Chapitre 1

Découvrir la plateforme Android

Plan du chapitre 1

- La plateforme Android
- L'architecture Android
- Les outils de développement

Android =

- Android = un système d'exploitation open source pour smartphones, PDA, tablettes : systèmes légers
- = une startup rachetée en août 2005 par Google
- 5 novembre 2007, création de l'OHA (Open Handset Alliance) = un consortium créé à l'initiative de Google réunissant des entreprises opérateurs mobiles, constructeurs et éditeurs logiciels
- But de l'OHA = favoriser l'innovation sur les appareils mobiles en fournissant une plate-forme véritablement ouverte et complète
- est gratuit
- Mais très souvent Android signifie environnement d'exécution

La documentation Android

- logo bugdroid (peut être utilisé librement)

- noyau Linux
- site de référence : `http://developer.android.com/index.html`
- On peut retrouver une partie de ce site en local à `%REP_INSTALL_ANDROID_SDK%/docs/index.html`
- source : `http://fr.wikipedia.org/wiki/Android`

Android et Google

- A suscité l'engouement des développeurs grâce à deux Android Developer Challenge en 2008 et 2009 financé par Google
- Conçu pour intégrer les applications Google : Gmail, Google Maps, Google Agenda, YouTube et la géolocalisation
- Les différentes versions ont des noms de dessert (qui suivent l'ordre alphabétique, de A à Z) qui sont sculptés et affichés devant le siège social de Google (Mountain View)

- source : `http://fr.wikipedia.org/wiki/Android`

Les différentes versions (1/2)

- 1.0 : Version connue des développeurs : sortie avant le premier téléphone Android (fin 2007)
- 1.1 : Version incluse dans le premier téléphone, le HTC Dream
- 1.5 : Cupcake (Petit Gâteau), sortie en avril 2009
- 1.6 : Donut (Beignet), sortie en septembre 2009
- 2.0 (2.0.1) : A cause de nombreux bugs, remplacée par la la 2.1
- 2.1 : Eclair, sortie en janvier 2010
- 2.2 : FroYo (Frozen Yogourt : Yaourt glacé), sortie en mai 2010
- 2.3 : Gingerbread (Pain d'épice), sortie le 6 décembre 2010
- 3.0 : Honeycomb10 (Rayon de miel), sortie le 26 janvier 2011

Les différentes versions (2/2)

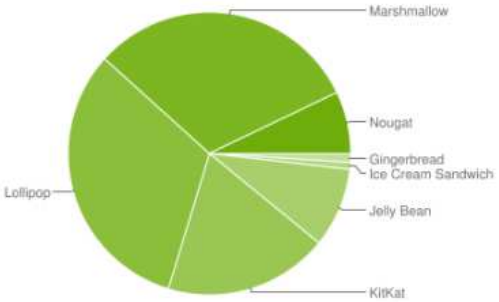
- 4.0 : Ice Cream Sandwich (Sandwich à la crème glacée), version unifiée pour Smartphone, Tablette et GoogleTV, combinant Gingerbread et Honeycomb, sortie le 19 octobre 2011
- 4.1 : API 16 (Jelly Bean) sortie le 9 juillet 2012
- 4.4 : API 19 (KitKat) sortie le 31 octobre 2013
- 5.0 : API 21 (Lollipop) sortie le 15 octobre 2014
- 5.1.1 : API 22 (Lollipop) sortie le 9 mars 2015
- 6.0 : API 23 (Marshmallow) sortie le 5 octobre 2015
- 7.0 : API 24 (Nougat) sortie le 22 août 2016
- 7.1.1 : API 25 (Nougat) sortie le 4 octobre 2016
- Bibliographie : `https://en.wikipedia.org/wiki/Android_version_history`

Parts de chaque version en mai 2017

- Versions des machines Android qui ont accédé au Google Play Store (ex Android Market) une semaine avant le 2 mai 2017
- Voir à `http://developer.android.com/about/dashboards/index.html`

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.8%
4.1.x	Jelly Bean	16	3.2%
4.2.x		17	4.6%
4.3		18	1.3%
4.4	KitKat	19	18.8%
5.0	Lollipop	21	8.7%
5.1		22	23.3%
6.0	Marshmallow	23	31.2%
7.0	Nougat	24	6.6%
7.1		25	0.5%

Data collected during a 7-day period ending on May 2, 2017.
Any versions with less than 0.1% distribution are not shown.



