateur du fichier son obtenu par :  
`R.raw.nom_du_fichier_son`
- Utiliser le **MediaPlayer** :
  - `lecteur.start()` pour jouer le son
  - `lecteur.pause()` pour suspendre le son, il sera repris par `start()`
  - `lecteur.stop()` pour arrêter le son, il sera repris par :
    - `lecteur.reset();`
    - `lecteur.prepare();`
    - `lecteur.start();`

# MediaPlayer (utilisation)

- **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
- [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)

- **Contrôle**

- [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

# MediaPlayer (événements)

- **Evénements**

- [setOnCompletionListener\(MediaPlayer.OnCompletionListener\)](#) associe un écouteur d'événements
  - [onCompletion\(MediaPlayer\)](#) appelée lorsque le média se termine
- [setOnBufferingUpdateListener\(MediaPlayer.OnBufferingUpdateListener\)](#) associe un écouteur d'événements
  - [onBufferingUpdate\(MediaPlayer, int\)](#) 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
  - [onPrepared\(MediaPlayer\)](#) appelée lorsque le MediaPlayer est prêt.
- [setOnSeekCompleteListener\(MediaPlayer.OnSeekCompleteListener\)](#) associe un écouteur d'événements
  - [onSeekComplete\(MediaPlayer\)](#) appelée lorsque déplacement dans le média est terminé.

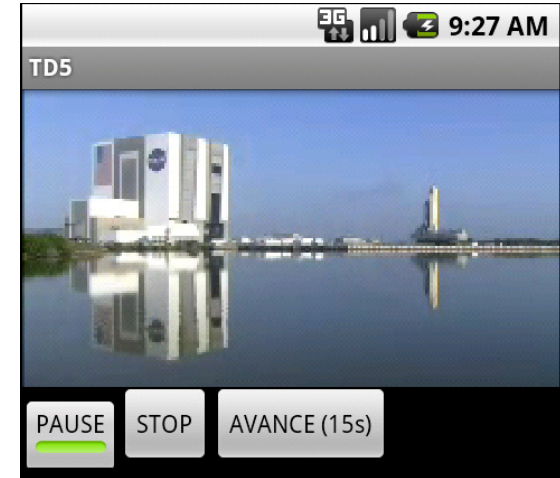
# Vidéo

- Mettre un `VideoView` dans l'interface

```
<VideoView android:id="@+id/ecran_video"
 android:layout_width="fill-parent"
 android:layout_height="fill-parent" />
```
- Définir le chemin de la vidéo (placée dans res/raw)

```
Uri chemin = Uri.parse("android.resource://paquetage_de_l_application/"
 + R.raw.nom_du_fichier_video);
```
- Associer un lecteur vidéo à la vue:

```
VideoView lecteur = (VideoView) findViewById (R.id.ecran_video);
lecteur.setVideoURI(chemin);
lecteur.setMediaController(new MediaController(activité)); // facultatif
lecteur.requestFocus();
```
- Si on a mis `setMediaController`, lors d'un clic long sur la vue une fenêtre de contrôle apparaît avec :
  - Un bouton Play/pause
  - Un bouton Avance rapide
  - Un bouton Recul rapide
  - Un curseur indiquant la position courante et permettant de se déplacer
- Sinon, et de toutes façons, on peut tout contrôler par programme :



# VideoView

- **Configuration**
  - [`setMediaController\(MediaController\)`](#) associe un contrôleur de média
  - [`setVideoPath\(String\)`](#) définit le média par un chemin de fichier
  - [`setVideoURI\(Uri\)`](#) définit le média par une Uri
- **Contrôle**
  - [`start\(\)`](#) lancement
  - [`pause\(\)`](#) mise en pause, reprise par `start()`
  - [`seekTo\(int\)`](#) déplacement 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
- **Événements**
  - [`setOnCompletionListener\(MediaPlayer.OnCompletionListener\)`](#) associe un écouteur d'événements
    - méthode [`onCompletion\(MediaPlayer\)`](#) appelée lorsque le média se termine.

