Développement d'applications pour Android

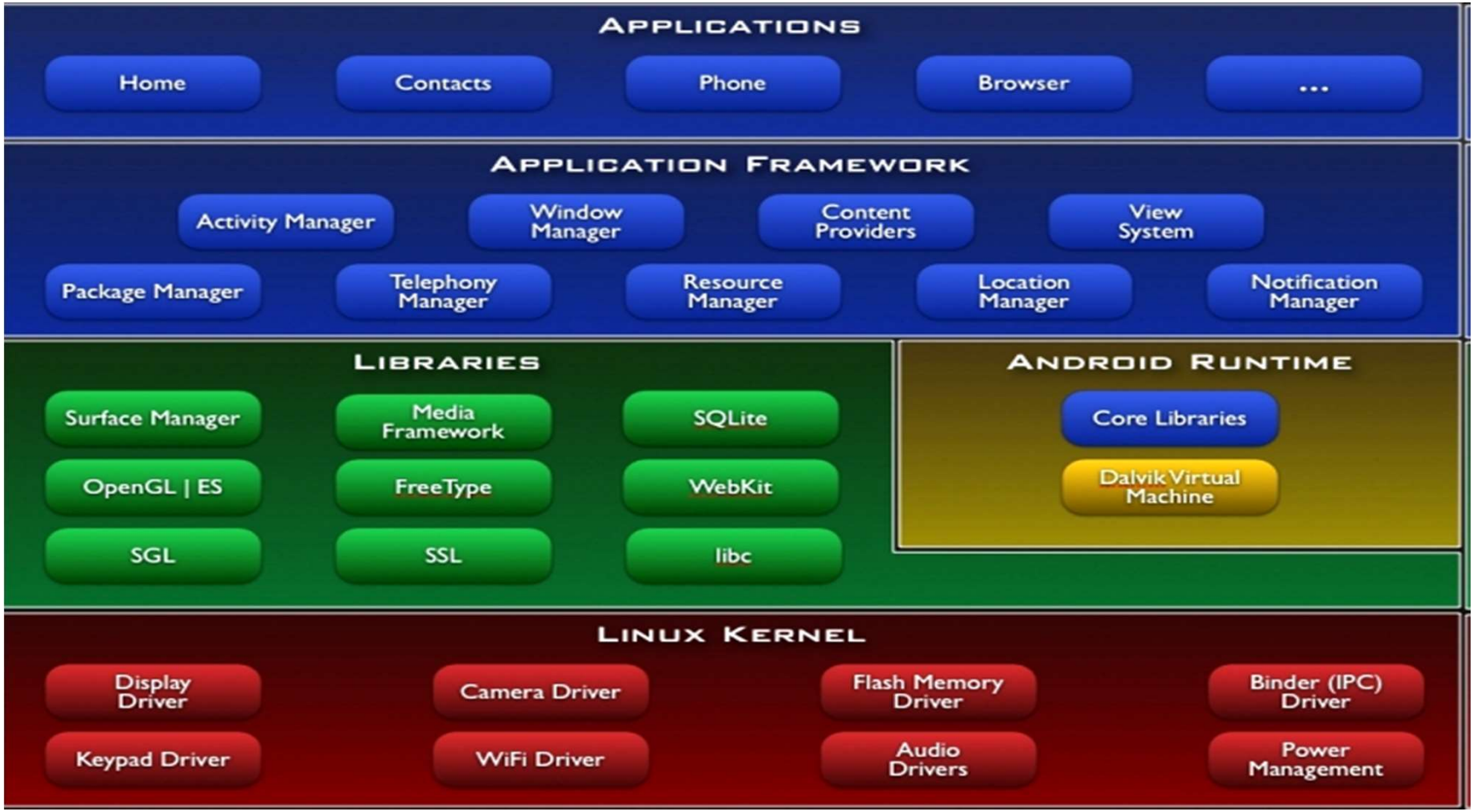
M. Dalmau – IUT de Bayonne – Pays Basque

Architecture matérielle

- Processeur
- Mémoire
- Processeur graphique
- Ecran tactile
- Stockage (flash, carte SD)
- Réseau (wifi, bluetooth, cellulaire)
- Connecteurs (USB, HDMI, …)
- Capteurs
  - Vidéo (1 ou 2)
  - Micro
  - GPS
  - Accéléromètre
  - Gyroscope
  - Lumière
  - Champ magnétique
  - Proximité
  - Pression
  - Température
  - …
- Actionneurs
  - Vibreur
  - Haut parleur/casque

Possibilité d'interfaces multimodales

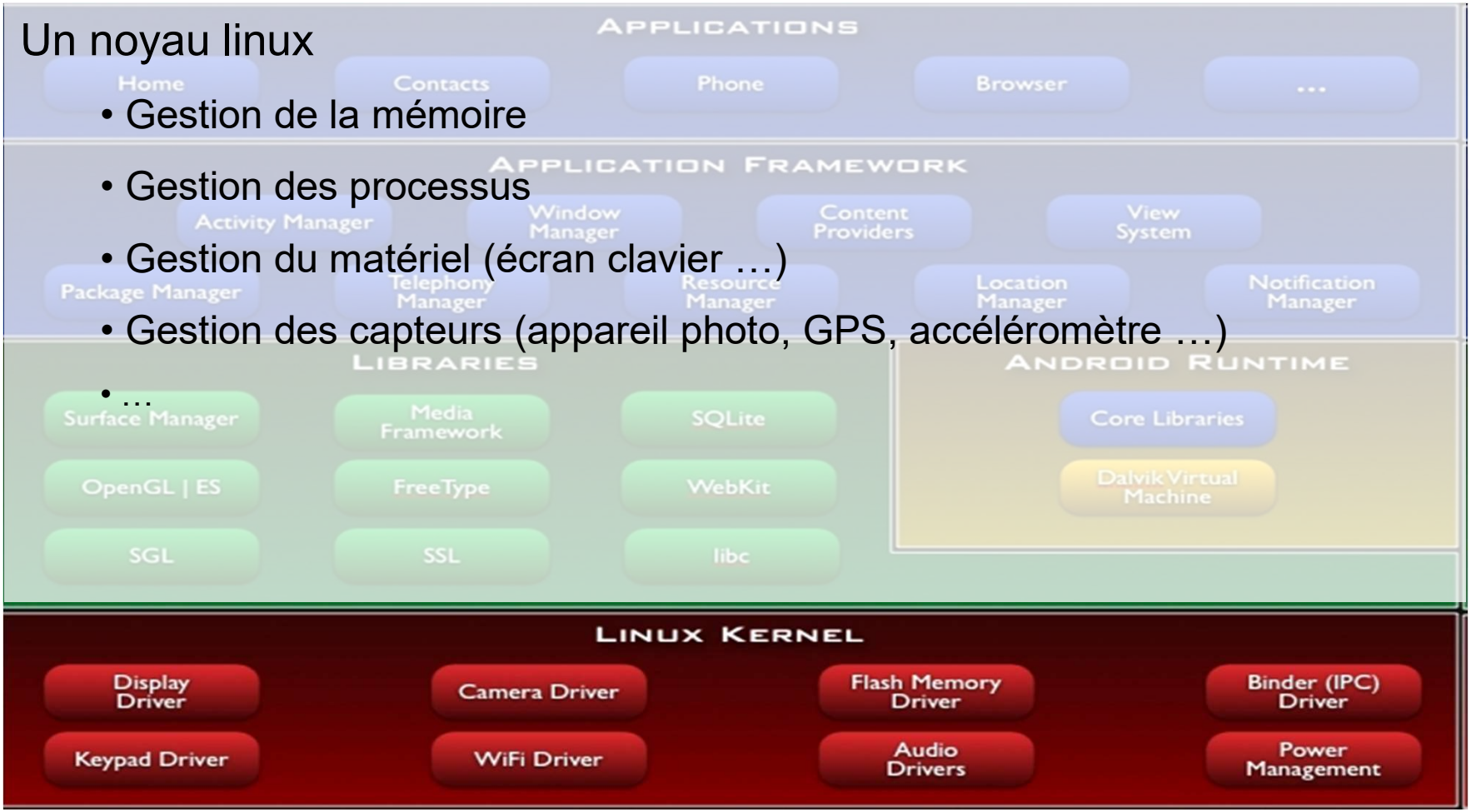
Architecture d'Android



Architecture d'Android

Un noyau linux

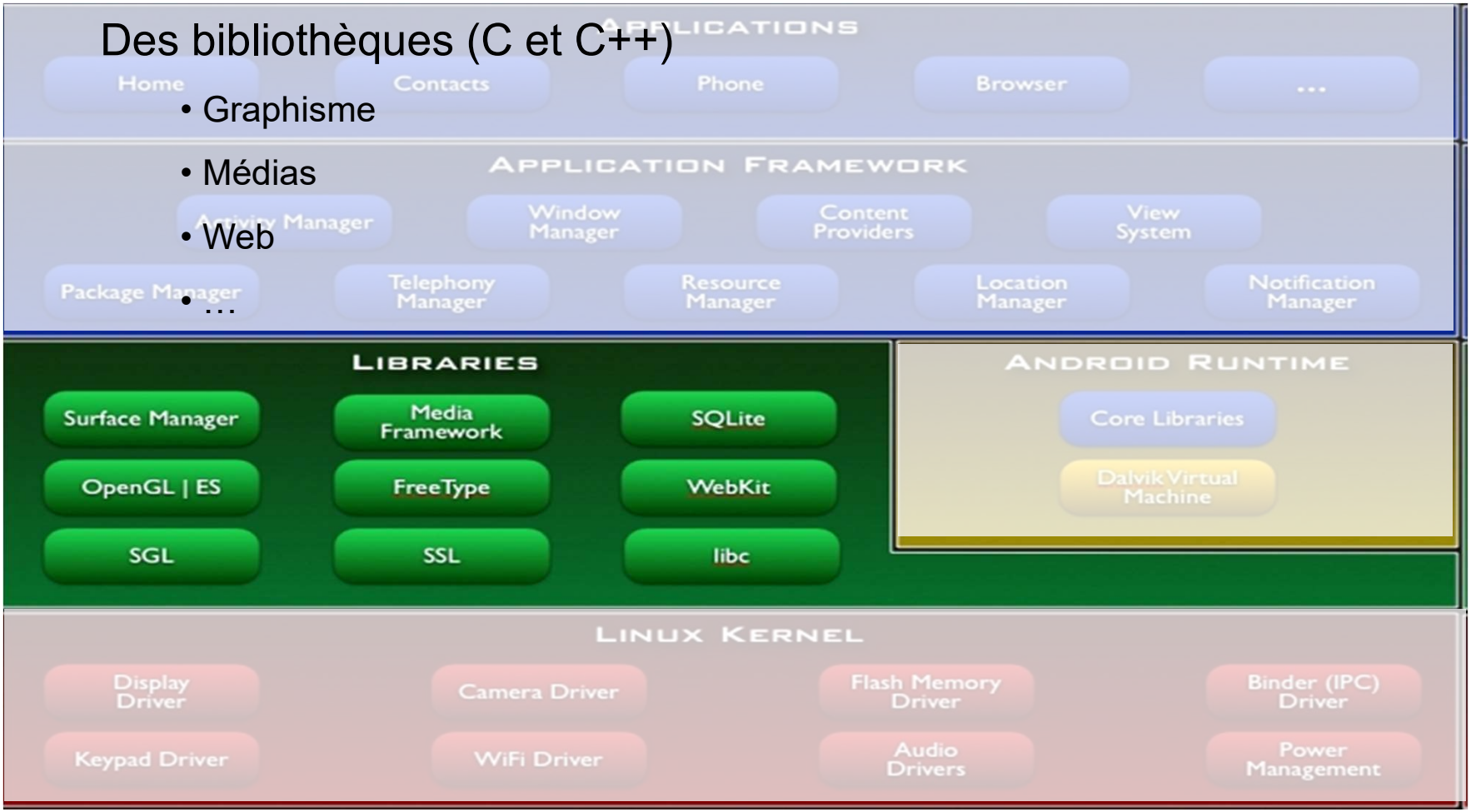
- Gestion de la mémoire
- Gestion des processus
- Gestion du matériel (écran clavier …)
- Gestion des capteurs (appareil photo, GPS, accéléromètre …)
- …



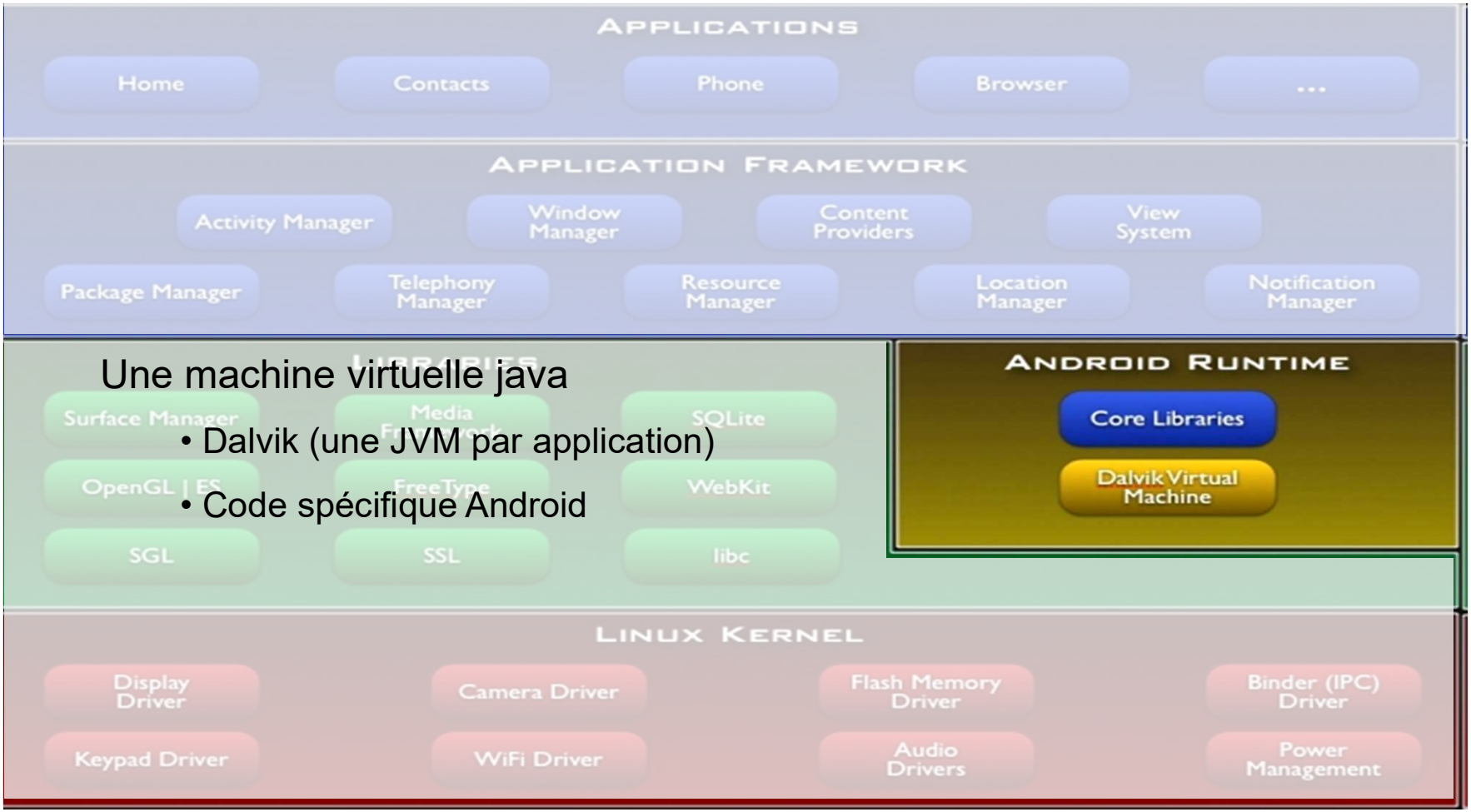
Architecture d'Android

Des bibliothèques (C et C++)