Le Google Play (ex Android Market)

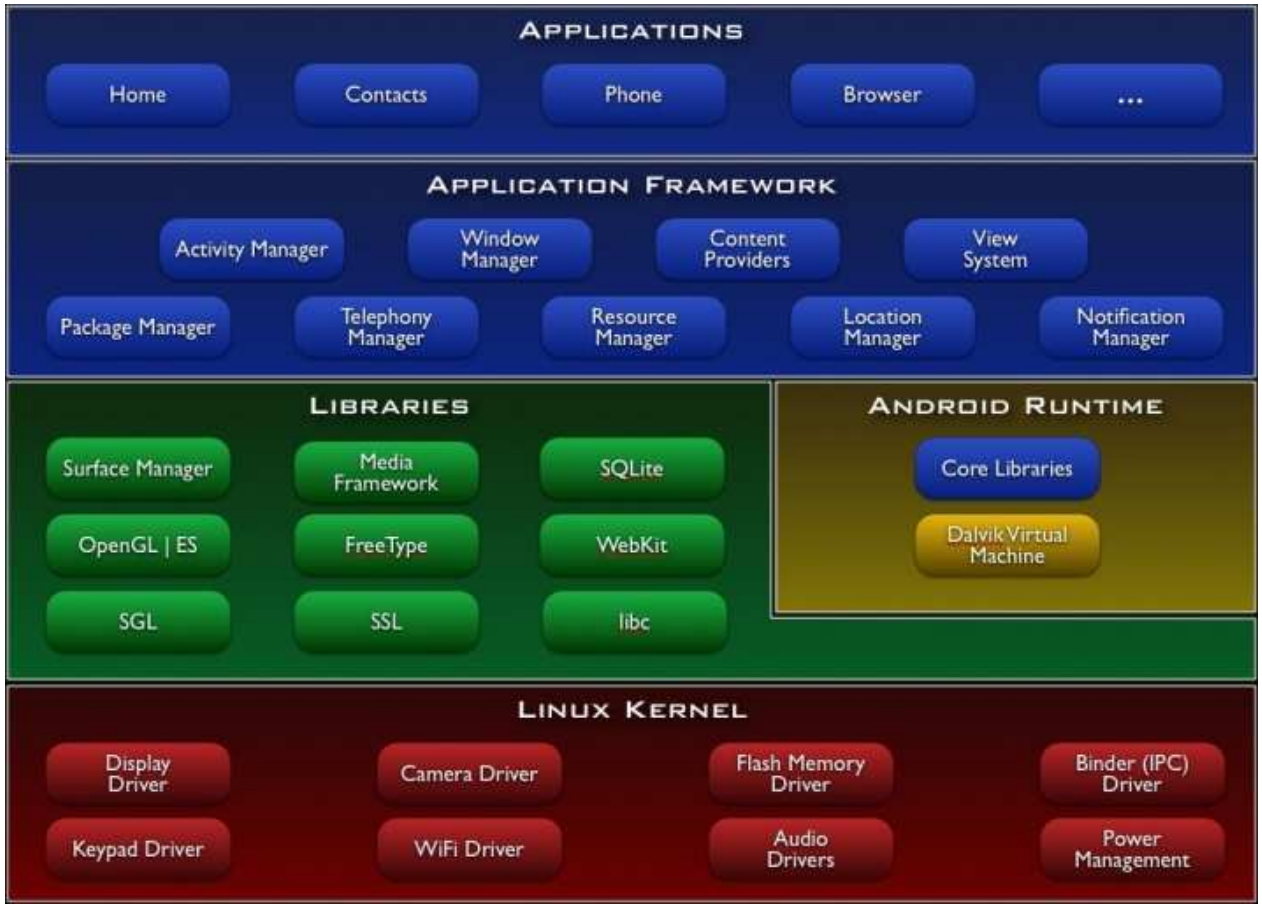
- Google Play (Play Store), est une boutique en ligne créée par Google (le 6 mars 2012) par fusion des services Android Market et d'autres services Google (location de films, achat de musique, etc.). Elle permet de télécharger et d'installer de nouvelles applications ("apps") dans le smartphone
- Android market est "né" le 22 octobre 2008
- "Au 30 octobre 2012, Google Play est fort de 700 000 applications ce qui le met à égalité avec iOS" : voir historique à `http://fr.wikipedia.org/wiki/Android_Market`
- Les développeurs d'applications payantes reçoivent 70 % du prix du logiciel, 30 % allant à Google
- Chaque nouveau développeur paie $25 comme frais de dossier (une seule fois)

Le SDK Android

- l'Android SDK (Software Development Kit) amène des outils :
  - un environnement de développement
  - une machine virtuelle Java adaptée : l'Android RunTime (ART)
  - un environnement debugueur DDMS (Dalvik Debug Monitor Service) utilisant adb (Android Debug Bridge)
  - un environnement de construction d'application Android aapt (Android Asset Packaging Tool)
  - des émulateurs de téléphones ou de tablettes AVD (Android Virtual Device)
- et une énorme API (voir à `http://developer.android.com/reference/packages.html`)