# Synthèse de parole

- Créer un synthétiseur :  
parle = new `TextToSpeech`(activité, new SynthPret());
- Initialiser le synthétiseur quand il est prêt par l'écouteur d'événements  
private class SynthPret implements `TextToSpeech.OnInitListener` {  
    public void `onInit`(int etat) {  
        if (etat == `TextToSpeech.SUCCESS`) {  
            parle.`setLanguage`(`Locale.FRANCE`); // langue  
        }  
    }  
}
- Synthétiser un texte :  
parle.`speak`("texte a dire", `TextToSpeech.QUEUE_FLUSH`, null);
- Ajouter du texte à synthétiser pendant la synthèse :  
parle.`speak`("texte a ajouter", `TextToSpeech.QUEUE_ADD`, null);
- Arrêter la synthèse de son :  
parle.`stop`(); // arrêt de la synthèse

# TextToSpeech

- 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)
- Arrêt du synthétiseur de son :  
`parole.shutdown();` // arrêt du synthétiseur
- Fermeture du synthétiseur dans la méthode `onDestroy` de l'activité :

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



Internet

# Ressources sur Internet

- Images

- Créer une connexion sur la ressource et se connecter :

```
HttpURLConnection connexion = (HttpURLConnection) new URL("url de
l'image").openConnection();
connexion.connect();
```

- Créer un flux de lecture sur la ressource :

```
BufferedInputStream lecture = new BufferedInputStream
(connexion.getInputStream());
```

- Lire la ressource et la transformer en `Drawable` de type image :

```
BitmapDrawable img;
BitmapFactory.Options opts = new BitmapFactory.Options();
opts.inSampleSize = x; // pour réduire la taille en 1/x
img = BitmapFactory.decodeStream(lecture, null, opts);
```

# Ressources sur Internet

- Son :

On utilise toujours un MediaPlayer mais créé différemment :

```
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'accès à l'URL provoque une erreur
}
```

- Vidéo :

La seule chose qui change c'est l'**Uri** qui désigne le média

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

Services WEB

# Géolocalisation par service WEB

La classe **Geocoder** permet de retrouver des adresses (type adresse postale) à partir :

- De coordonnées GPS
- D'un nom (par exemple : "parc Montaury, Anglet")

La recherche peut être limitée à un pays ou à une zone géographique

Utilise la classe **Address** qui contient :

- Coordonnées GPS
- Nom de pays
- Nom de ville
- Code postal
- Adresse complète (n°, rue, ...)

Il faut avoir positionné la permission :

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

# Localisation

## – Création

[Geocoder](#)(activité, [Locale](#)) le second paramètre indique la zone géographique concernée, il peut prendre les valeurs ([Locale.FRANCE](#), [Locale.CANADA](#), [Locale.UK](#), [Locale.US](#) ...). Omis si l'on ne souhaite pas limiter la localisation.

## – Recherches

- [getFromLocation](#)(double, double, int) renvoie les adresses connues proches du point défini par ses coordonnées géographiques (latitude et longitude exprimées en degrés). Le dernier paramètre permet de limiter la taille de la liste.
- [getFromLocationName](#)([String](#), int) renvoie les adresses connues proches d'un point défini par un nom (chaîne du type "parc Montauray, 64600, Anglet"). Le second paramètre permet de limiter la taille de cette liste.
- [getFromLocationName](#)([String](#), int, double, double, double, double) fonctionne comme la précédente mais permet de limiter la zone de recherche à un rectangle. longitude et latitude du coin inférieur gauche de ce rectangle et longitude et latitude du coin supérieur droit.

Toutes ces méthodes renvoient une liste (classe [List](#) de java) contenant des objets de classe [Address](#)

# Exemple d'utilisation du service de localisation

- Le code suivant :

```
Geocoder localisateur = new Geocoder(this, 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

# La classe Address

- **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)
- [getLongitude](#)() renvoie la longitude en degrés (réel)
- [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](#)()

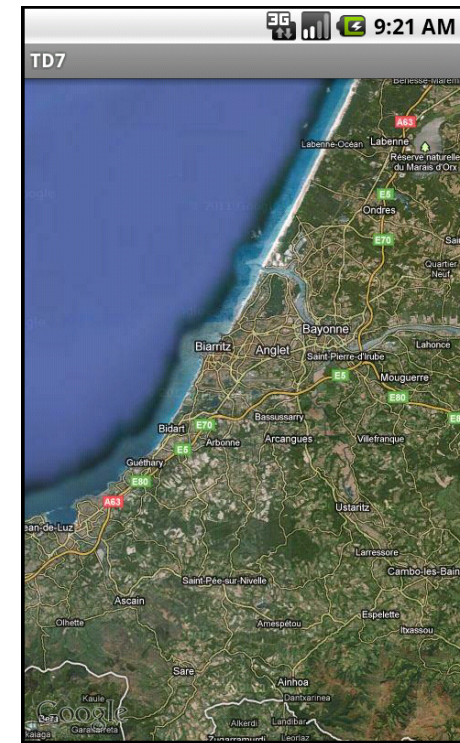


# Google Maps (prérequis)

- Le SDK utilisé doit être "Google API ..."
- Le fichier `AndroidManifest` doit intégrer la bibliothèque Google Maps :  
`<uses-library android:name="com.google.android.maps" />`
- L'activité doit hériter de `MapActivity`
- Une seule carte est possible par activité
- L'activité doit surcharger la méthode :  