- Graphisme
- Médias
- Web
- …



Architecture d'Android



Une machine virtuelle java

- Dalvik (une JVM par application)
- Code spécifique Android

Architecture d'Android

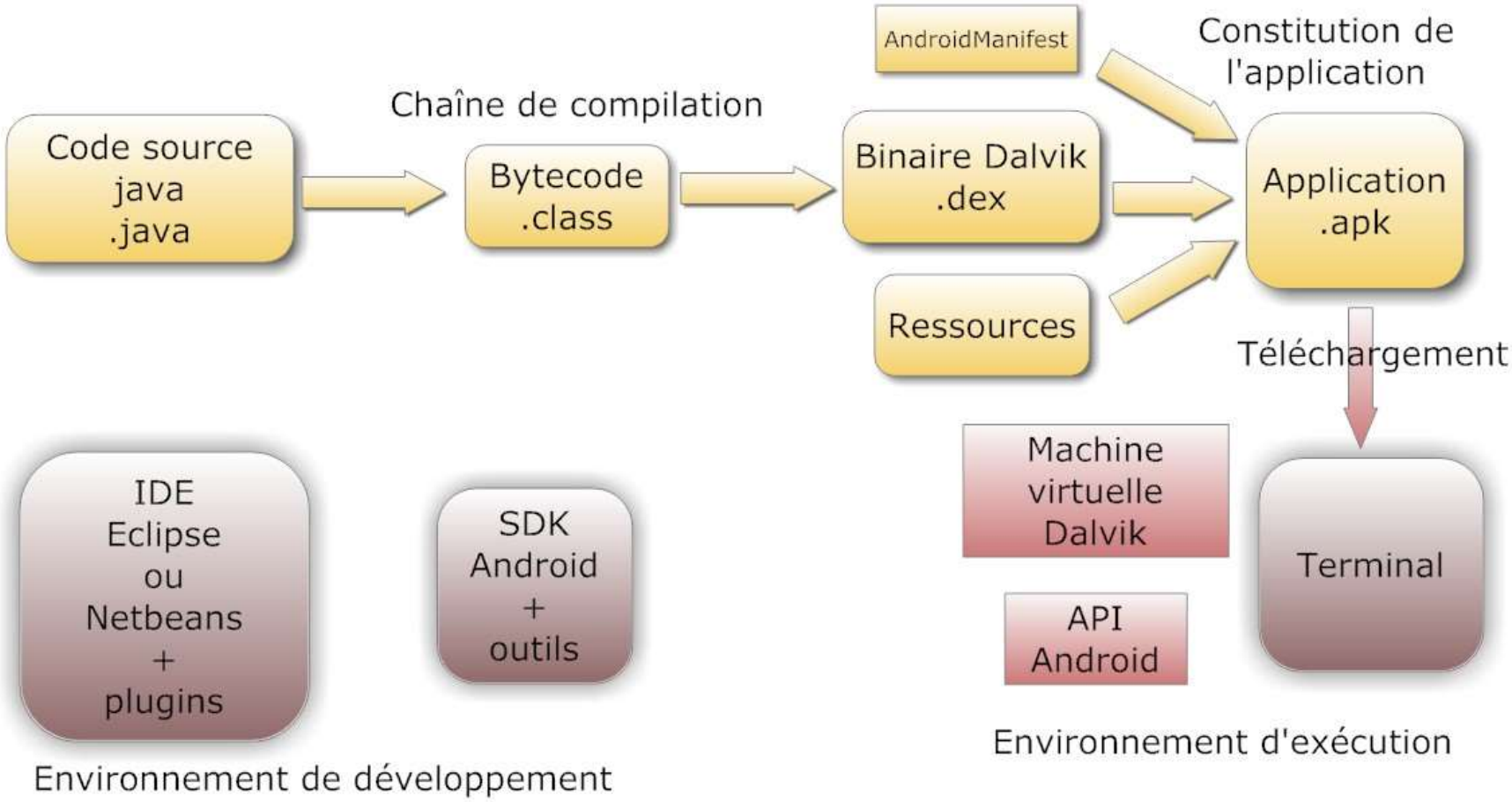
Des gestionnaires pour les applications + une API en java

- Gestion des fenêtres, des activités, des ressources …
- API pour développement des programmes

Architecture d'Android

Développement d’applications pour Android

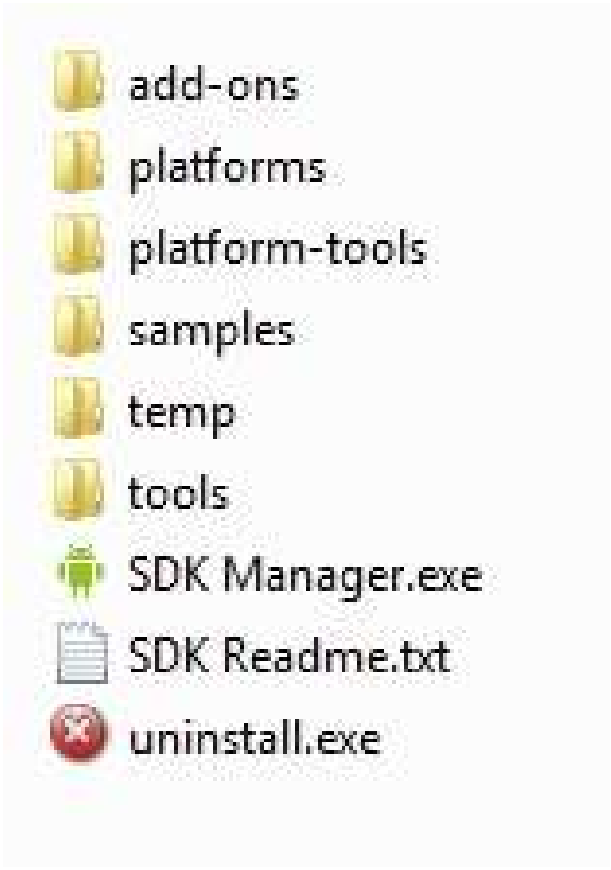
Production de logiciel



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

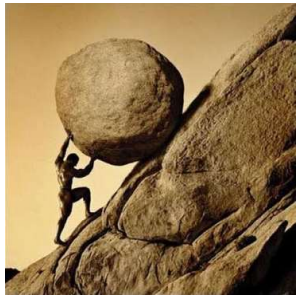
Accessibles à partir d'une ligne de commande (fenêtre DOS)

- **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 …)

Développement

- Directement avec un éditeur de texte et les outils du SDK



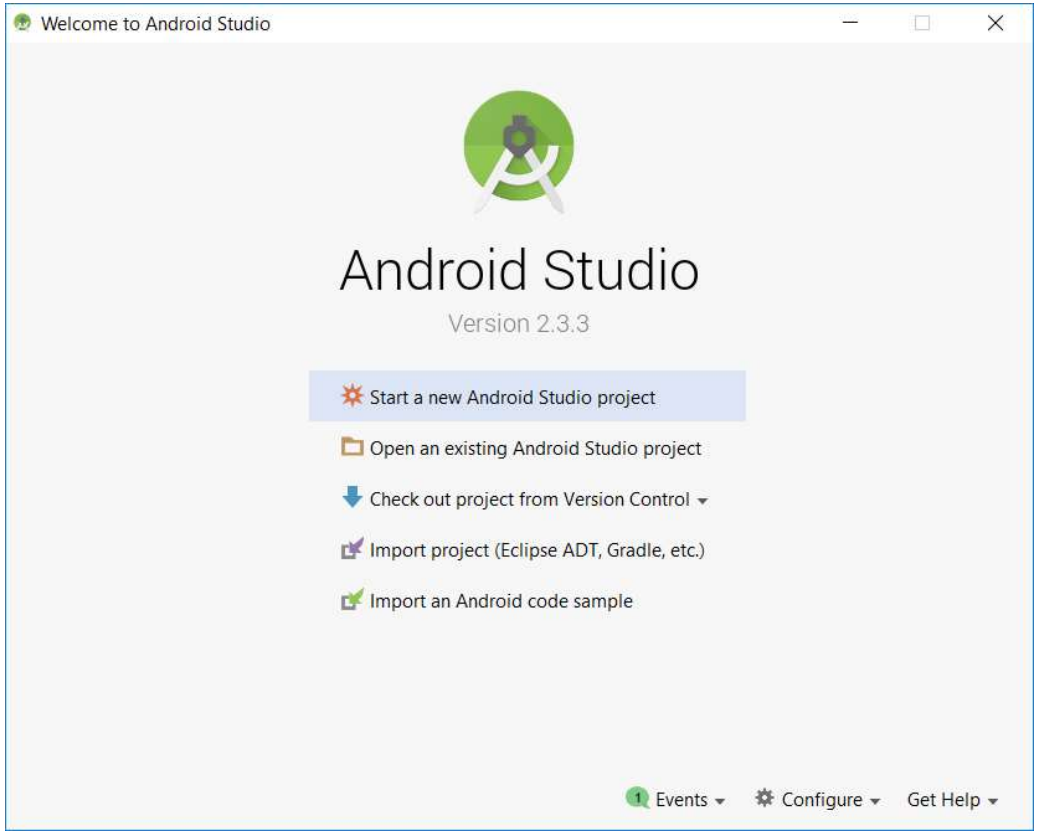
- Eclipse
  - Téléchargeable sur www.eclipse.org

- Android Studio
  - Téléchargeable sur

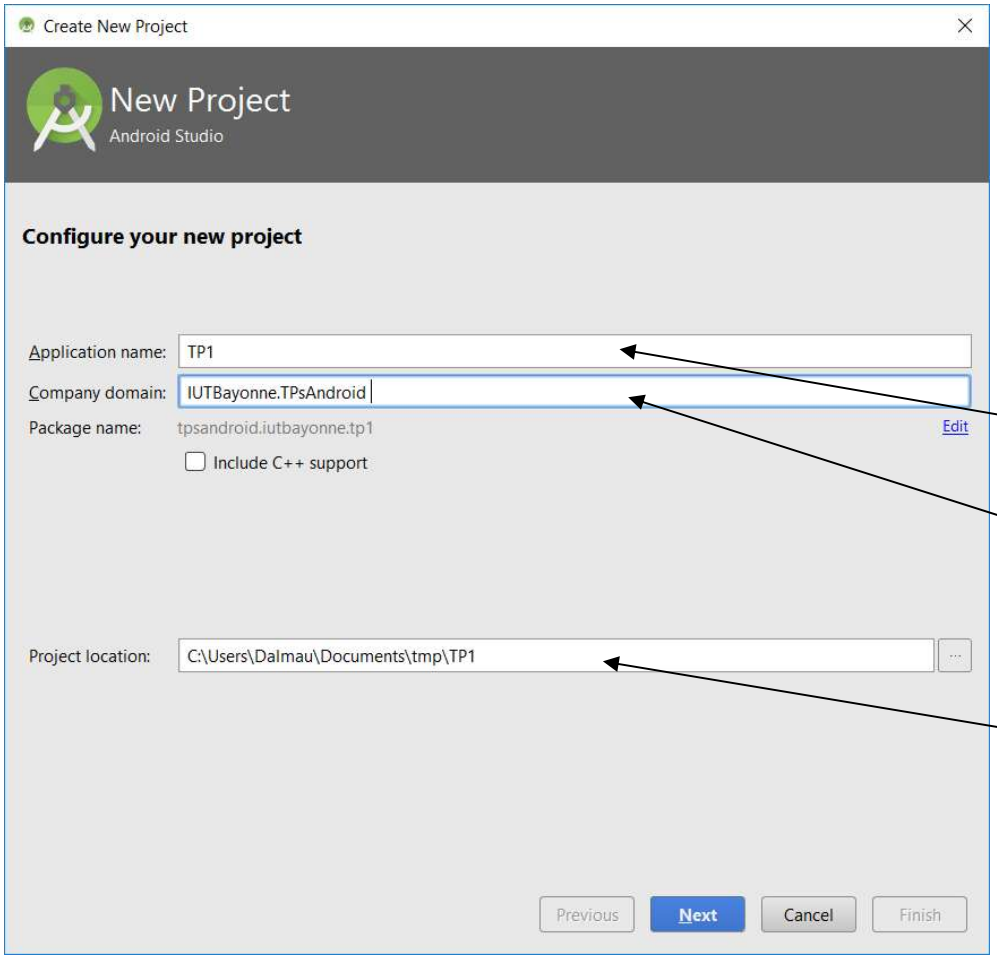
https://android-studio.fr.uptodown.com/

Développer avec AS

Créer un projet



Développer avec AS



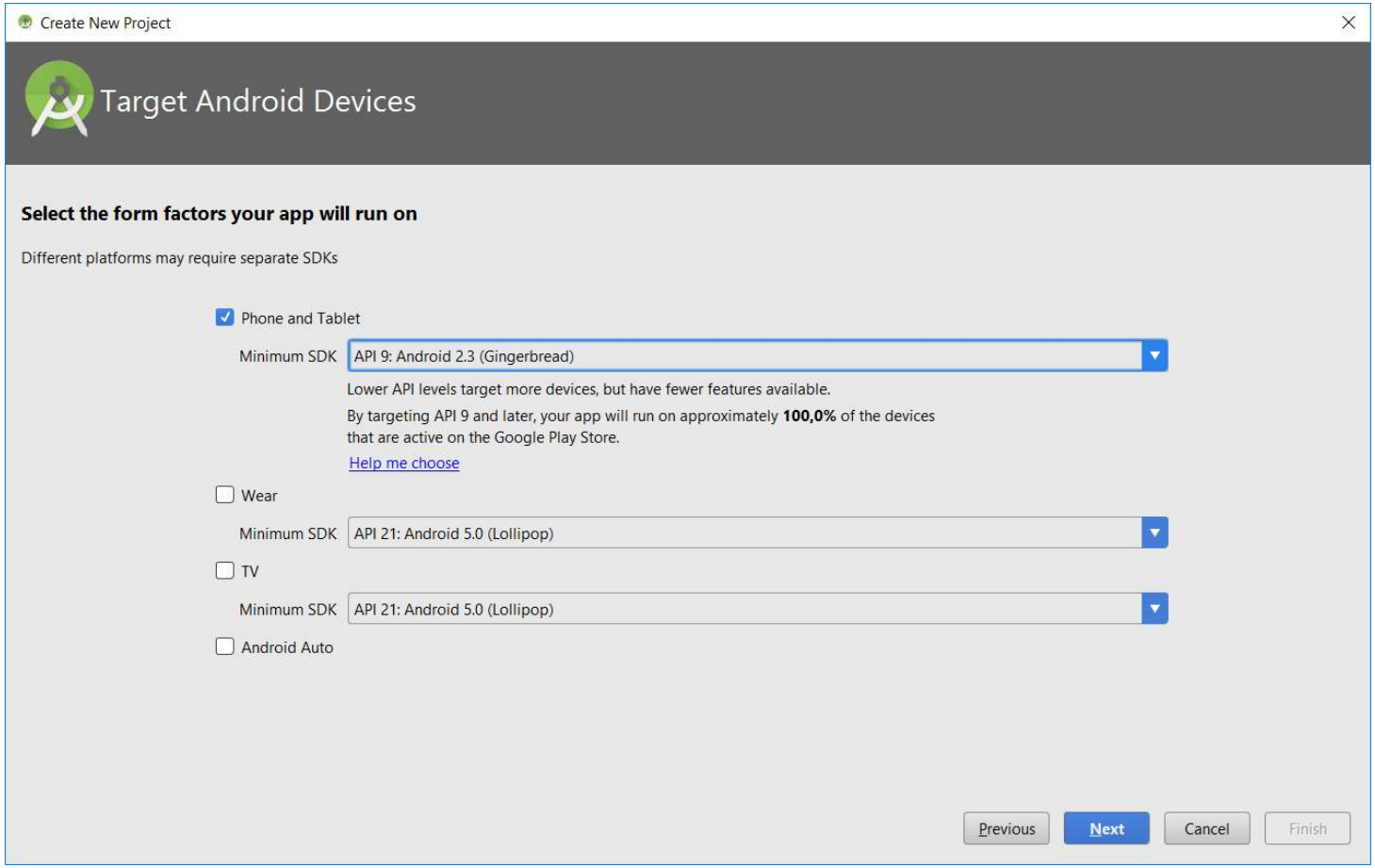
Nom de l'application

Nom unique (le plus souvent une URL)

Répertoire

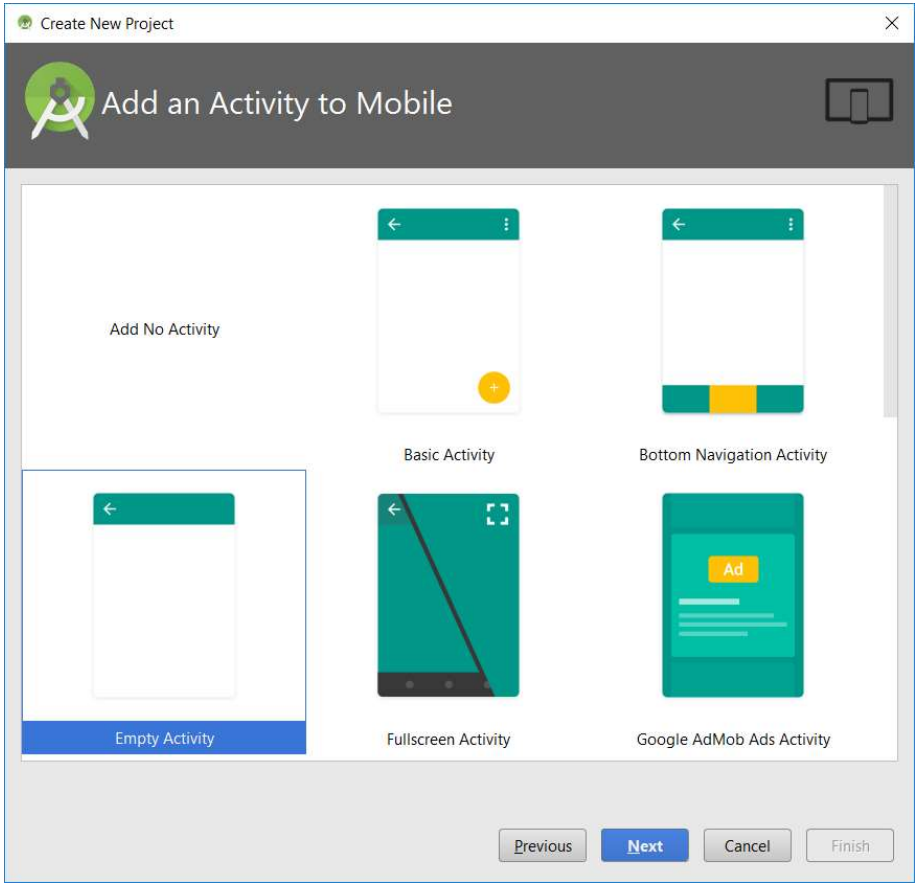
Développer avec AS

Choix du SDK (version d'Android)