L'architecture Android (1/3)

- Architecture en "pile logicielle"



L'architecture Android (2/3)

- La couche "Applications" est un ensemble contenant des applications comme, un client de mail, des programmes pour envoyer des SMS, pour gérer un agenda, un navigateur web, un gestionnaire de contacts personnels
- La couche "Application Framework" : cette couche permet au programmeur de construire de nouvelles applications. Cette couche fournit la gestion :
  - des `Views` (= IHM)
  - des `ContentProviders` = l'accessibillité aux données des autres applications (ex : les contacts) et donc les partages de données
  - des ressources = les fichiers non codes comme les images, les écrans (Resource Manager)
  - des `Notifications` (affichage d'alerte dans la barre de titre)
  - des `Activitys` = l'enchaînement des écrans

L'architecture Android (3/3)

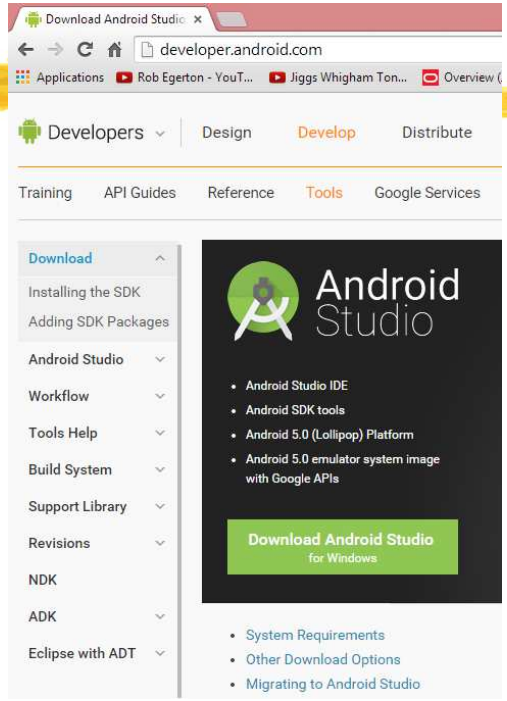
- La couche "`Libraries`" (bibliothèques) = couche logicielle basse écrites en C, C++ (= native libraries) pour utiliser
  - les formats multimédia : images, audio et vidéo enregistrement comme rendu
  - les dessins 2D et 3D, bitmap et vectoriel
  - une base de données SQL (SQLite)
- L'environnement d'exécution (Android Runtime). Toute application est éxécutée dans son propre processus, dans son propre Android RunTime
- Le noyau Linux sur lequel la Dalvik virtual machine s'appuie pour gérer le multithreading, la mémoire. Le noyau Linux apporte les services de sécurité, la gestion des processus, etc.

ART (Android RunTime)

- Depuis la version 4.4, la machine virtuelle Java pour les applications Android est ART
- Conçue pour exécuter du code Java pour des systèmes ayant des contraintes de place mémoire et rapidité d'exécution
- Exécute du code `.dex` (Dalvik executable) = des `.class` adaptés à l'environnement Android
- Avant la version 4.4, la JVM était la Dalvik Virtual Machine (DVM) écrit par Dan Bornstein d'où le nom (= village islandais dont sont originaires certains de ses ancêtres)