```
protected boolean isRouteDisplayed() {
 return false;
}
```

Utilisée pour les statistiques de Google
- Il faut obtenir une **clé d'utilisation** auprès de Google :
  - Avoir un compte Google
  - Obtenir la signature de l'application par la commande **keytool** de java
  - Récupérer la clé avec cette signature sur :  
<http://code.google.com/intl/fr/android/maps-api-signup.html>



# Google Maps (affichage)

- La carte est affichée dans un widget **MapView** :  

```
<com.google.android.maps.MapView android:id="@+id/carte"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent" android:apiKey="mettre ici la
clé obtenue" />
```
- **MapView** permet de définir l'apparence de la carte :
  - [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 trafic

# Google Maps (déplacements)

Les déplacements dans la cartes sont gérés par un [MapController](#)

- Obtention :  
par la méthode [getController\(\)](#) de l'objet [MapView](#)
- Utilisation :
  - [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 millièmes de degrés
  - [setCenter\(GeoPoint\)](#) positionne la carte sur le point passé en paramètre sans faire d'animation.
  - [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

# Google Maps (projection)

la classe `Projection` permet de récupérer les coordonnées géographiques d'un point de la carte affichée :

- Obtention :  
par la méthode `getProjection()` de l'objet `MapView`
- Utilisation :
  - `fromPixels(int, int)` renvoie un `GeoPoint` avec les coordonnées du point désigné par sa position en pixels sur la carte affichée (x et y)

ATTENTION : `GeoPoint` utilise des coordonnées géographiques entières en millièmes de degré alors que le GPS travaille en coordonnées réelles en degrés

Formatage du texte

# SpannableStringBuilder

- Les widgets contenant du texte héritent de `TextView`
- Les méthodes d'ajout de texte (`setText` et `append`) prennent en paramètre un `CharSequence`
- `String` hérite de `CharSequence` mais le format est alors celui associé au widget (dans le XML par exemple)
- Pour formater le texte on utilise `SpannableStringBuilder` qui hérite de `CharSequence` mais permet de définir zone par zone :
  - La police
  - La taille
  - L'étirement horizontal
  - La couleur du fond et du texte
  - Le style (gras, italique ...)
  - Le soulignement

# Texte formaté

- Création d'un texte formattable :  
`SpannableStringBuilder` texte = new `SpannableStringBuilder`("chaîne à formater");
- Association d'un format à une zone du texte :  
texte.`setSpan`(`CharacterStyle`, int, int, `Spannable.SPAN_EXCLUSIVE_EXCLUSIVE`);
- Les 2<sup>ème</sup> et 3<sup>ème</sup> paramètres de `setSpan` indiquent le point de départ et de fin de la zone la chaîne de caractères à laquelle le format est appliqué
- Format possibles (classes pour le premier paramètre) :
  - `ForegroundColorSpan` (int couleur)
  - `BackgroundColorSpan` (int couleur)
  - `AbsoluteSizeSpan` (int taille, boolean pixel\_a\_densité\_dépendante)
  - `RelativeSizeSpan` (double échelle)
  - `ScaleXSpan` (float échelle)
  - `UnderlineSpan` ()
  - `TypeSpaceSpan` (" {`normal`, `sans`, `serif`, `monospace` } ")
  - `StyleSpan` (android.graphics.Typeface.{`BOLD`, `BOLD_ITALIC`, `ITALIC`, `NORMAL` })

# Et le reste ?

Beaucoup de choses ont été vues mais beaucoup ne l'ont pas été :

- Ecriture de services
- Ecriture de fournisseurs de contenu
- Ecriture d'écouteurs d'intensions diffusées
- Interfaces à onglets ou à liste (une activité est associée à chaque onglet ou élément de la liste)
- Création de composants d'interface personnalisés
- Ecriture de pilotes permettant de modifier l'interface depuis l'extérieur de l'activité
- Animations des vues
- Persistance des données (SQLite)
- Communication par réseau (sockets , HTTP, SOAP, REST ...)
- Sérialisation (Parcelable , JSON)
- Graphisme en 2D et 3D (openGL)
- Utilisation du Bluetooth
- Internationalisation des applications
- ...