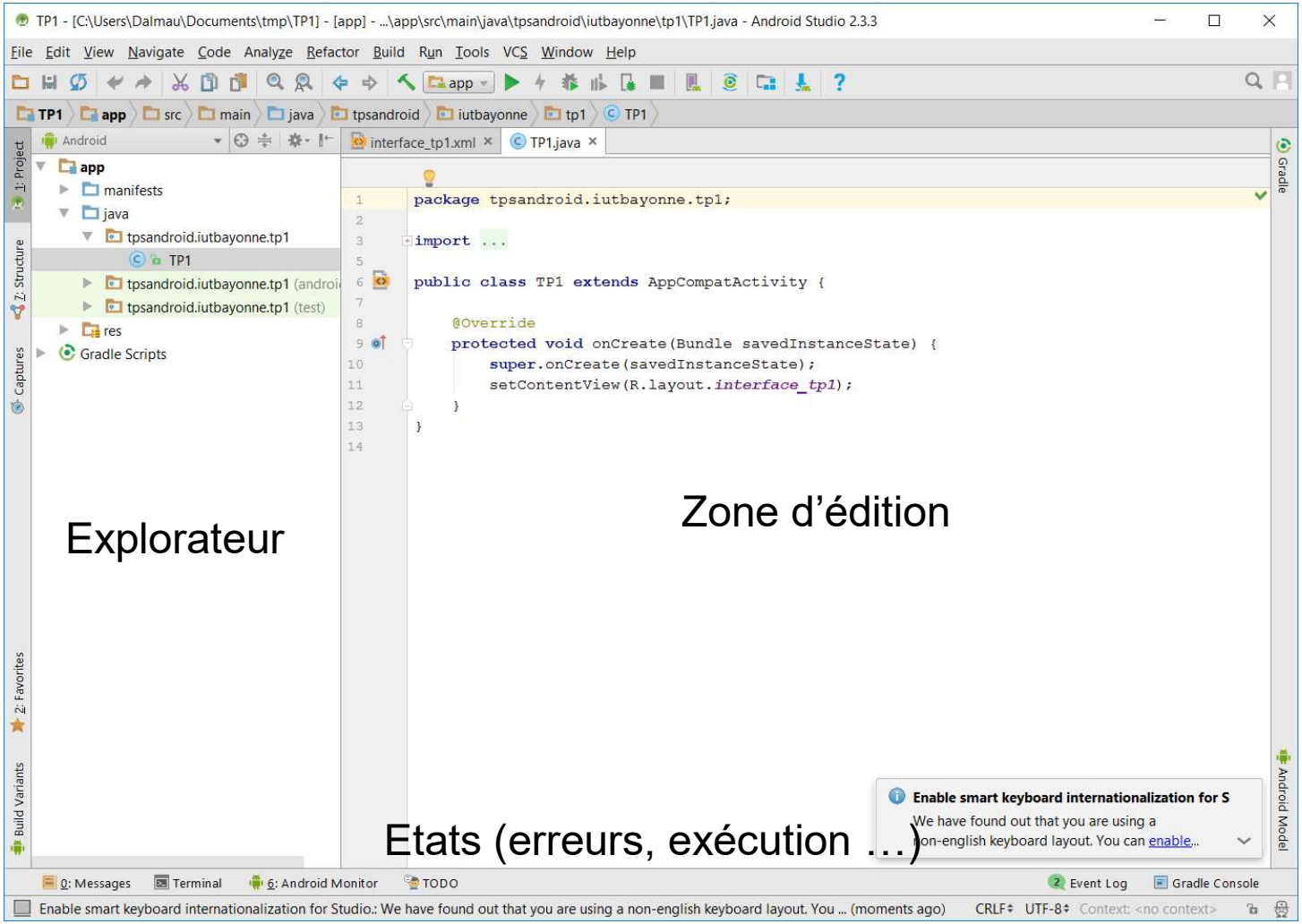


Développer avec AS

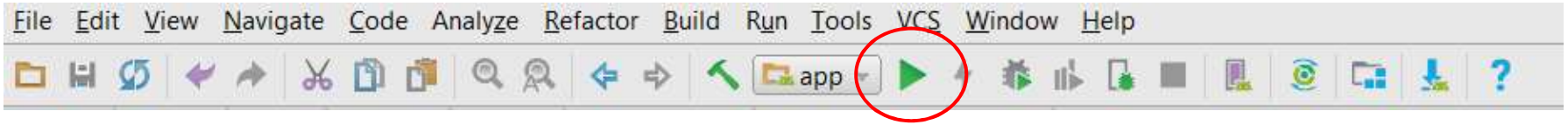
Choix du modèle d'application (thème)



Développer avec AS



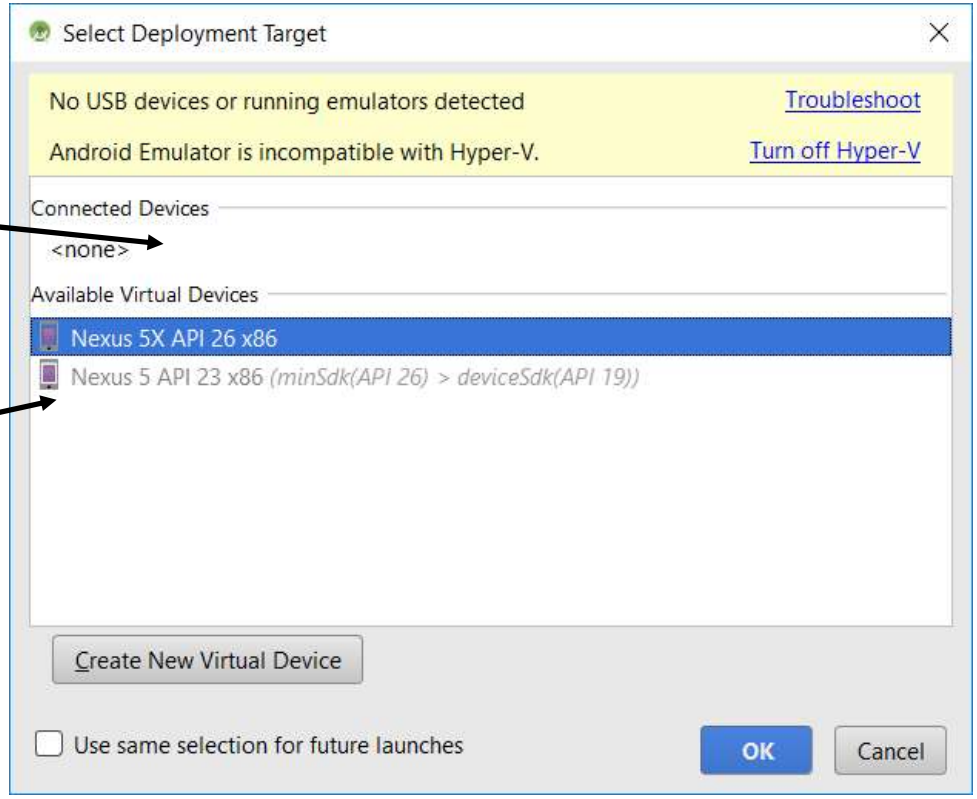
Tester une application



- Sur un terminal connecté par USB

OU

- Sur un terminal virtuel (simulateur)



Le fichier AndroidManifest

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

```
<?xml version="1.0" encoding="utf-8"?>
    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="tpsandroid.iutbayonne.tp1">

        <application
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:roundIcon="@mipmap/ic_launcher_round"
            android:supportsRtl="true"
            android:theme="@style/AppTheme">
            <activity android:name=".TP1">
                <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.

Rubriques du fichier AndroidManifest

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

Les ressources

Les ressources

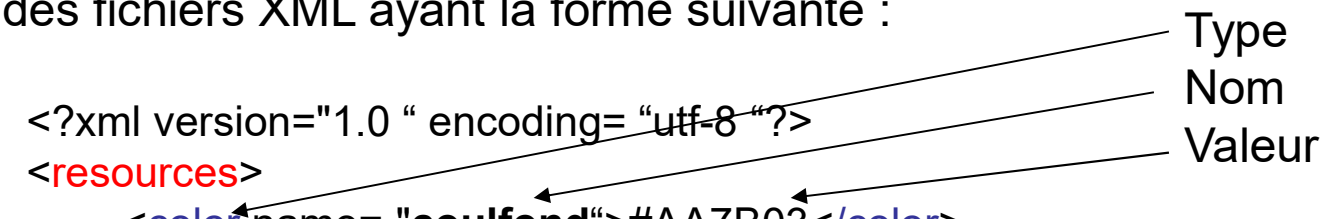
- Application embarquée ⇒ tout doit être dans le fichier .apk téléchargé

- 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>
```

Les noms (identificateurs) servent a 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 chaine contenue dans un fichier XML placé dans le répertoire **res/values** et définie comme suit :

```
<resources>
   …
   <string name="machaine">Contenu de cette chaine</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 chaine

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.ma_ressource);
    ```

  - Ressource distante

    ```
    Uri.parse("http://domaine.sous_doamine/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, …)

Eléments 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 />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
        </activity>
        <service>
            <intent-filter>
            . . .
            </intent-filter>
        </service>
        <receiver>
            <intent-filter>
            . . .
            </intent-filter>
        </receiver>
        <provider>
            <grant-uri-
              permission />
        </provider>
        <uses-library />
    </application>
</manifest>
```

Général

Pour chaque service

Pour chaque écouteur d'intentions diffusées

Pour chaque fournisseur de contenu

Pour chaque activité

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)

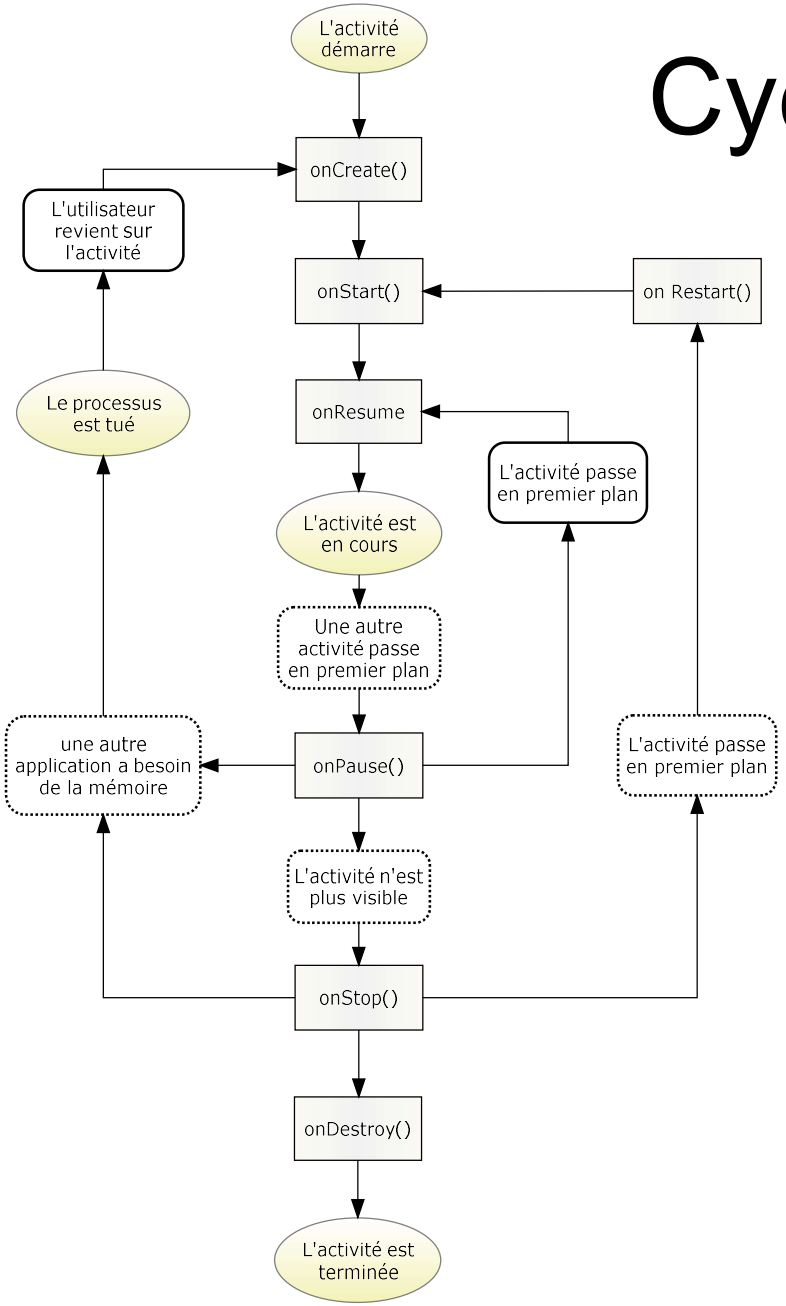
- On surcharge certaines méthodes qui sont appelées par Android pour définir le comportement (même principe que les applets) :
  - onCreate lors de la création
  - onDestroy lorsque l'activité se termine
  - onStart lorsque l'activité démarre ou redémarre
  - onPause lorsque l'activité n'est plus en 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 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

Pensez vos interface pour un smartphone

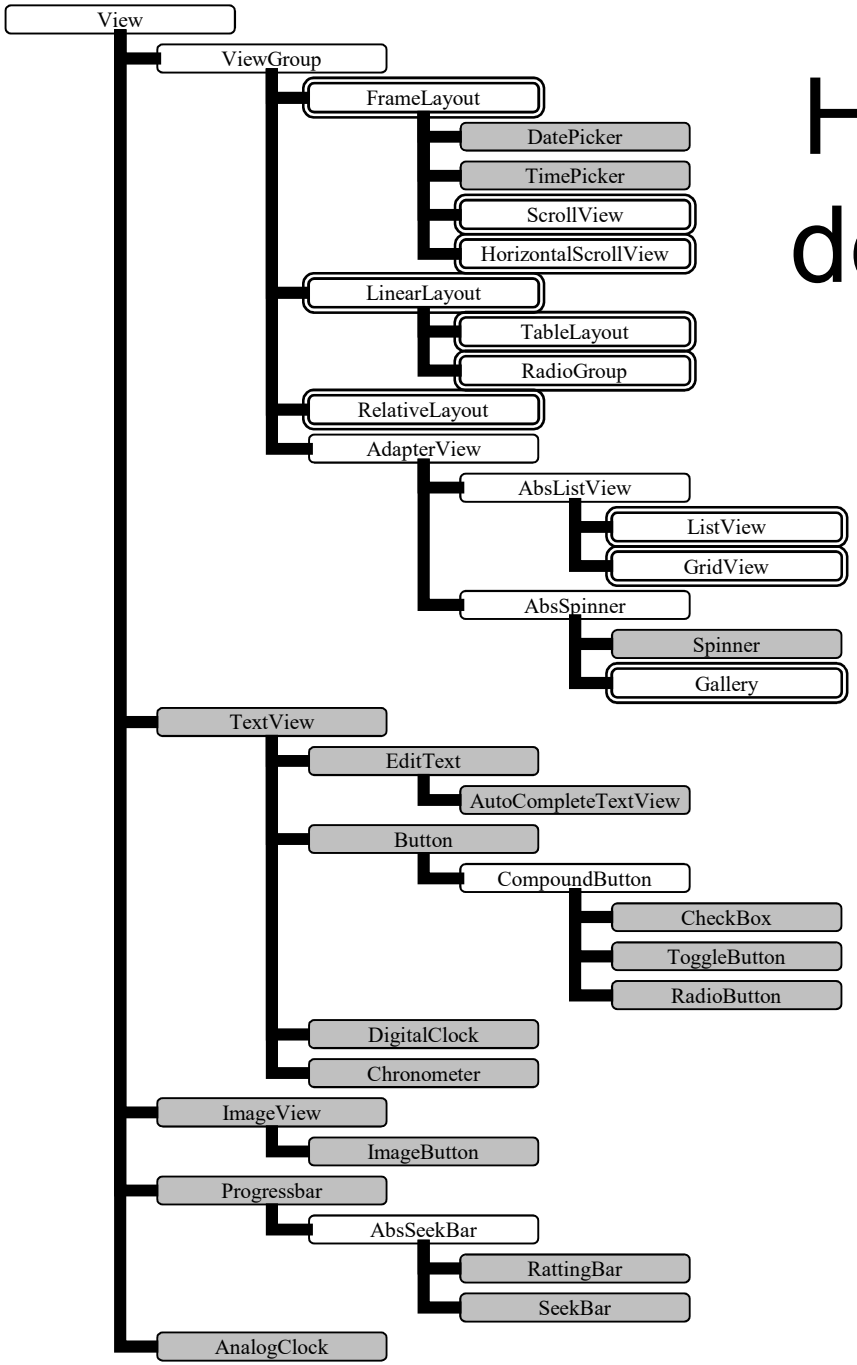
- Tous les smartphones n'ont pas la même définition d'écran => une partie de votre interface peut être totalement inaccessible sur un petit écran !
- Prévoir des ascenseurs quand c'est possible
- Découper l'interface en sous parties, passer d'une interface à une autre
- Eviter de donner des dimensions fixes
  - Utiliser le plus possible les tailles relatives « wrap_content » , « match_parent » et « fill_parent »
  - Préférer les dimensions relatives « dp » et « sp » aux dimensions fixes « px » et « pt »