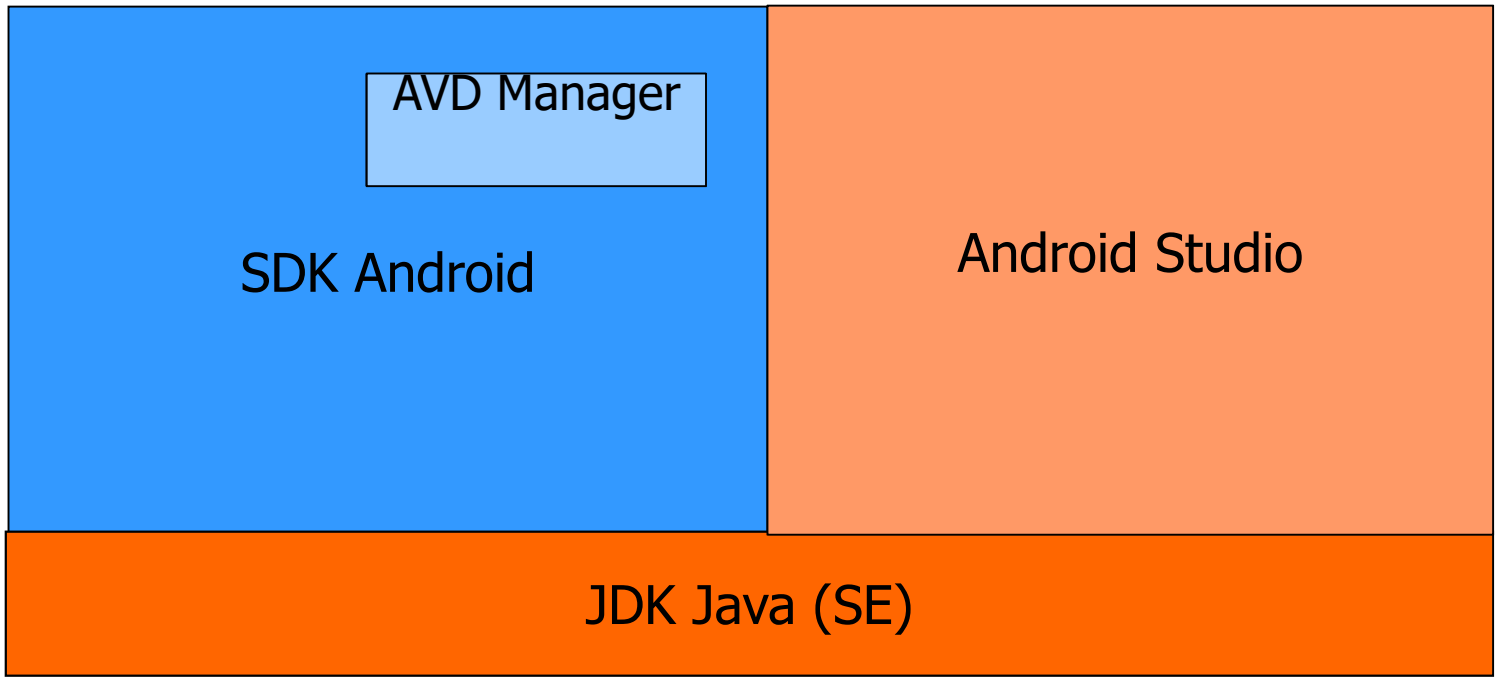
- La DVM est un Just In Time Compiler : compilation à l'exécution
- ART est un ahead-of-time (AOT) compiler : compilation à l'installation de l'app
- référence :
  `https://www.youtube.com/watch?v=ptjedOZEXPM`,
  `https://en.wikipedia.org/wiki/Android_Runtime`

Android Studio



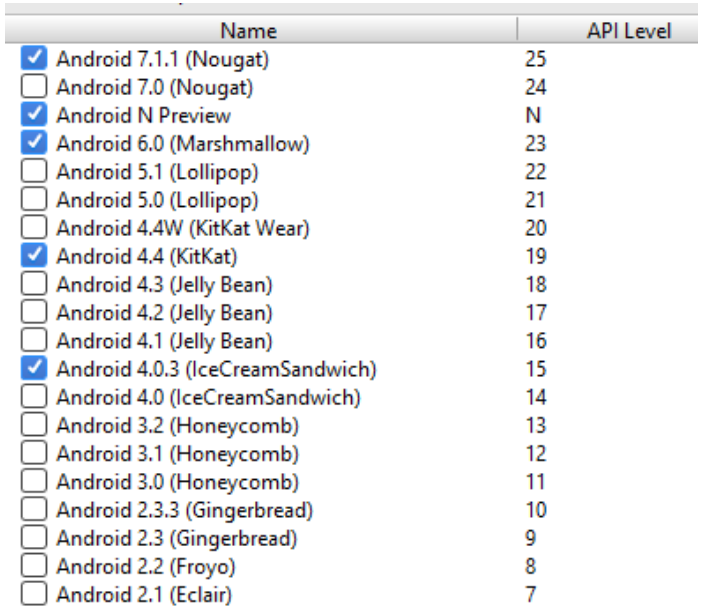
- L'environnement officiel (depuis le 9 décembre 2014) de développement pour Android est Android Studio
- Voir à `http://developer.android.com/sdk`
  On récupère un fichier `.exe` `android-studio-bundle-XXX.exe` de plus de 1,6 Go !
- Android Studio est décrit à `http://developer.android.com/tools/studio/index.html`
- Doc d'installation d'Android Studio à `http://developer.android.com/sdk/installing/index.html?pkg=studio`
- Voir gestion des AVD à `http://developer.android.com/tools/devices/managing-avds.html`

La pile des outils de développement pour Android



Correspondance num Android, num API

- Vous pouvez éventuellement, charger plusieurs "SDK Platform Android" et "Documentation". Pour cela on utilise l'"Android SDK Manager". On obtient :
- Remarquer la correspondance entre les versions du SDK et les numéros d'API, par exemple :
  SDK Android 2.2 <-> API 8
- Voir aussi à `http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels`