Création d'interfaces

- Par programme (comparable à java swing) mais avec des classes propres à Android
  - Définition de layouts (un layout = un conteneur + un mode de placement ≡ JPanel + xxxLayout)
  - 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 (wigets)

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
>
   <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>
```

Pour chaque 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* celui de l'activité qui gère cette interface (obtenu par getApplicationContext())
  – *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 : "#AARRVVBB"

  – 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)
- 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 devenu MATCH_PARENT rempli 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="match_parent"
   android:layout_height="wrap_content"
   >
       <Spinner andoid:id="@+id/planetes"
             android:layout_width="match_parent"
             android:layout_height="wrap_content"
       />
       <ProgressBar andoid:id="@+id/attente"
             android:layout_width=" wrap_content "
             android:layout_height="wrap_content"
       />
</LinearLayout>
```

Exemple d'interface simple

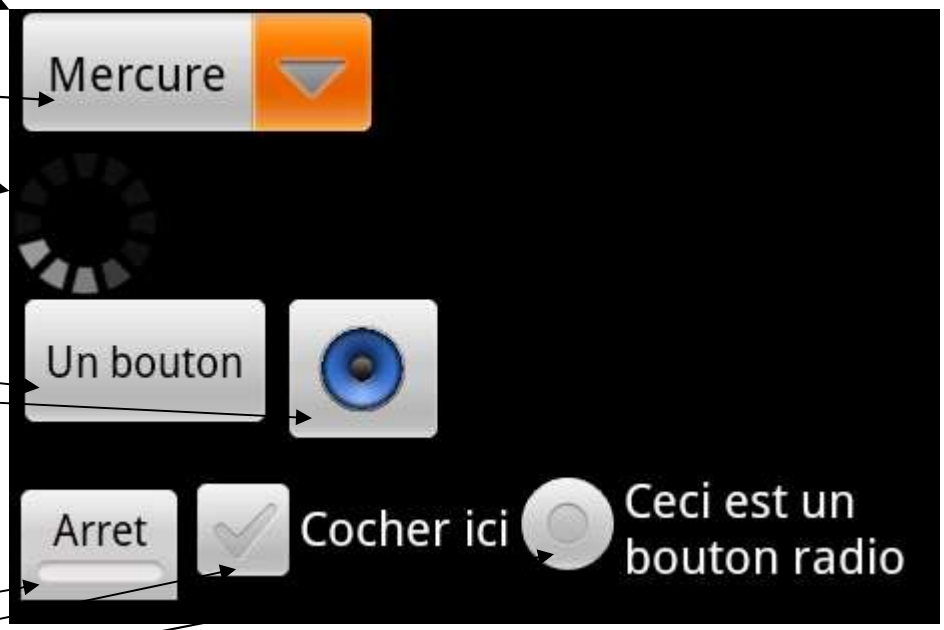
Un LinearLayout contenant 3 éléments placés l'un à coté de l'autre

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

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="match_parent"
   android:layout_height="match_parent"
   >
        <Spinner … />
        <ProgressBar … />
        <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        >
              <Button …/>
              <ImageButton …/>
        </LinearLayout>
        <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_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

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

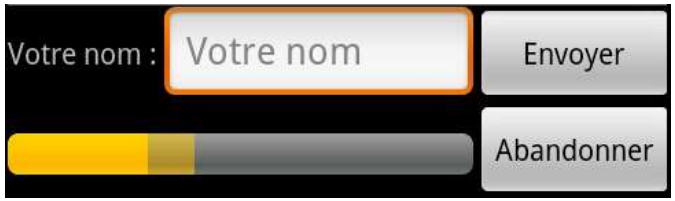
  android:shrinkColumns Pour définir les numéros de colonnes qui peuvent être rétrécies en fonction de la place disponible

  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="match_parent"
android:layout_width="match_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>
```

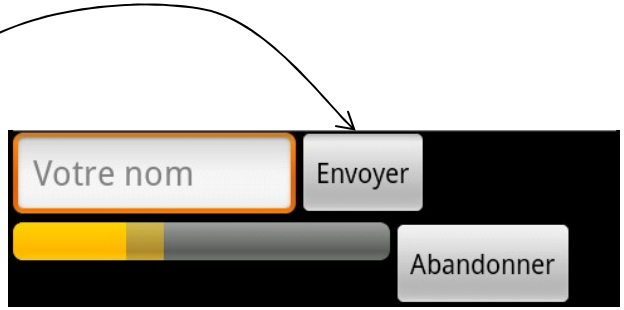


RelativeLayout

- 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="match_parent"
android:layout_width="match_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

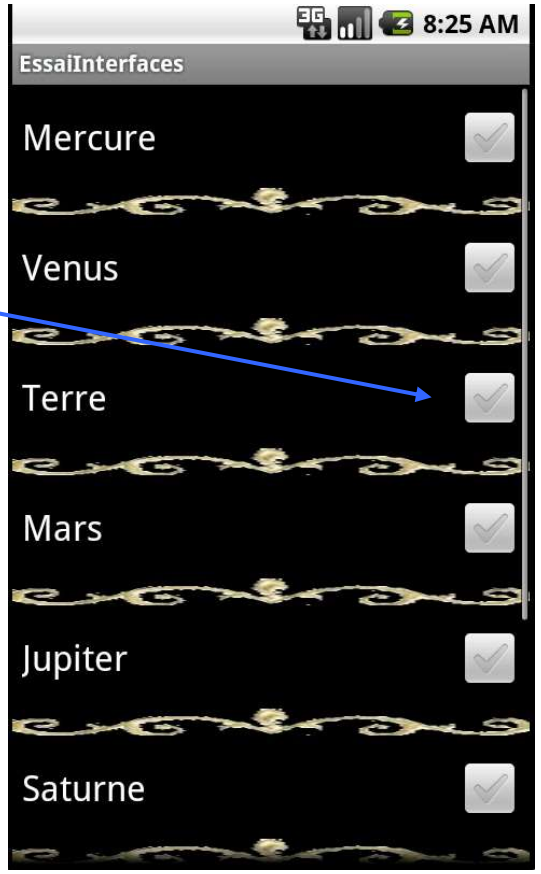
```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/liste_de_planetes«
   android:entries ="@array/planetes"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:divider="@drawable/separateur"
   android:dividerHeight="25px"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

  - Dans le XML de valeurs

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

- Dans le code

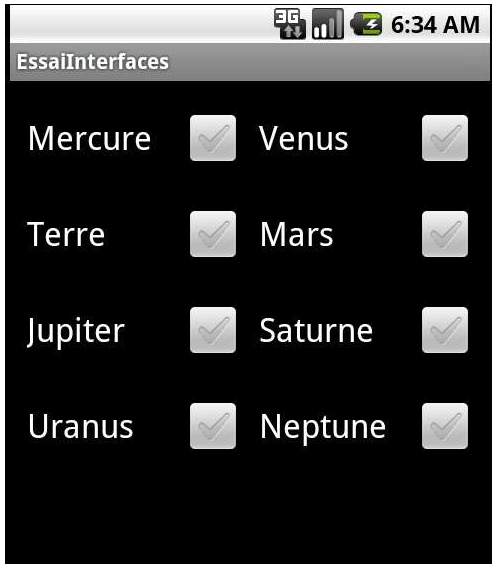
```
ListView liste = (ListView) findViewById(R.id.liste_de_planetes);
String[] elements = getResources().getStringArray(R.array.planetes);
ArrayAdapter<String> adaptateur = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_multiple_choice);
for (int i=0; i<elements.length; i++) 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="match_parent"
  android:layout_height="match_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="match_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 RelativeLayout leur position est définie par rapport à ce contenant et/ou aux autres composants (android:layout_above …)
  - Lorsqu'ils sont placés dans un autre contenant leur position est définie par ce contenant (android:layout_height et android:layout_width)

- Taille

  android:layout_height="t" (où t peut être une unité ou prendre les valeurs MATCH_PARENT ou WRAP_CONTENT)

  android:layout_width="t" (où t peut être une unité ou prendre les valeurs MATCH_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 coté 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 Exemples

```
<ImageView android:id="@+id/image"
  android:src="@drawable/keithwembley"
  android:layout_width="match_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="match_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" en général une référence @string/xxx
  android:hint="initial" définit le texte à afficher quand la zone est vide (idem)

- ***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:texteStyle="g" (où s peut être normal, bold, italic) ces styles peuvent être combinés par |
  android:typeface="s" (où s peut être normal, sans, serif, monospace)
  android:singleLine="b" (où b vaut true ou false) limite le texte à une seule ligne
  android:lines 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:numerics="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^{0s} 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 match_parent et wrap_content, définit la hauteur du menu déroulant

android:dropDownWidth="unité" on peut aussi utiliser les constantes match_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

WebView

WebView permet l'affichage de pages web :

- Récupération:

```
maVue = (WebView) findViewById(R.id.nom);
```

- Charger une page :

```
maVue. loadUrl("http://..... ");
maVue.reload();
maVue.stopLoading()
```

- Naviguer dans la page :

```
maVue.pageUp();
maVue.pageDown();
```

- Naviguer dans l'historique :

```
maVue.goBack();
maVue.goForward();
```

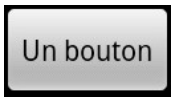
- Configurer pour l'utilisation du javascript :

```
WebSettings webSettings = maVue.getSettings();
WebSettings.setJavaScriptEnabled(true);
```

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:maxWidth 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 

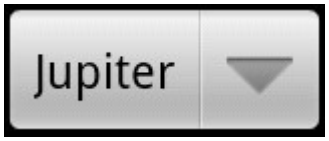
- RadioButton 

- 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

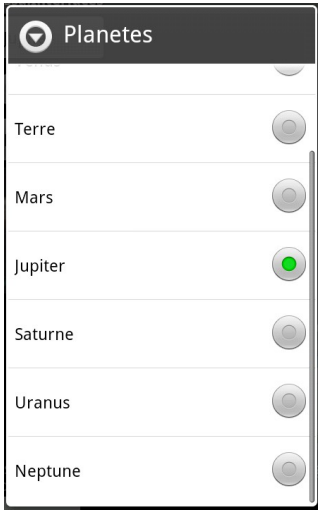


- Propriétés :

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

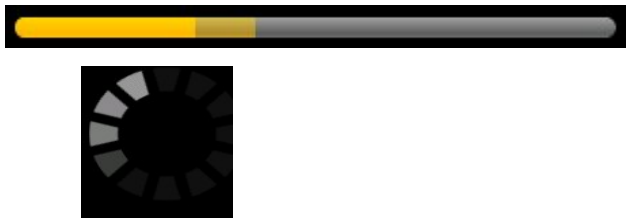
- TimePicker

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 ¼ d'étoiles par exemple)

Horloges et Chronomètres

- AnalogClock

- DigitalClock

- Chronometer

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

Evénements généraux

- View

Evénement	Association Classe	Méthode • Paramètres
Clic	setOnClickListener View.OnClickListener	onClick(View) • Élément concerné
Clic long	setOnLongClickListener View.OnLongClickListener	onLongClick(View) • Élément concerné
Clavier	setOnKeyListener View.OnKeyListener	onKey(View, int, KeyEvent) • Élément concerné • Code clavier • Evénement clavier
Touché	setOnTouchListener View.OnTouchListener	onTouch(View, MotionEvent) • Élément concerné • Evénement de touché

Evénements spécifiques

- ToggleButton RadioButton CheckBox

Evénement sur un élément	**Association** **Classe**	**Méthode** • **Paramètres**
Changement d'état	setOnCheckedChangeListener CompoundButton.OnCheckedChangeListener	onCheckedChange(CompoundButton, boolean) • Élément concerné • État actuel

Evénements spécifiques

- ListView , GridView , Spinner et Gallery

Evénement sur un élément	**Association** **Classe**	**Méthode** • **Paramètres**
Clic	setOnItemClickListener AdapterView.OnItemClikListener	onItemSelected (AdapterView, View, int, long) •Vue concernée •Elément cliqué •Rang de cet élément •ID de cet élément
Sélection	setOnItemSelectedListener AdapterView.OnItemSelectedListener	onItemSelected (AdapterView, View, int, long) onNothingSelected(AdapterView) •Idem

Evénements spécifiques

- TextView et EditText

Evénement	Association Classe	Méthode • Paramètres
Fin de saisie	setOnEditorActionListener TextWatcher	onEditorAction(TextView, int, KeyEvent) • Elément concerné • Code de l'action (EditorInfo.IME_ACTION_DONE) • Evénement clavier (peut être null)
Modification	addTextChangedListener TextChangedListener	beforeTextChanged(CharSequence, int, int, int) afterTextChanged(CharSequence, int, int, int) • Texte • Point de départ de la modification • Nombre de cars remplacés • Nombre de cars de remplacement
Saisie	setKeyListener KeyListener	onKeyDown(View, Editable, int, KeyEvent) onKeyUp(View, Editable, int, KeyEvent) • Elément concerné • Texte • Code de la touche • Evénement clavier

Evénements spécifiques

- DatePicker

Evénement de choix	Association Classe	Méthode • Paramètres
Choix	init DatePicker.OnDateChangedListener	onDateChanged(DatePicker, int, int, int) • Elément concerné • Année • Mois • Jour

- TimePicker

Evénement de choix	Association Classe	Méthode • Paramètres
Choix	setOnTimeChangedListener TimePicker.OnTimeChangedListener	onTimeChanged(TimePicker , int, int) • Elément concerné • Heure • Minutes

Evénements spécifiques

- SeekBar

Evénement	Association Classe	Méthode • Paramètres
Curseur déplacé	setOnSeekBarChangeListener SeekBar.OnSeekBarChangeListener	onProgressChanged(SeekBar, int, boolean) • Elément concerné • Position du curseur • Action de l'utilisateur
Début de déplacement		onStartTrackingTouch(SeekBar) •Elément concerné
Fin de déplacement		onStopTrackingTouch(SeekBar) •Elément concerné

Evénements spécifiques

- RatingBar

Evénement	Association Classe	Méthode • Paramètres
Valeur modifiée	setOnRatingBarChangeListener RatingBar.OnRatingBarChangeListener	onRatingChanged(RatingBar, float, boolean) • Elément concerné • Valeur choisie • Action de l'utilisateur

- Chronometer

Evénement	Association Classe	Méthode • Paramètres
Incrémentation	setOnChronometerTickListener Chronometer.OnChronometerTickListener	onChronometerTick(Chronometer) • Elément concerné

Exemple

Mettre en place l'interface et récupérer les widgets

Dans main.xml :

<TimePicker android:id="@+*id/temps*"
<Spinner android:id="@+id/type"
<*Button android:id="@+id/ok*"
...

```
public class MonActivite extends Activity {
        private TimePicker choixHeure;
        private Button ok;
        private Spinner type;

        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            choixHeure=(TimePicker)findViewById(R.id.temps);
            type=(Spinner)findViewById(R.id.type);
             ok=(Button)findViewById(R.id.ok);
        }
}
```

Exemple traiter les événements

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        choixHeure=(TimePicker)findViewById(R.id.temps);
        choixHeure.setOnTimeChangedListener(new ChangeHeure());
        type=(Spinner)findViewById(R.id.type);
        type.setOnItemSelectedListener(new ChoixType());
        ok=(Button)findViewById(R.id.ok);
        ok.setOnClickListener(new ActionOK());
}
private class ChangeHeure implements OnTimeChangedListener {
        public void onTimeChanged(TimePicker choix, int heure, int minute) {
                // traitement
        }
}
private class ActionOK implements OnClickListener {
        public void onClick(View bouton) {
                finish(); // terminer l'activité
        }
}
 private class ChoixType implements OnItemSelectedListener {
        public void onItemSelected(AdapterView<?> arg0, View arg1, int rang, long arg3) {
                // traitement
        }
        public void onNothingSelected(AdapterView<?> arg0) {
                // traitement
        }
   }
```

```
// propriétés de l'activité
private TimePicker choixHeure;
private Button ok;
private Spinner type;
…
```

Notifications

La classe Toast

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

- ***Création d'un Toast***

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

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 peut être construite à partir d'un Bitmap

La classe BitmapFactory

Permet de créer des images (classe Bitmap) depuis diverses sources

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

```
decodeResource(getResources(), R.drawable.xxx ); // xxx = image dans res/…
```

- 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
- inPugeable pour libérer la mémoire occupée par l'image
- outHeigth, outWidth pour définir la taille

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'aactivité)
  - 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)

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

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

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 contrtôleur de média
  - setVideoPath(String) définit le média par un chemin de fichier
  - setVideoURI(Uri) définit le média par une Uri

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

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

- **Evénements**
  - setOnCompletionListener(MediaPlayer.OnCompletionListener) associe un écouteur d'événements
    - 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 :

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

- Fermeture du synthétiseur dans la méthode onDestroy de l'activité :

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

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

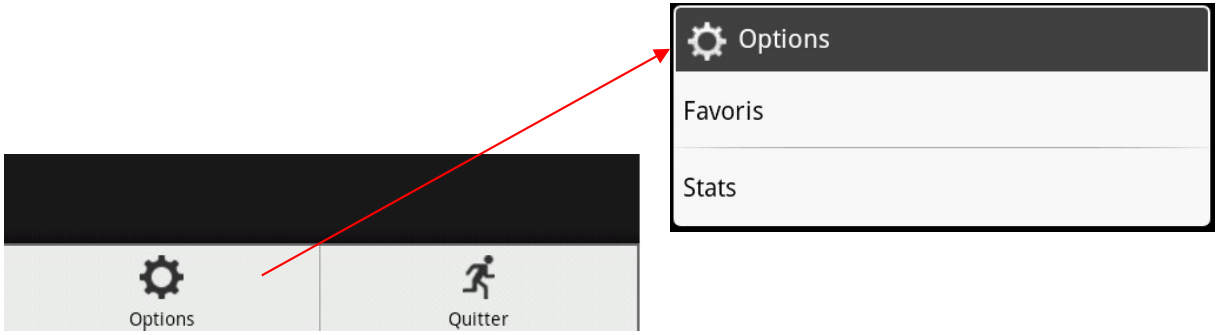
- Chaque élément d'un menu peut proposer des sous menus
- Décrits dans le fichier XML sous la forme :

```
<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 onCreateOptionMenu 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 :

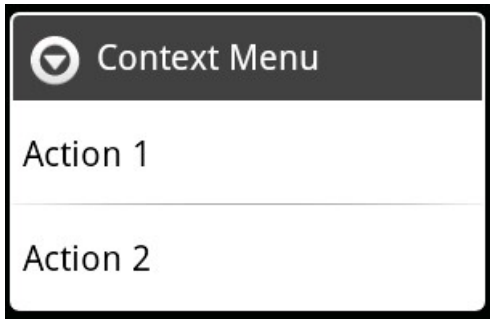
```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.nom_du_choix_1:
            // traitement du choix 1
            return true;
        …
        case R.id.nom_du_sous_choix_1:
            // traitement du sous choix 1
            return true;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

  registerForContexMenu(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);
    }
}
```

Navigation entre activités

Navigation entre activités

Démarrer une activité

- Mode explicite :
  - On indique la classe de l'activité à lancer (cette classe fait partie de l'application et a été chargée avec elle)
  - 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

Navigation entre activités

Utilisations courantes

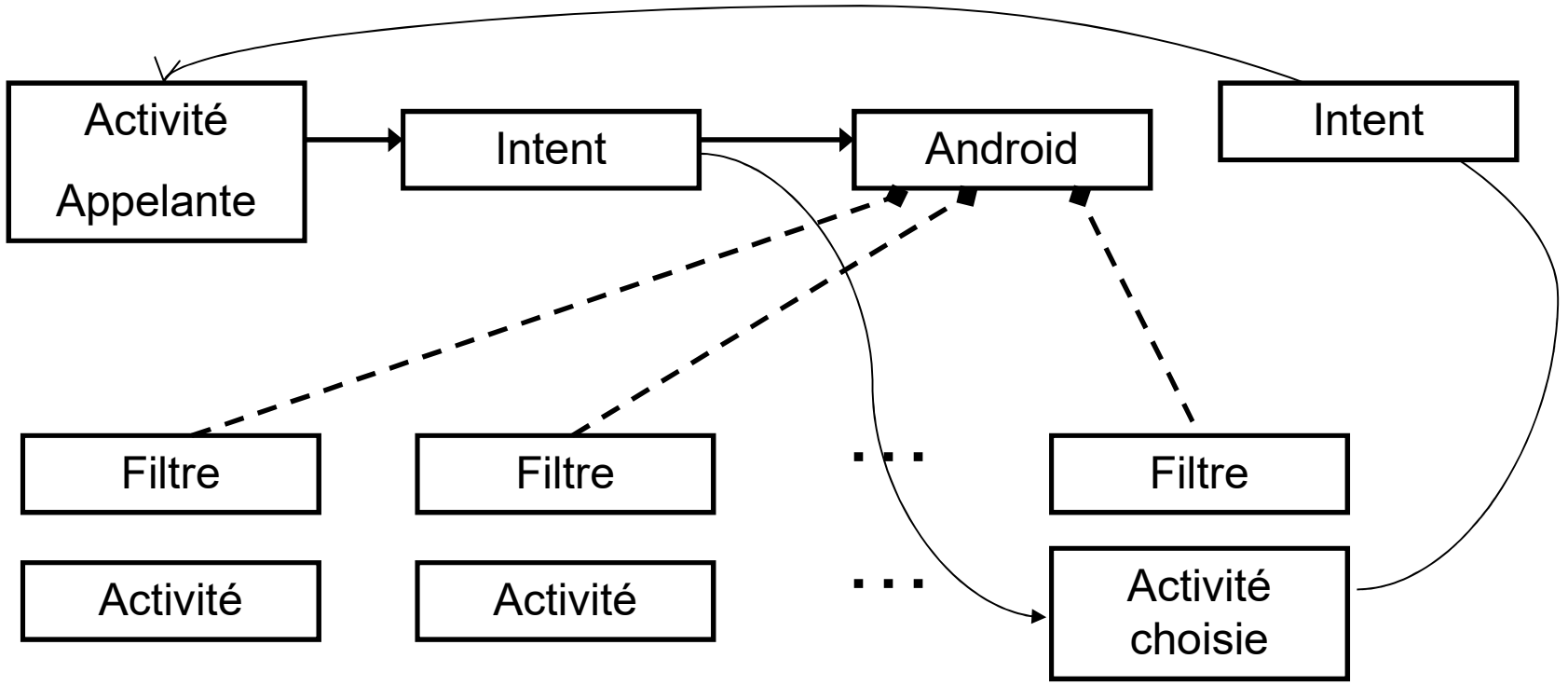
- Mode explicite :
  - Multi fenêtre : une activité = une interface

  Naviguer entre plusieurs interfaces = naviguer entre plusieurs activités

  ATTENTION : les activités sont empilées ⇒ retour en arrière

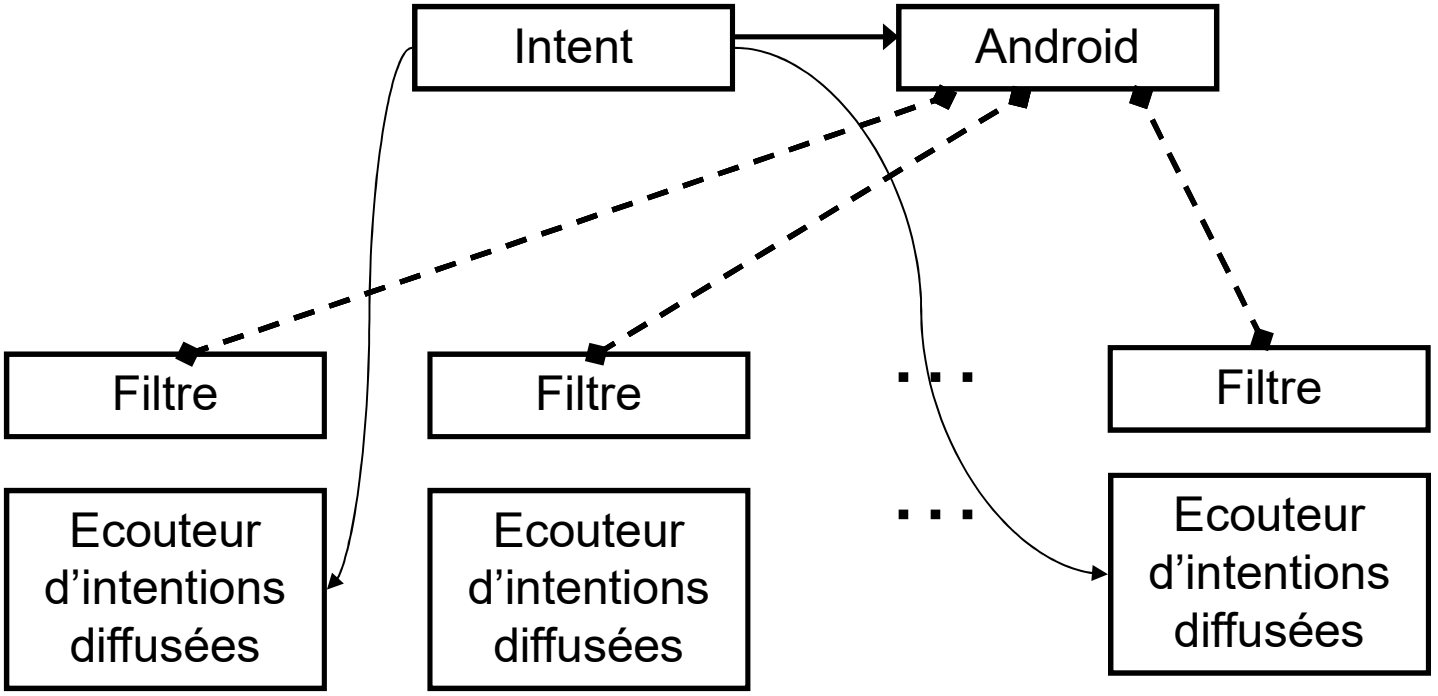
- Mode implicite :
  - Changer d'activité
  - Lancer une activité comme un service
  - Définir des écouteurs d'intensions diffusées pour réagir a des événements d'Android

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 catégorie (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 paramètres 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://hote: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 à 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 d'utilisation

- 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 1ère zone quand on clique le bouton la traduction s'affiche dans la 2ème zone
- On prévoit que l'activité puisse démarrer avec un texte à traduire qu'elle reçoit en paramètre (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);
```

Dans le manifest on doit mettre :

```
<activity android:name="NomDeLaClasseDeLActiviteALancer"/>
```

- 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);
```

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 demarre = new Intent(this, NomDeLaClasseDeLActiviteB.class);
// ajouter les paramètres passés à B dans l'Intent demarre
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 1er 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 un 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)
  - getCharSequence(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)
  - getStringExtra(String)
  - getBooleanArrayExtra(String)
  - …

Exemple : lecture d'un QrCode (appel explicite)

```
// Lancement de l'application de scan de QrCode « zxing » dans notre appli
    Intent action = new Intent("com.google.zxing.client.android.SCAN"); // voir doc zxing
    action.putExtra("SCAN_MODE", "QR_CODE_MODE"); // voir doc zxing
    startActivityForResult(action, SCAN_NUM); // SCAN_NUM est une constante

// Récupération de la chaîne résultat du scan dans notre appli
    protected void onActivityResult(int ident, int code_retour, Intent retour) { // surcharge
        switch(ident) {
            case SCAN_NUM :
                String lu;
                if (code_retour == RESULT_OK)
                        lu = retour.getStringExtra("SCAN_RESULT"); // voir doc zxing
                 if (code_retour == RESULT_CANCELED)
                        lu = new String("Scan non fait"); // pas de scan
                …
                break;
            ….
        }
    }
```

Écouteurs d'intensions diffusées

- Dans le manifest

```
<receiver name= "nom_de_la_classe" >
    <intent-filter>
        <action>
            action réalisée par cet écouteur par exemple :
        android:name="android.bluetooth.BluetoothDevice.ACTION_ACL_CONNECTED"

        </action>
    </intent-filter>
</receiver>
```

- Ecrire une classe

```
public class nom_de_la_classe extends BroadcastReceiver {
    public void onReceive(Context contexte, Intent parametre) {
        …
    }
}
```

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>
```

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 précissant 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 "pdus"

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. Acces 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" />
```

Utilisation de la localisation

- Récupérer le service
  position = (LocationManager).getSystemService(LOCATION_SERVICE);

- Savoir si ondispose du GPS ou du réseau
  boolean parGPS = position.isProviderEnabled(LocationManager.GPS_PROVIDER);
  boolean parReseau = position.isProviderEnabled(LocationManager.NETWORK_PROVIDER);

- Initialiser le service
  - Pour le GPS :
    position.requestLocationUpdates(LocationManager.GPS_PROVIDER,
    temps_min, distance_min, ecouteur);

  - Pour le réseau :
    position.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
    temps_min, distance_min, ecouteur);

Ecouteur de localisation

```
Class Ecouteur implements LocationListener {

    public void onProviderDisabled(String fournisseur) { // perte de localisation
    }

    public void onProviderEnabled(String fournisseur) { // localisation accessible
    }

    public void onStatusChanged(String fournisseur, int etat, Bundle extras) {
        // changement d'état du service de localisation
        etat peut être OUT_OF_SERVICE, TEMPORARILY_UNAVAILABLE ou
        AVAILABLE.
    }

    public void onLocationChanged(Location position) {
        // On récupère la nouvelle position sous forme d'un objet de classe
        Location
    }
}
```

La classe Location

- Récupérer les coordonnées
  - getLatitude() en degrés (réel)
  - getLongitude() en degrés (réel)
  - getAltitude() en mètres (entier)
  - getAccuracy() en mètres (réel)
- Récupérer la vitesse
  - getSpeed() en mètres/seconde (réel)
- Calculer une distance
  - distanceTo(Location autre) en mètres (réel)

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" />
```

Appareil photo (préparatifs)

- Récupérer le SurfaceView placé dans l'interface (XML)

```
SurfaceView ecran = (SurfaceView)findViewById(R.id.xxx);
SurfaceHolder surface = ecran.getHolder();
```

- Lui associer un écouteur d'événements

```
surface.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
surface.addCallback(new Previsualisation());
```

Appareil photo (écouteur d'événements)

```
private class Previsualisation implements SurfaceHolder.Callback {

        public void surfaceCreated(SurfaceHolder surf) { // Quand la vue est créée
            appareilPhoto = Camera.open(); // récupérer l'appareil photo
            try {
                appareilPhoto.setPreviewDisplay(surf); // lui associer la vue
            }
            catch (IOException ioe) {
                // erreur lors de l'association de la prévisualisation
                appareilPhoto.stopPreview(); // supprimer la prévisualisation
                appareilPhoto.release(); // libérer l'appareil photo
            }
        }

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

        public void surfaceDestroyed(SurfaceHolder holder) { // Quand la vue est détruite
            if (appareilPhoto != null) {
                appareilPhoto.stopPreview(); // supprimer la prévisualisation
                appareilPhoto.release(); // libérer l'appareil photo
            }
        }
}
```

Prise de photo

- Prise de photo :

  appareilPhoto.startPreview();

  appareilPhoto.takePicture(EcouteurPrise, Ecouteur_raw, Ecouteur_jpeg);

  Remarque : on peut mettre null pour les écouteurs que l'on n'a pas besoin (par exemple les 2 premiers)

- Ecouteur pour JPEG

```
Class EcouteurJPEG implements PictureCallback {
        public void onPictureTaken(Camera c, byte[] img) {
            // Pour récupérer une image (Bitmapdrawable)
                BitmapFactory.Options options = new BitmapFactory.Options();
                options.inPurgeable = true; // pour libérer la mémoire prise par l'image
                options.inSampleSize = x; // pour réduire l'image dans un rapport 1/x
                BitmapDrawable image = new BitmapDrawable( BitmapFactory.decodeByteArray (img,
                    0, img.length, options));
        }
}
```

Autre méthode

- Par appel de l'application appareil photo

```
Intent fairePhoto = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
File directory = new File(Environment.getExternalStoragePublicDirectory
    (Environment.DIRECTORY_PICTURES), getPackageName());
File fich = new File(directory.getPath() + "/maphoto.jpg");
fichUri = Uri.fromFile(fich);
fairePhoto.putExtra(MediaStore.EXTRA_OUTPUT, fichUri);
startActivityForResult(fairePhoto, CAPTURE_IMAGE);
```

- Puis récupération du fichier jpeg

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
    case CAPTURE_IMAGE) :
      if (resultCode == RESULT_OK) {
                    photoUri = data.getData();
      }
      else {
                    …
      }
      break;
      …
    }
}
```

Il faut les permissions :

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

Utilisation du Micro

- Démarrer la capture

```
enregistreur = new MediaRecorder();
enregistreur.setAudioSource(MediaRecorder.AudioSource.MIC);
enregistreur.setOutputFormat(MediaRecorder.OutputFormat. AMR_NB);
                        ou MPEG_4 ou AAC_ADTS ou AMR_WB ou RAW_AMR ou THREE_GPP …
enregistreur.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
                        ou AAC ou AAC_ELD ou AMR_WB ou VORBIS ou HE_AAC …
try {
        enregistreur.prepare();
        enregistreur.start();
    }
    catch (IOException e) {
        // erreur
    }
```