Name	API Level
☑ Android 7.1.1 (Nougat)	25
☐ Android 7.0 (Nougat)	24
☑ Android N Preview	N
☑ Android 6.0 (Marshmallow)	23
☐ Android 5.1 (Lollipop)	22
☐ Android 5.0 (Lollipop)	21
☐ Android 4.4W (KitKat Wear)	20
☑ Android 4.4 (KitKat)	19
☐ Android 4.3 (Jelly Bean)	18
☐ Android 4.2 (Jelly Bean)	17
☐ Android 4.1 (Jelly Bean)	16
☑ Android 4.0.3 (IceCreamSandwich)	15
☐ Android 4.0 (IceCreamSandwich)	14
☐ Android 3.2 (Honeycomb)	13
☐ Android 3.1 (Honeycomb)	12
☐ Android 3.0 (Honeycomb)	11
☐ Android 2.3.3 (Gingerbread)	10
☐ Android 2.3 (Gingerbread)	9
☐ Android 2.2 (Froyo)	8
☐ Android 2.1 (Eclair)	7

Le AVD (Android Virtual Device)

- Pour exécuter les programmes Android, il suffit d'un émulateur. C'est le AVD (Android virtual device)
- A la première utilisation il faut en obtenir un
- On lance l"AVD Manager par Tools | Android | AVD Manager ou l'icône

- On peut alors en construire
- Il y a (eu ?) parfois des problèmes de construction d'AVD par Android Studio !

Petits jeux avec les AVD !

- Après un `telnet localhost 5554`, lancer des commandes comme :
  - `network speed edge` pour avoir un réseau edge,
  - `power capacity 10` pour avoir une batterie à 10%,
  - `sms send` *`numTelEmetteur`* `"un message"` pour recevoir un sms provenant de *`numTelEmetteur`*
  - `window scale 0.7` pour avoir une fenêtre d'AVD plus petite
- Voir ces résultats dans la barre de notification
- `geo fix -77.04 38.897` en lançant l'app maps montre Washington DC
- `help` pour avoir les diverses commandes possibles

SD Card

- Une carte SD (SD = Secure Digital) est une carte mémoire amovible de stockage de données numériques créée en janvier 2000 par une alliance formée entre les industriels Panasonic, SanDisk et Toshiba
- Les cartes SD sont utilisées pour le stockage de fichiers dans les divers appareils (appareils photo numériques, les caméscopes numériques, consoles de jeux, smartphones, ...)
- Depuis 2010, les cartes SD font office de standard de stockage, suite à l'abandon progressif des autres formats. En 2013, leur capacité s'échelonne jusqu'à 256 Go
- source : `http://fr.wikipedia.org/wiki/Carte_SD`

AVD et SD Card

- On peut préciser la taille d'une SD Card associé à cet AVD en éditant les caractéristiques de l'AVD (puis bouton Show Advanced Settings)

voire indiquer le fichier qui joue le rôle de SD Card et ainsi partager la même SD Card pour plusieurs AVD

Bibliographie pour ce chapitre

- Le site officiel d'Android : `http://developer.android.com/`
- Pour installer l'environnement de développement Android : `http://developer.android.com/sdk`

Résumé du chapitre 1

- Le terme Android dénote à la fois, une société initiatrice pour le développement d'applications pour smartphones rachetée par Google, un environnement de développement, un environnement d'exécution
- L'environnement d'exécution est une pile logicielle avec, à la base un système d'exploitation Linux
- Pour développer des applications Android on utilise des outils de développement comme Android Studio

Chapitre 2

Développement Android

Plan du chapitre 2

- Les composants d'une application Android : `Activity`, `Service`, `ContentProvider`, `BroadcastReceiver`
- Les `Intent`
- Le manifeste : `AndroidManifest.xml`
- Développer une application Android
- Une première application (app)

Concepts de base

- Le code est écrit en Java (eh oui)
- L'Android SDK compile l'ensemble du développement (code Java, données, fichier de ressources, fichier XML) dans un paquetage Android : un `.apk`
- Toute une application tient dans un `.apk` et un `.apk` = une application
- Les composants fondamentaux d'une application Android sont : `Activity`, `Service`, `ContentProvider`, `BroadcastReceiver`. Donc une application = {composants}
- Certains de ces composants communiquent entre eux à l'aide d'`Intent`