Pour enregistrer le son dans un fichier :

```
enregistreur.setOutputFile(String nom);
```

Pour mesurer le niveau sonore :

```
enregistreur. getMaxAmplitude();
```

- Arrêter la capture

```
enregistreur.stop();
enregistreur.release();
```

Capture vidéo

- Par appel de l'application caméra

```
File mediaFile = new File(Environment.getExternalStorageDirectory().
                                                    getAbsolutePath()+"/mavideo.mp4");
Intent demarre = new Intent (MediaStore.ACTION_VIDEO_CAPTURE);
Uri videoUri = Uri.fromFile(mediaFile);
demarre.putExtra(MediaStore.EXTRA_OUTPUT, videoUri);
startActivityForResult(demarre, VIDEO_CAPTURE);
```

- Puis récupération du fichier video

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == VIDEO_CAPTURE) {
            if (resultCode == RESULT_OK) {
                    videoUri = data.getData();
            }
            else{
                    …
            }
    }
}
```

Vibreur (classe Vibrator)

- Peut être utilisé pour alerter l'utilisateur

- Accès au vibreur par :
  Vibrator vib = (Vibrator) getSystemService(Context.Vibrator_SERVICE)

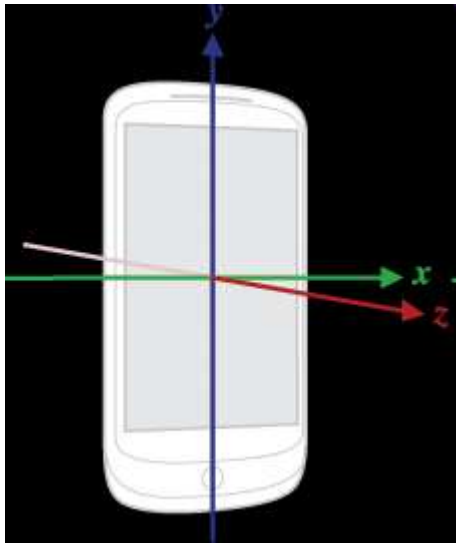
- Faire vibrer par :
  La méthode vib.vibrate(int) de la classe Vibrator
  En donnant la durée en ms
- Ou par :
  La méthode vib.vibrate(long[], int) de la classe Vibrator
    Le 1er paramètre est un tableau de durées en ms arrêt/vibre/arrêt/vibre/…
    Le 2ème est -1 si pas de répétition sinon indique l'index à partir duquel on recommence

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)
  - Champ magnétique (champ magnétique ambiant sur 3 axes)
  - *Orientation (angles du périph par rapport aux 3 axes) pas un vrai capteur : calculé*
  - Lumière (luminosité ambiante)
  - 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_SER
    VICE);
```

- Récupération d'un capteur par :

```
Sensor monCapteur =

                                        gestionnaireCapteurs.getDefaultSensor(ty
    pe_de_capteur);
```

Avec type_de_capteur :

– Sensor.TYPE_ACCELEROMETER	3 valeurs en m/s²
– Sensor.TYPE_GRAVITY	3 valeurs en m/s²
– Sensor.TYPE_GYROSCOPE	3 valeurs en radiants/s
– Sensor.TYPE_LIGHT	1 valeur en lux
– Sensor.TYPE_MAGNETIC_FIELD	3 valeurs en µtesla
– Sensor.TYPE_ORIENTATION	3 valeurs en °
– Sensor.TYPE_PRESSURE	1 valeur en psi (1 psi ≡ 6894,76 Pa)
– Sensor.TYPE_PROXIMITY	1 valeur en cm
– Sensor.TYPE_AMBIENT_TEMPERATURE	1 valeur en °Celsius

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

Attention : les 3 valeurs ne correspondent pas toujours aux mêmes axes
(en général : X,Y,Z sauf pour orientation : Z,X,Y)

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

Exemple (boussole)

```
private SensorManager gestCapt
private Sensor boussole
private EcouteBoussole ecouteur;
```

- Dans onCreate

```
gestCapt = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
boussole = gestCapt.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
ecouteur = new EcouteBoussole();
```

- Dans onResume

```
gestCapt.registerListener(ecouteur, boussole, SensorManager.SENSOR_DELAY_NORMAL);
super.onResume();
```

- Dans onPause

```
gestCapt.unregisterListener(ecouteur, boussole);
super.onResume();
```

Exemple (boussole suite)

```
private class EcouteBoussole implements SensorEventListener {

        public void onAccuracyChanged(Sensor capt, int prec) {
                // souvent rien ici
        }

        public void onSensorChanged(SensorEvent evt) {
                float[] mesures = evt.values.clone();
                // traitement des valeurs
                // mesures [0] = axe Z
                // mesures [1] = axe X
                // mesures [2] = axe Y

        }
}
```

Utilisation des capteurs

- Interfaces multimodales
  - En secours
  - En complément
  - En combinaison

- Prise en compte du contexte
  - Bruit
  - Lumière
  - Localisation
  - Mouvement

Ressources sur Internet

Ressources MM 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);
```

Audio (rappel)

- 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();
```

Ressources Audio sur Internet

On utilise toujours un MediaPlayer mais créé différemment et préparé :

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

Vidéo (rappel)

- 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 :

Ressources Vidéo sur Internet

La seule chose qui change c'est l'Uri qui désigne le média associé par setVideoURI()

- Définir le chemin de la vidéo (sur internet)

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

- Associer un lecteur vidéo à la vue (idem) :

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

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

Renvoie des résultats de classe Address qui contiennent :

- 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 Montaury, 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

Parcours d'une liste

Les listes (classe List de java) se parcourent grâce à un objet de classe Iterator

- Méthodes de la classe Iterator
  - hasNext() renvoie true s'il reste des éléments non parcourus
  - next() renvoi l'élément suivant (Object)

- Un parcours de liste peut se faire par :

```
Iterator curseur = maListe.iterator();
while (curseur.hasNext()) {
     ClasseElementListe ele = (ClasseElementListe)maliste.next();
    // traiter l'élément ele
}
```

- Depuis java 1.5 on peut aussi faire :

```
for(ClasseElementListe ele : maListe) {
    // traiter l'élément ele
}
```

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 affiché dans un widget MapView :

```
<com.google.android.maps.MapView android:id="@+id/carte"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:apiKey="mettre ici la clé obtenue" />
```

- MapView permet de définir des contrôle d'apparence de la carte :
  - setBuiltInZoomControls(boolean) autorise (true) ou invalide (false) le contrôle de zoom
  - displayZoomControls(boolean) affiche le contrôle de zoom. Le paramètre permet de donner le focus à ce contrôle (true)
  - 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 millionièmes 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 dur la carte affiché (x et y)

ATTENTION : GeoPoint utilise des coordonnées géographiques entières en millionièmes de degré alors que le GPS travaille en coordonnées réelles en degrés. Les méthodes sont :

  - getLatitudeE6()
  - getLongitudeE6()

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 formatable :

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

- Association d'un format à une zone du texte :

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

- Les 2ème et 3ème 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 echelle)
  - ScaleXSpan (float echelle)
  - UnderlineSpan ()
  - TypeSpaceSpan (" {normal, sans, serif, monospace } ")
  - StyleSpan (android.graphics.Typeface.{BOLD, BOLD_ITALIC, ITALIC, NORMAL})

Interfaces avancées
Fragments

Fragments (utilisation)

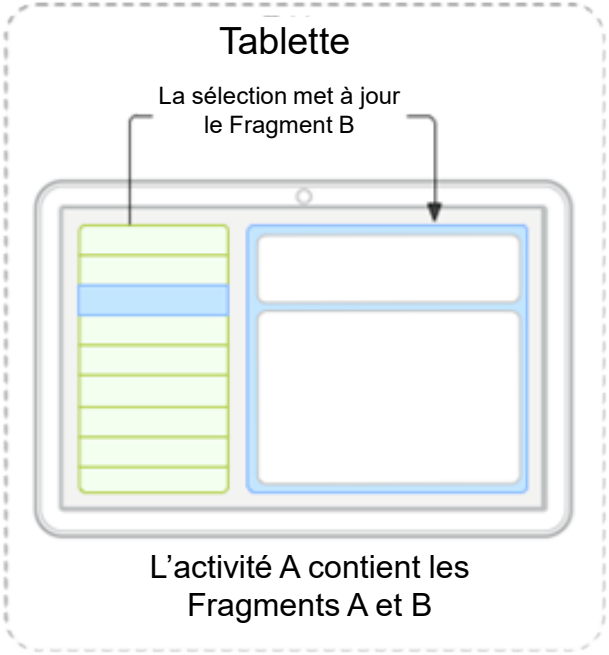
- Objectif : offrir un mécanisme de type activité secondaire permettant de réaliser une interface similaire pour les petits écrans (téléphones) et les grands écrans (tablettes).

- L'interface est divisée en parties (fragments) qui peuvent apparaître ou pas et être remplacés par d'autres

Selon la taille de l'écran :

- Le fragment apparaît (en remplace un autre)

- Le fragment est affiché par une sous activité



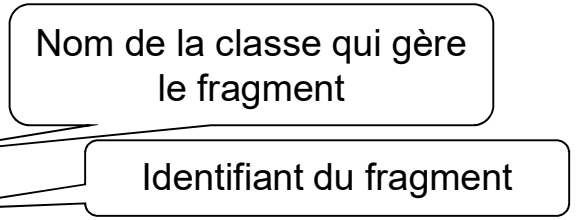
Fragments (comment faire ?)

- L'activité hérite de FragmentActivity au lieu de Activity
- L'interface est découpée en zones (Fragments) dans le fichier XML de description de l'interface de l'activité
- Un fichier XML est créé pour le contenu de chaque fragment
- Création d'une classe qui hérite de Fragment pour gérer un fragment et surcharge de certaines méthodes de Fragment (qui ressemble à la classe Activity)
- Modification des fragments par transactions :

```
FragmentTransaction transaction = getFragmentManager().beginTransaction();
    • transaction.add(R.id.nomDuContenant, fragment);
    • transaction.remove(fragment);
    • transaction.replace(R.id.nomDuContenant, fragment);
    • transaction.hide(fragment);
    • transaction.attach(fragment);
    • transaction.detach(fragment);
  transaction.commit()
```

Fragments (Layouts)

- Dans le fichier XML de l'interface




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

- Contenu dans le fichier XML du fragment (classique)

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

          <TextView android:id = "@+id/titre"
          android:layout_width = "wrap_content" android:layout_height = "wrap_content"
          …. />
          …
    </LinearLayout>
```

Fragments (code)

- Création du fragment :

```
class MonFragment extends Fragment {

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

         public void onViewCreated(View vue, Bundle sauve) {
               super.onViewCreated(vue, sauve);
                // Récupération des widgets (comme dans onCreate)
               titre = (TextView) vue.findViewById(R.id.titre);
         }

        public void onPause() {
             // sauvegarder l'état
        }
}
```

Fragments (code)

- Autres méthodes de la classe Fragment (à éventuellement surcharger) :
  - **onActivityCreated** appelée lorsque l'activité a été créée
  - **onAttach** appelée lorsque le fragment est attaché à une activité
  - **onDetach** appelée avant que le fragment ne soit détaché de son activité
  - **onDestroyView** appelée lorsque le fragment disparaît
  - **onViewStateRestored** appelée lorsque l'état du fragment a été restauré
  - **onCreate/onResume/onPause/onStart/onStop/onDestroy** semblable au cycle de vie d'une activité
  - **onActivityResult** comme pour une activité
- Accès à l'activité :
  - **getActivity** retourne l'activité qui contient ce fragment

Fragments (multi-écrans)

- Selon la taille de l'écran
  - On utilisera le fragment directement. On l'obtient par :

```
getSupportFragmentManager().findFragmentById(R.id. nom_du_fragment);
```

OU

  - On le fera apparaître (éventuellement par remplacement d'un autre) :

```
Fragment nouveauFragment = new AutreFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();

transaction.add(R.id.nomDuContenant, nouveauFragment); // ajout
OU
transaction.replace(R.id.nomDuContenant, nouveauFragment); // remplacement

transaction.commit();
```

OU

  - On lancera une sous activité qui affichera le fragment (startActivity ou startActivityForResult)

Fragments et Google Maps

- Une carte peut s'afficher dans un fragment qui la gère.
- L'activité hérite de FragmentActivity
- Dans onCreate l'activité récupère le fragment et le connecte à la carte en associant un écouteur :

```
MapFragment monFragment = (MapFragment) getFragmentManager()

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

- L'écouteur hérite de OnMapReadyCallback et surcharge la méthode

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

qui sera appelée quand la carte sera disponible

Fragments et Google Maps

- Le fragment en XML :

```
<fragment
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/nomDeMonFragment"
        android:name="com.google.android.gms.maps.SupportMapFragment" />
```

- Dans le manifest :

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

ET pour autoriser OpenGL (utilisé par Maps)

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

ET Enfin pour la clé

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

Interfaces avancées
Styles et Thèmes

Styles (définition)

- Un style est un ensemble de propriétés s'appliquant à un widget
- On le définit comme une ressource (res/values) avec la balise <style> :

```
<style name =  "enVert" >
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
</style>
```

- On l'applique en suite au widget :

```
<TextView
    style = "@style/enVert"
    android:text = "@string/duStyle" />
```

Styles (création)

- Il existe de nombreux styles prédéfinis

- On peut les utiliser directement

OU

- En créer de nouveaux par héritage :

```
<style name =  "mediumVert" parent = "@android:style/TextAppearance.Medium">
        <item name = "android:textColor">#00FF00</item>
</style>
```

- On peut alors :
  - Ajouter des propriétés
  - En surcharger

- Syntaxe de l'héritage
  - Si on hérite d'un style prédéfini : parent = "@android:style/nom"
  - Si on hérite d'un style défini par nous : <style name = "mediumVert.Grand">

Styles (partiels)

- Un style peut être défini pour une seule propriété d'un widget et non pour tout le widget :

```
<style name = "numerique">
    <item name="android:inputType">number</item>
</style>
```

- On l'utilise alors par :

```
<EditText ...
    style="@style/numerique"
    ...
/>
```

Thèmes

- C'est un style qui s'applique à toute l'application ou à une activité

- On le référence dans le fichier AndroidManifest par :
  <application android:theme="@style/ThemeChoisi"> pour l'application
  <activity android:theme="@style/ThemeChoisi"> pour une activité

- Il existe des quantités de thèmes prédéfinis comme : Theme.Dialog , Theme.Light , Theme.Translucent , …

- On peut créer un thème par héritage d'un autre :

```
<color name = "maCouleur">#b0b0ff</color>
<style name = "MonTheme" parent = "android:Theme.Light"  >
        <item name = "android:windowBackground">@color/maCouleur </item>
        …
</style>
```

Et dans le AndroidManifest on met :

```
<activity android:theme="@style/MonTheme">
```

Interfaces avancées
Barre d’actions (ActionBar)

ActionBar

- Barre qui apparaît en haut de la fenêtre

Icône de l'application
Nom de l'application
Actions
Actions supplémentaires

• L'activité doit hériter de AppCompatActivity

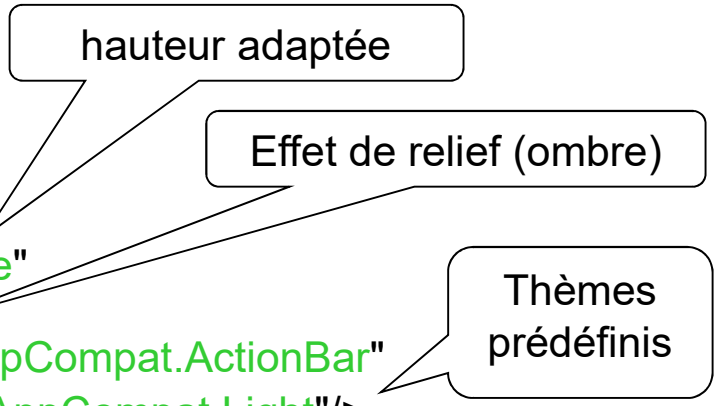
• Le thème de l'application doit être de type : NoActionBar (si on veut assurer la compatibilité !) :

```
<application android:theme="@style/Theme.AppCompat.Light.NoActionBar"  />
```

ActionBar (interface)

- Le fichier XML de l'interface doit contenir la description de la barre d'action (Toolbar) :

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



- On met en place la barre dans le onCreate par :

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

ActionBar (contenu)

- L'ajout de boutons dans la barre se fait par un fichier XML de type menu :

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

On peut mettre :

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

visibilité peut être :

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

ActionBar (actions)

- L'écouteur d'événements associé aux boutons d'action de la barre s'écrit par surcharge de la méthode onOptionsItemSelected de l'activité (comme pour un menu général) :

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

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

            default:
                return super.onOptionsItemSelected(item);

        }
}
```

ActionBar (navigation entre sous activités)

- Normalement quand une sous activité se termine on revient à celle qui l'a lancée (empilement d'écrans)
- On peut ajouter un bouton à la barre d'actions pour quitter la sous activité actuelle et revenir à une autre (pas forcément celle qui l'a lancée). On l'active par :

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

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

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

 <activity
      android:name = ".sousActivite1 "
      android:label = "@string/nomSousActivite1 "

      android:parentActivityName = ".ActivitePrincipale" >

      <meta-data
            android:name = "android.support.PARENT_ACTIVITY"
            android:value = ".ActivitePrincipale" />
</activity>
```

Activité principale

Sous activité

On veut qu'elle retourne à l'activité principale

Pour la compatibilité avant 4.0

Qualification des ressources

Internationalisation

- Constantes

  Android les cherche par défaut dans res/values/xxx.xml mais si on crée res/values-fr/xxx.xml et que le système est en français il ira les chercher là.

- Interfaces

  Même principe avec res/layout-fr/main.xml

- Images

  Même principe avec res/drawable-fr/

- Formats (dans le code) il existe des classes d'utilitaires :
  - Pour les dates : DateUtils et DateFormat
  - Pour les nombres : DecimalFormat
  - Pour les n°s de téléphone : PhoneNumbersUtils

Gestion des versions d'Android

- Dans AndroidManifest.xml balise <uses-sdk …>
  - android:minSdkVersion = version minimale acceptée
  - android:targetSdkVersion = version pour laquelle l'application a été développée
  - android:maxSdkVersion = **à éviter**

- On peut proposer des fichiers XML d'interface différents selon la version d'Android en utilisant les répertoires :
  - res/layout/ (par défaut)
  - res/layout-v4/ (à partir d'Android 1.6)
  - res/layout-v11/ (à partir d'Android 3.0)
  - …

- Dans le code on peut tester la version par Build.VERSION.SDK dont les principales valeurs sont :
  - FROYO (2.2 API 8)
  - GINGERBREAD (2.3 API 9 ou 10)
  - HONEYCOMB (3.0 API 11 à 13)
  - ICE_CREAM_SANDWICH (4.0 API 14 ou 15)
  - JELLY_BEAN (4.1 API 16 à 18)
  - KITKAT (4.4 API 19 ou 20)
  - LOLLIPOP (5.0 API 21 à 22)
  - MARSHMALLOW (6.0 API 23)
  - …

Ressources en fonction de l'écran

- Les images sont mises dans res/drawable, les icones dans res/mipmap. Les répertoires res/drawable-*yyy* et res/mipmap-*yyy* correspondent aux différentes densités d'écran :
  - ldpi : 120dpi
  - mdpi : 160 dpi
  - hdpi : 240 dpi
  - xhdpi : 320 dpi
  - xxhdpi : 480 dpi
  - xxxhdpi : 640 dpi
- Les fichiers xml d'interface sont dans res/layout. Les répertoires suivants correspondent à des tailles d'écran :
  - res/layout — écran normal
  - res/layout-large — grand écran
  - res/layout-xlarge — très grand écran

Combinaisons de qualifications

- Les noms de répertoires dans *res* permettent de les qualifier en fonction de (liste non exhaustive) :
  - La langue (fr …)
  - La direction de l'écran (ldrtl gauche à droite ou ldltr droite à gauche)
  - La taille de l'écran (small normal large xlarge)
  - L'orientation de l'écran (port land)
  - La densité de l'écran (ldpi mdpi …)
  - La version de l'API (v3 v11 …)
- On peut les combiner à condition de respecter l'ordre ci-dessus :
  - res/layout-fr-xlarge-land est correct
  - res/drawable-fr-mdpi-v11 est correct
  - res/layout-xlarge-fr est incorrect
  - res/layout-v11-xlarge est incorrect

Prise en compte des caractéristiques de l’écran

L'écran

Récupérer un objet de classe Display :

```
WindowManager wm = (WindowManager) context.
                        getSystemService(Context.WINDOW_SERVICE);
Display ecran = wm.getDefaultDisplay();
```

Taille de l'écran :

```
Point taille;
ecran.getSize(taille);
```

taille.x et taille.y sont les dimensions de l'écran en pixels

Rotation de l'écran :

ecran.getRotation() renvoie Suface.ROTATION_0 ou _90 ou _180 ou _270

Vitesse de rafraîchissement :

ecran.getRefreshRate() renvoie la vitesse de rafraîchissement de l'écran en images par seconde

Métriques de l'écran

Récupérer un objet de classe DisplayMetrics :

```
DisplayMetrics metrique = new DisplayMetrics();
getWindowManager().getDefaultDisplay().getMetrics(metrique);
```

- metrique.widthPixels et heighPixels = taille de l'écran en pixels
- metrique.densityDpi = densité de l'écran en dpi
- metrique.xdpi et ydpi = pixels par pouces en X et Y
  (correspond au **dp** dans les fichiers XML)
- metrique.scaledDensity = facteur d'échelle pour les polices
  (correspond au **sp** dans les fichiers XML)

Gérer la rotation d'écran

- Bloquer la rotation d'écran
  - Ajouter dans AndroidManifest (dans la balise <Activity>) :
    android:screenOrientation = "portrait" ou "landscape"
- Gérer la rotation
  - Lors d'une rotation d'écran l'activité est **détruite puis recréée**
  - Il faut sauvegarder son état à la destruction pour le restaurer à la création. La sauvegarde de l'état se fait dans un Bundle par couples clé/valeur
  - C'est faisable en surchargeant les méthodes :

```
public void onSaveInstanceState(Bundle etat) {
    etat.putXxx(cle1, valeur1); // putInt , putString , etc
    ….
    super. onSaveInstanceState(etat);
}
```

Et

```
public void onRestoreInstanceState(Bundle etat) {
    super.onRestoreInstanceState(etat);
    valeur1 = etat.getXxx(cle1);
    ….
}
```

Interactions avancées

Suivi de toucher (écouteur)

- Associer au Widget (View) un écouteur par setOnTouchListener(new Ecouteur());

- L'écouteur implémente OnTouchListener
  Il surcharge la méthode :
  public boolean onTouch(View *vue*, MotionEvent *evenmt*)

- Dans cette méthode on récupère l'action par :
  int action = evenmt.getActionMasked()

Ses valeurs sont :

- MotionEvent.ACTION_DOWN *// contact à 1 doigt*
- MotionEvent.ACTION_POINTER_DOWN: *// contact à plusieurs doigts*
- MotionEvent.ACTION_MOVE *// déplacement avec 1 ou plusieurs doigts*
- MotionEvent.ACTION_UP: *// fin de contact*
- MotionEvent.ACTION_CANCEL: *// Si le touché est récupéré par la vue parente*
- MotionEvent.ACTION_OUTSIDE: *// Si le touché est hors zone*

Suivi de toucher (traitement)

- Pour ACTION_DOWN :
  - Position par *evenmt*.getX() et *evenmt*.getY()
- Pour ACTION_POINTER_DOWN :
  - Nombre de doigts par *evenmt*.getPointerCount()
  - Positions par *evenmt*.getX(*num*) et *evenmt*.getY(*num*)
    num entre 0 et *evenmt*.getPointerCount() -1
- Pour ACTION_MOVE et ACTION_UP
  - Traitement selon que l'on a eu auparavant ACTION_DOWN ou ACTION_POINTER_DOWN
- Positions antérieures par
  - *evenmt*.getHistoricalX(*num*, *position*) et
  - *evenmt*.getHistoricalY(*num*, *position*)
    position est limité à 0… *evenmt*.getHistorySize()-1

Suivi par détecteur de geste

- Classe permettant de détecter des gestes On le crée en lui associant un écouteur d'événements :

```
detecteur = new GestureDetectorCompat(contexte, new Ecouteur());
```

- Il analyse les mouvements que l'on lui transmet par sa méthode onTouchEvent ceci sera fait par surcharge de onTouchEvent de l'activité :

```
@Override
public boolean onTouchEvent(MotionEvent evt) {
    detecteur.onTouchEvent(evt); // Transmission de l'événement
    return super.onTouchEvent(evt);
}
```

- L'écouteur hérite de GestureDetector.SimpleOnGestureListener et posède des méthodes appelées quand les mouvements suivants sont détectés :
  - Mouvement de « tap »
  - Mouvement de « double tap »
  - Mouvement de « jeter »
  - Mouvement de « glisser »

Ecouteur du GestureDetector

- L'écouteur hérite de GestureDetector.SimpleOnGestureListener
- La méthode onDown doit être surchargée pour renvoyer true (par défaut elle renvoie false et aucun geste n'est détecté) :

```
@Override
public boolean onDown(MotionEvent evt) {
        return true;
}
```

- D'autres méthodes peuvent être surchargées si nécessaire :
  - Mouvement de « tap »
    public boolean onSingleTapConfirmed(MotionEvent *evt*)
  - Mouvement de « double tap »
    public boolean onDoubleTap(MotionEvent *evt*)

Ecouteur du GestureDetector

- D'autres méthodes peuvent être surchargées si nécessaire :
  - Mouvement de « jeter »

    public boolean onFling(MotionEvent *evt1,* MotionEvent *evt2*, float *vitX*, float *vitY*)
    - *evt1* est l'événement de début du geste
    - *evt2* est l'événement actuel
    - *vitX* et *vitY* indiquent la vitesse du mouvement en pixels/seconde

  - Mouvement de « glisser »

    public boolean onScroll(MotionEvent *evt1,* MotionEvent *evt2*, float distX, float distY)
    - *evt1* est l'événement de début du geste
    - *evt2* est l'événement actuel
    - *distX* et *distY* indiquent la distance parcourue en pixels depuis le précédent appel de onScroll

Animations automatiques

Animations

- L'animation est décrite dans un fichier xml placé dans res/anim/

- Récupérée par :

```
Animation monAnim = AnimationUtils.loadAnimation(getApplicationContext(),
                                                 R.anim.nom_du_xml);
```

- Exécutée par :

```
widget_a_animer = (xxxView) findViewById(R.id.nom_du_widget_a_animer);
widget_a_animer.startAnimation(monAnim);
```

- Association d'écouteurs à l'animation par :

```
monAnim.setAnimationListener(new Ecouteur());
```

Avec :

```
Class Ecouteur implements AnimationListener {
    public void onAnimationStart(Animation animation) { … }
    public void onAnimationEnd(Animation animation) { … }
    public void onAnimationRepeat(Animation animation) { … }
}
```

Animations (fichiers XML)

- Structure :

```
<?xml version="1.0" encoding="utf-8"?>
    <set xmlns:android=http://schemas.android.com/apk/res/android
        paramètres_d_ensemble >

        <type paramètres_de_type />
        …
        <type paramètres_de_type />
    </set>
    …
```

Paramètres d'ensemble :

S'appliquent à l'ensemble

android:ordering="sequentially" ou "together"
android:fillAfter="true" ou "false"
android:duration= "en_ms"
android:startOffset= "en_ms"
android:repeatCount= "nbr" ou "infinite"
android:repeatMode="respeat" ou "reverse"

types et paramètres de type :

S'appliquent à un seul type

Type	Paramètres_de_type
alpha	android:fromAlpha="*float*" android:toAlpha="*float*"
scale	android:fromXScale="*float*" android:toXScale="*float*" android:fromYScale="*float*" android:toYScale="*float*" android:pivotX="*float*" android:pivotY="*float*"
translate	android:fromXDelta="*float*" android:toXDelta="*float*" android:fromYDelta="*float*" android:toYDelta="*float*"
rotate	android:fromDegrees="*float*" android:toDegrees="*float*" android:pivotX="*float*" android:pivotY="*float*"

Animation (effets)

- Ajouter en propriété d'ensemble ou de type un interpolateur par :
  android:interpolator = "@android:anim/nom d'interpolateur"

  Les interpolateurs prédéfinis dans Android sont :
  - @android:anim/accelerate_interpolator
  - @android:anim/decelerate_interpolator
  - @android:anim/accelerate_decelerate_interpolator
  - @android:anim/anticipate_interpolator
  - @android:anim/anticipate_overshoot_interpolator
  - @android:anim/overshoot_interpolator
  - @android:anim/bounce_interpolator
  - @android:anim/cycle_interpolator
  - @android:anim/linear_interpolator

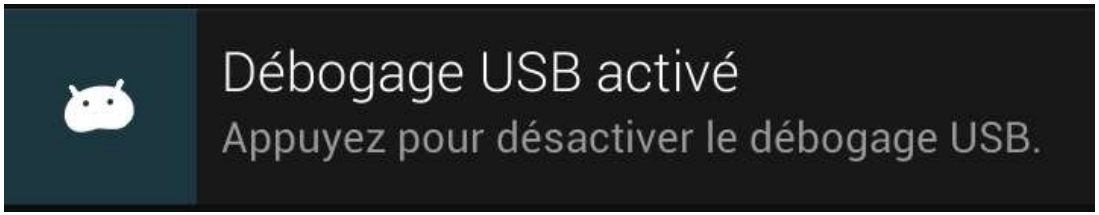
Il sont paramétrables (un peu) par des fichiers XML

- On peut écrire son propre interpolateur par héritage de la classe Animator
  Les actions liées à l'animation sont dans le constructeur de la classe et on utilisera un écouteur (qui hérite de AnimationListener) pour gérer le début / la fin / la répétition de l'animation

- Il sera lancé par :
  widget_a_animer.startAnimation(new MonAnimation());

Notifications

Notification



- Une notification informe l'utilisateur de l'arrivée d'un événement (mail, mise à jour disponible, …). Elle apparaît en haut de l'écran sous la forme d'une icône, d'un titre et d'un texte.

- Lorsqu'il clique dessus une activité correspondante est lancée.

- Démarche :
  - Utiliser NotificationCompat.Builder pour créer le contenu de la notification
  - Créer un PendingIntent contenant un Intent pour indiquer quelle activité doit être lancée par la notification
  - Utiliser NotificationManager pour afficher la notification

Notification

- Créer le contenu de la notification :

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

- Indiquer quelle activité doit être lancée par la notification :

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

Ce paramètre est un n° de PendingIntent qui sert à les différencier si on en créé plusieurs

- Afficher la notification :
- int identNotification = 1; *// Identifiant de la notification (comme pour startActivity for result)*

```
NotificationManager gestNotif = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
gestNotif.notify(identNotification, constr.build()); // La notification apparaît
```

- Supprimer la notification :

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

Bases de données

SQLiteDatabase (BD)

- Création/Ouverture (renvoie un objet de classe SQLiteDatabase)
  openDatabase(String chemin, CursorFactory curs, int flags)
  - Le 2ème paramètre est d'une classe héritant de CursorFactory si on veut que les valeurs renvoyées ne soient pas de classe Cursor mais d'une classe à nous sinon on met **null**.
  - flags : OPEN_READWRITE , OPEN_READONLY , CREATE_IF_NECESSARY.

- Suppression
  deleteDatabase(File chemin)

- Transactions
  beginTransaction()
  endTransaction()
  beginTransactionWithListener (SQLiteTransactionListener ecouteurDeTransaction)
  SQLiteTransactionListener possède les méthodes (à surcharger)
  - onBegin()
  - onCommit()
  - onRollback()

- Commandes
  execSQL(String sql)
  query(String table, String[] colonnes, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit) renvoie un objet de classe Cursor

Cursor (tuples)

- Position/Déplacements
  getPosition()
  move(int offset) , moveToPosition(int position)
  moveToFirst() , moveToLast() , moveToNext() , moveToPrevious()
  getColumnIndex(String nomDeColonne)

- Contenu
  getCount()
  getColumnCount()

- Acces au contenu
  getXxx(int indexDeColonne)
  Xxx = Double, Float, Int, Short, Long, String, Extras

- Fermeture
  close()

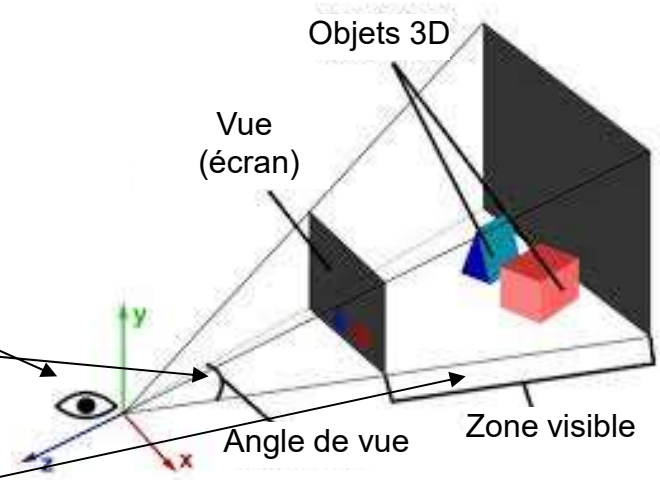
3D avec openGL

La 3D

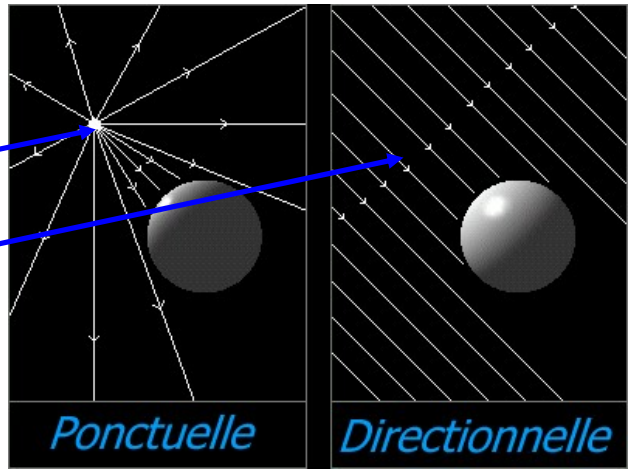
- Le linux d'Android contient un openGL (pas tout à fait compatible) dont les fonctions correspondent aux méthodes de la classe GL10
- Le widget dans l'interface est un GLSurfaceView (ou classe dérivée)
- On lui associe un mode de rafraîchissement par :
  setRenderMode(mode)
  Où mode vaut GLSurfaceView.RENDERMODE_WHEN_DIRTY
  ou GLSurfaceView.RENDERMODE_CONTINUOUSLY
- On lui associe un écouteur d'événements qui implémente Renderer par :
  setRenderer(ecouteur)
- Dans cet écouteur, on surcharge les méthodes :
  - onSurfaceCreated(GL10 gl10, EGLConfig config) *// création*
  - onDrawFrame(GL10 gl10) *// rafraîchissement*
  - onSurfaceChanged(GL10 gl10, int width, int height) *// changement de taille*