Activity, Service

- Une activité (activity) gère l'affichage et les interactions utilisateurs sur un écran. Les activités sont indépendantes les unes des autres. Une activité est une sous classe de `android.app.Activity`
- Une application peut avoir plusieurs activités pouvant lancer cette application
- Un service (service) est un composant qui est exécuté en tâche de fond. Il ne fournit pas d'interface graphique. Exemple de service : jouer de la musique, rechercher des données sur le réseau. Un service est une sous classe de `android.app.Service`

ContentProvider

- Un fournisseur de contenu (content provider) gère des données partageables. C'est le seul moyen d'accéder à des données partagées entre applications
- Exemple de fournisseur de contenu : les informations de contacts de l'utilisateur du smartphone
- On peut créer un fournisseur de contenus pour des données qu'on veut partager
- On récupère un fournisseur de contenu pour des données partageables en demandant le `ContentResolver` du système et en donnant ensuite, dans les requêtes, l'URI des données. Par exemple :

```
ContentResolver cr = getContentResolver();
cr.query(lURIDesDonnees, ...);
```

- Si le processus qui gère la ressource est inactif, le système le lance

BroadcastReceiver

- Un récepteur d'informations (broadcast receiver) est un composant à l'écoute d'informations qui lui sont destinées. Un tel récepteur indique le type d'informations qui l'intéressent et pour lesquelles il se mettra en écoute. Exemple : appel téléphonique entrant, réception d'un SMS, réseau Wi-Fi connecté, informations diffusées par des applications. Les informations peuvent être envoyées par le système (réception de la liste des réseaux Wi-Fi, ...)
- L'application réceptrice d'informations (c'est à dire possédant un récepteur d'informations) n'a pas besoin d'être lancée. Si elle ne l'est pas, Android la démarre automatiquement
- Un récepteur n'est pas une IHM mais peut en lancer une (éventuellement petite dans la barre de notification), ou peut lancer un service traitant l'arrivée de l'information
- Un récepteur d'informations est une sous classe de `android.content.BroadcastReceiver`

Intent

- Un événement (intent) est une "intention" à faire quelque chose contenant des informations destinées à un composant Android. C'est un message asynchrone
- Les activités, services et récepteurs d'informations utilisent les `Intent` pour communiquer entre eux
- Un `Intent` ne contient pas obligatoirement explicitement le composant qui va le gérer. Dans ce cas c'est un `Intent` implicite. Si l'Intent indique explicitement la classe du composant qui va le gérer c'est un `Intent` explicite
- Un événement est une sous classe de `android.content.Intent`

Composants applicatifs : un résumé

- `Activity` = logique associée à un écran
- `Service` = opération s'exécutant en arrière plan (de l'interface graphique = IHM)
- `ContentProvider` = fournisseur de contenu ! c'est à dire gestionnaire des données partageables et/ou persistentes
- `BroadcastReceiver` = gestionnaire des messages systèmes ou applicatifs
- `Intent` = message indiquant une intention à faire une certaine action

Le manifeste : `AndroidManifest.xml`

- Toute application doit contenir le fichier XML `AndroidManifest.xml`
- Ce fichier déclare, entre autre, les différents composants de l'application
- Parmi les composants, seuls les récepteurs d'évènements (`BroadcastReceiver`) ne sont pas forcément dans le manifeste. Les autres (`Activity`, `Service`, `ContentProvider`) doivent l'être sinon ils ne seront jamais lancés quel que soit le code écrit !
- Ce fichier contient aussi :
  - les permissions nécessaires à l'application (accès internet, accès en lecture/écriture aux données partagées)
  - des logiciels et matériels utilisés par l'application (caméra)
  - des données pour des bibliothèques supplémentaires à l'API de base (Google maps library)

Développer une application Android

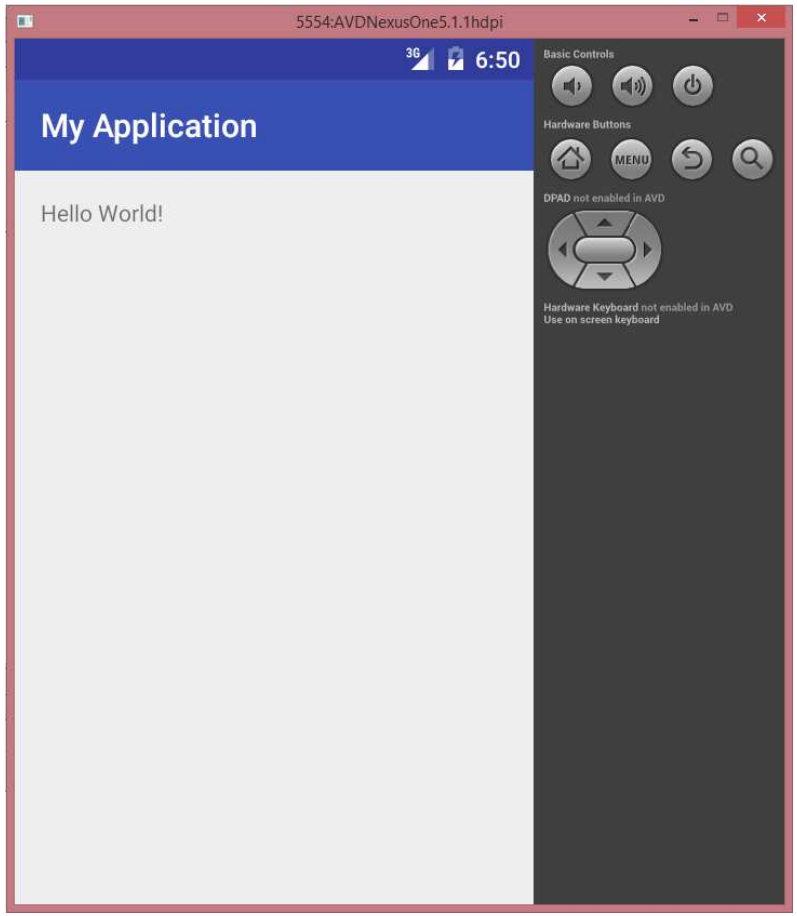
- De l'installation de l'environnement à la publication de l'application
- Les phases sont :
  - installation de l'environnement (setup) = installer le SDK et les AVDs Android Virtual Device
  - le développement = développer le code et les ressources de l'application
  - débuguer et tester = DDMS et JUnit
  - publier sur le Google Play
- On peut faire ces diverses phases en ligne de commandes ou avec Android Studio

- source : `http://developer.android.com/guide/developing/index.html`

Un premier projet : Hello World

Hello World en Android

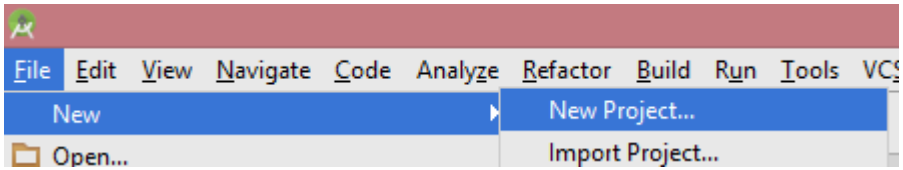
- On veut obtenir :



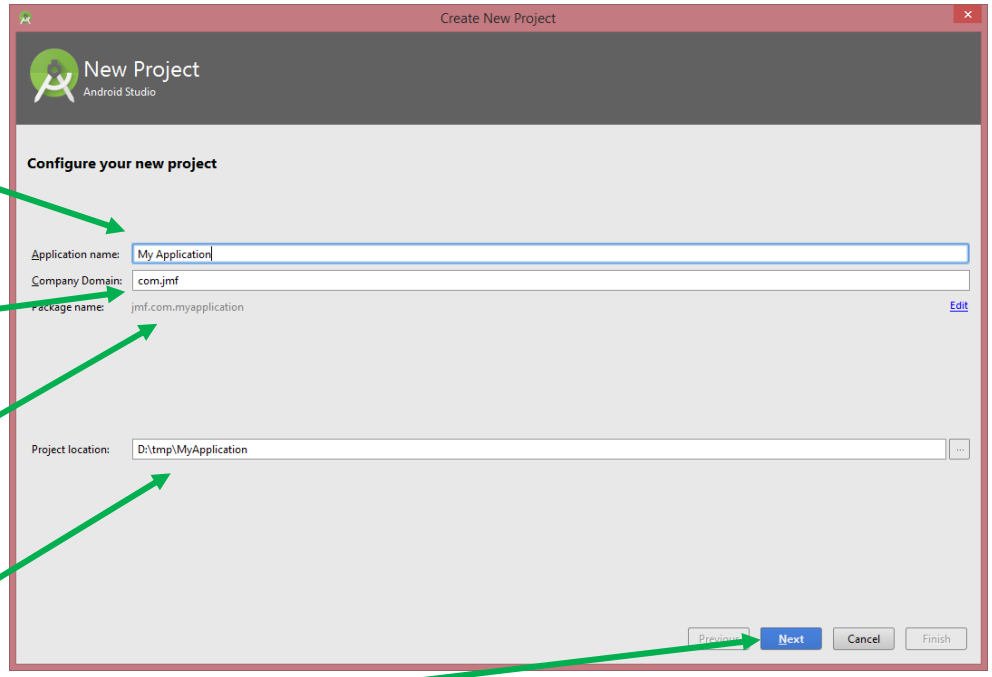
- <u>Remarque :</u>
  Suivant les versions des environnements que vous avez, vous pouvez avoir des fenêtres légèrement différentes des captures d'écran des diapos suivantes