La vue 3D

- Placement de la caméra (coordonnées x, y, z)
- Réglage de l'angle de vue
- Réglage de la profondeur de champ (zone visible)

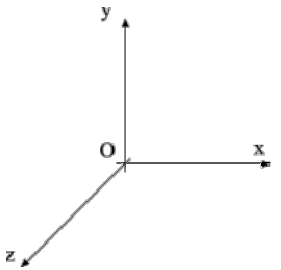


- Ajout des éclairages (8 maximum)
  - Position (coordonnées x, y, z)
  - Type spot (ponctuelle)
    ou à l'infini (directionnelle)
  - Couleur

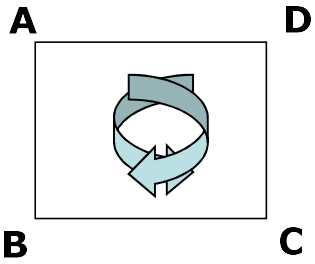


Les objets 3D (tracé)

- Un volume est défini par des surfaces
  - 6 surfaces (carrés) pour un cube
- Une surface est définie par des points
  - 4 points pour un carré
- Un point est défini par 3 coordonnées (x, y z)

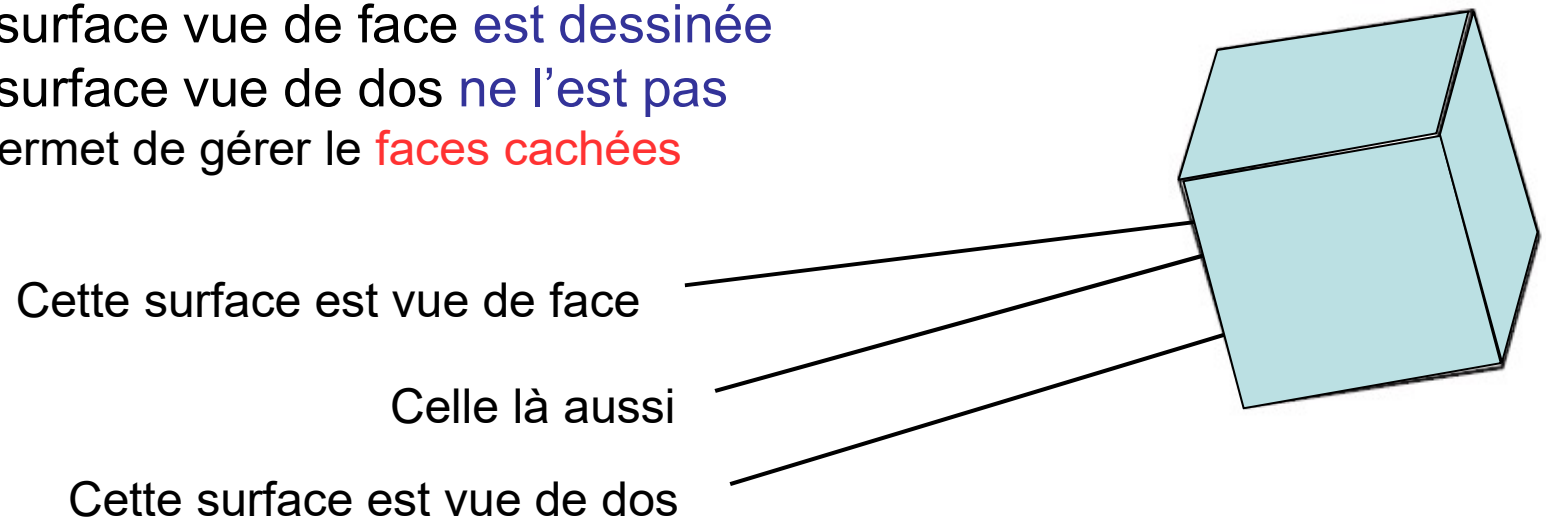


- OpenGL a des fonctions de dessin de polygones (surface définie par N points ne comportant pas de trous)
- Une surface a un coté face et un coté dos
  - Le sens du tracé est important
    - Dessin par A puis B puis C puis D ⇒ vue de face
    - Dessin par A puis D puis C puis B ⇒ vue de dos
    - Autre ordre interdit



Les objets 3D (affichage)

- Une surface vue de face est dessinée
- Une surface vue de dos ne l'est pas
  - Permet de gérer le faces cachées



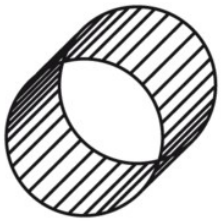
- Au final ça donne

- Une surface dessinée doit avoir une couleur

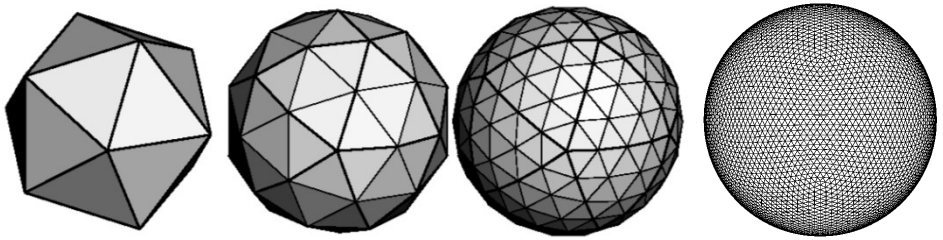
- La couleur est généralement uniforme sur toute la surface
  Mais il existe des fonctions pour obtenir des dégradés (shaders)

Surfaces courbes

- Un polygone est une surface plane
- Pour faire des surfaces courbes il faut les découper en polygones
  - Par exemple un cylindre est découpable en rectangles



  - Une sphère est découpable en triangles



- Plus le découpage est fin plus la surface paraît courbe
  (à la limite découpage en surfaces de 1 pixel)
  Mais plus le nombre de surfaces à dessiner est élevé ⇒ temps de calcul

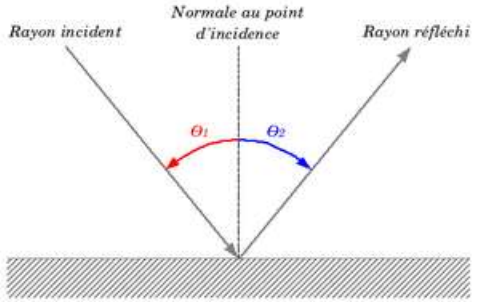
Objets 3D éclairage

- OpenGL éclaire les objets en utilisant les lois de l'optique

- Atténuation réglable par 3 coefficients

$$\frac{1}{k_c + k_l d + k_q d^2}$$

k_c = atténuation constante
k_l = atténuation linéaire
k_q = atténuation quadratique
d = distance (spot , point de contact)

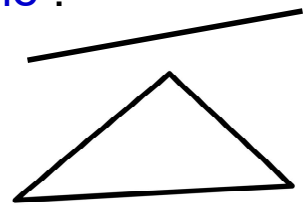


- Mais au niveau d'une surface
  - Le calcul n'est fait que pour un point de la surface et on considère qu'il est le même pour tous
  - Ce n'est pas réaliste car une surface éclairée par un spot sera plus éclairée en certains point qu'en d'autres

- Solution :
  - Découper une surface en surfaces plus petites
  - Le calcul sera fait pout chaque partie
  - Faire des parties de petite taille ⇒ beaucoup de calculs

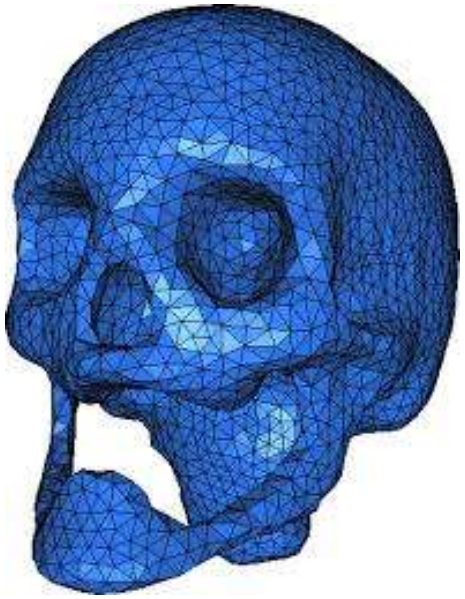
Objets 3D (triangulation)

- Choix de la surface utilisée pour le découpage :
  - Le triangle est la surface la plus simple (celle définie par le moins de points)
  - C'est la seule surface toujours plane :
    - 2 points définissent une droite
    - 3 points définissent un plan
    - Au-delà de 3 points ils ne sont pas obligatoirement dans le même plan

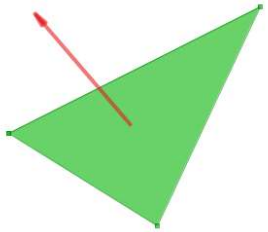


  - OpenGL ne calcule l'éclairage que pour des surfaces planes (lois de l'optique)
- Une bonne pratique est donc de diviser les surfaces en petits triangles (triangulation)

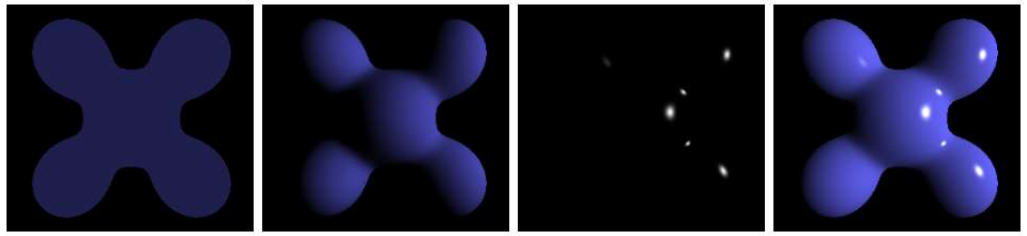
Les objets 3D



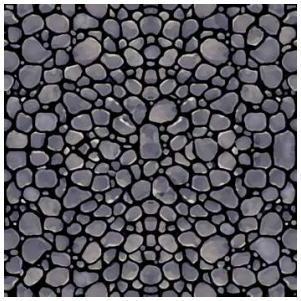
- Dessinés par triangulation
- A chaque triangle on doit associer une normale pour le calcul d'éclairage



- On lui applique :
  - Un matériau constitué de :
    - Couleur ambiante
    - Couleur spéculaire
    - Couleur diffusée



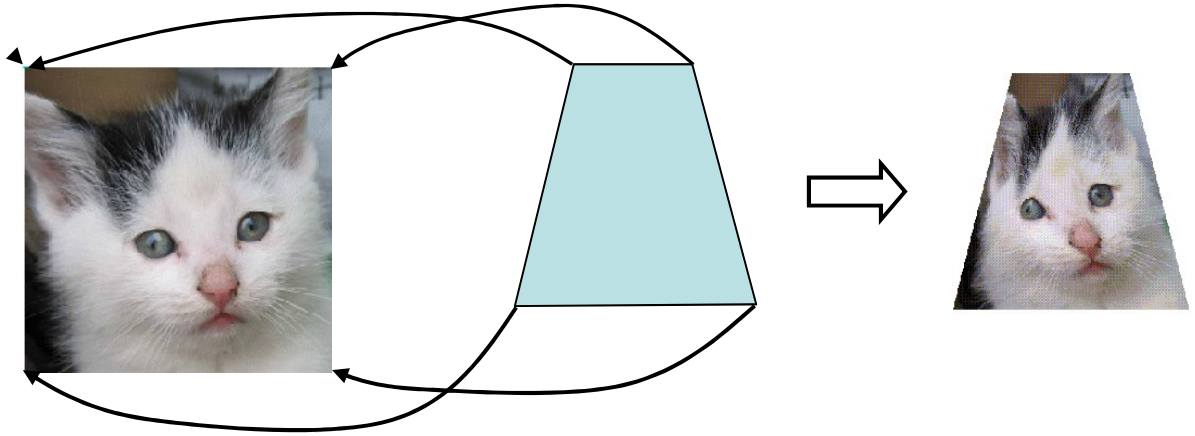
Ambiante + Diffusée + Spéculaire = Aspect



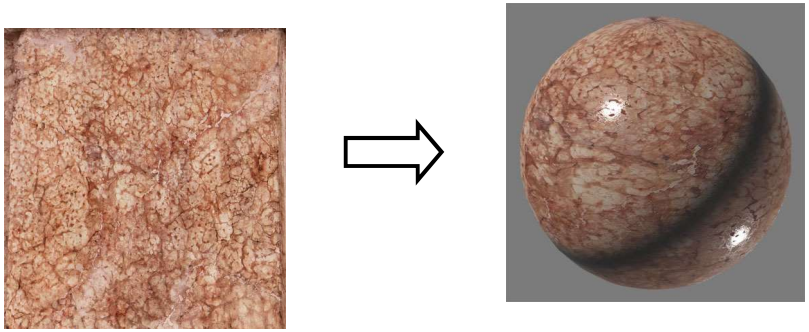
- Une texture (éventuellement)

Texturation

- Consiste à plaquer une image sur une surface
- A chaque point de cette surface (sommet d'un polygone la constituant) on fait correspondre un point de l'image



- OpenGL calcule les points intermédiaires et déforme l'image en conséquence



Le calculs sont complexes ⇒ ralentissement

Des maths ? (un peu oui)

- Ne pas avoir peur de la trigonométrie

  Exemple cœur de l'algorithme de triangulation d'une sphère :

```
int parties = (int)(rayon/precision);
float drho =   (float)Math.PI / parties;
float dtheta = (float)Math.PI / parties;
for (int i = -1; i < parties; i++) {
      rho = i * drho;
      for (int j = 0; j <= parties; j++) {
         theta = j * dtheta;
         x = (float)(-Math.sin(theta) * Math.sin(rho));
         y = (float)(Math.cos(theta) * Math.sin(rho));
         z = (float)(Math.cos(rho));
         // Calcul du vecteur normal (premier point)
         vn = { x, y, z };
         // Calcul des coordonnées de texture (premier point)
         texture = { theta/2/(float)Math.PI, rho/(float)Math.PI };
         // Calcul du premier point de surface
         point = { x * rayon, y * rayon, z * rayon };
         x = (float)(-Math.sin(theta) * Math.sin(rho + drho));
         y = (float)(Math.cos(theta) * Math.sin(rho + drho));
         z = (float)(Math.cos(rho + drho));
         // Calcul du vecteur normal (second point)
         vn = { x, y, z };
         // Calcul des coordonnées de texture (second point)
         texture = { theta/2/(float)Math.PI, (rho+drho)/(float)Math.PI };
         // Calcul du second point de surface
         point = { x * rayon, y * rayon, z * rayon };
      }
}
```

Pour conclure

Android : constat

- Avantages :
  - Possibilités importantes (images, sons, animations, vidéo, 3D, synthèse de parole, BD, web services, communications, …)
  - API riche + modèles (styles, thèmes, …)
  - Internationalisation simplifiée
  - Outils pour matériel différent (écran, versions, …)
  - Interactivité riche (graphique, tactile, capteurs)
  - Diffusion facile (Google Play ou autres stores)

- Inconvénients :
  - API riche mais comment savoir quelle classe ou méthode utiliser ?
  - Documentation abondante mais il est parfois difficile de trouver ce que l'on cherche (Java doc, tutoriels et forums plus ou moins fiables, …)
  - Problèmes de versions (*entre septembre 2008 (version 1) et décembre 2015 (version 6.0.1) il y en a eu 45*) , de tailles d'écran, de périphériques différents, …
  - Adapter l'interactivité aux publics et aux périphériques
  - Difficile de sortir du lot (beaucoup d'applications existent et beaucoup apparaissent chaque jour).

Et le reste ?

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

- Ecriture de services
- Ecriture de fournisseurs de contenu
- Création de composants d'interface personnalisés
- Ecriture de pilotes permettant de modifier l'interface depuis l'extérieur de l'activité
- Sécurité
- Communication par réseau (sockets , HTTP, SOAP, REST …)
- Sérialisation (Parcelable , JSON)
- Utilisation du Bluetooth
- …


That's all Folks!