Développement du projet Hello World (1/6)

- Dans Android Studio, choisir File | New | New Project…



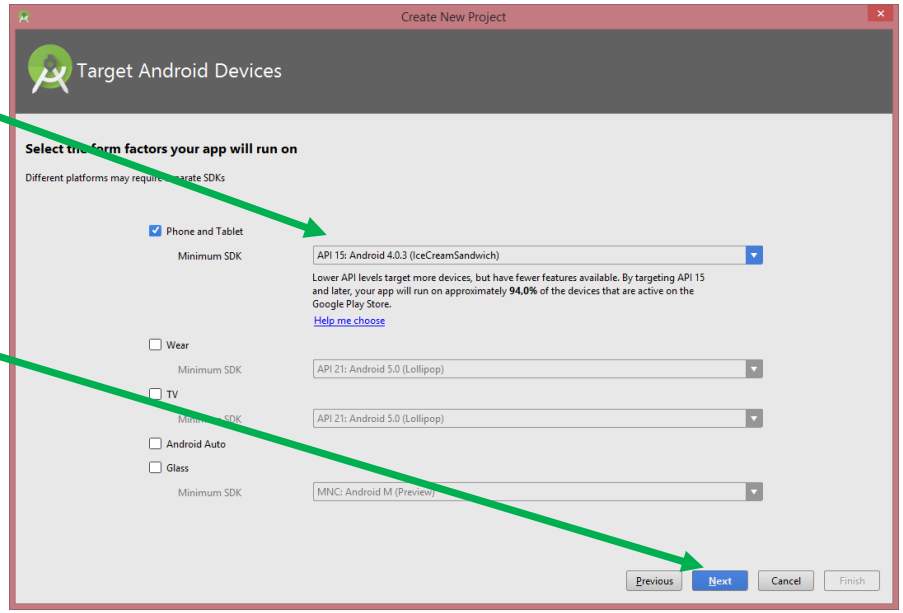
- On obtient la fenêtre :



- Mettre le nom de l'app
- Le nom de domaine de votre entreprise
- Le nom du paquetage de l'app est alors construit
- Indiquer où cette app est sauvegardé
- Cliquez Next

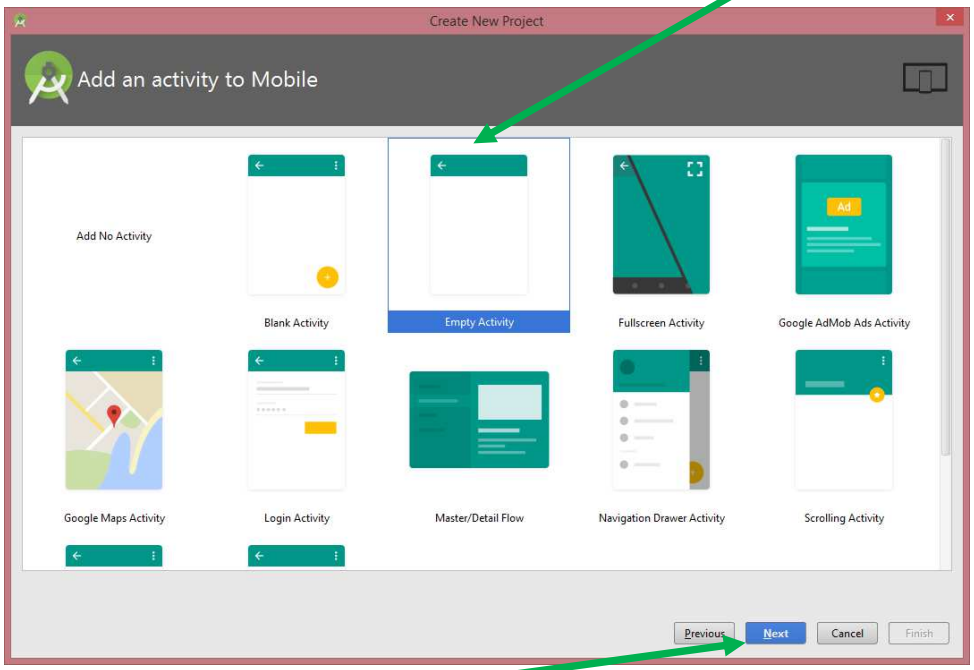
Développement du projet Hello World (2/6)

- Dans l'écran suivant, choisir le numéro de version minimal pour l'app
- Cliquez Next



Développement du projet Hello World (3/6)

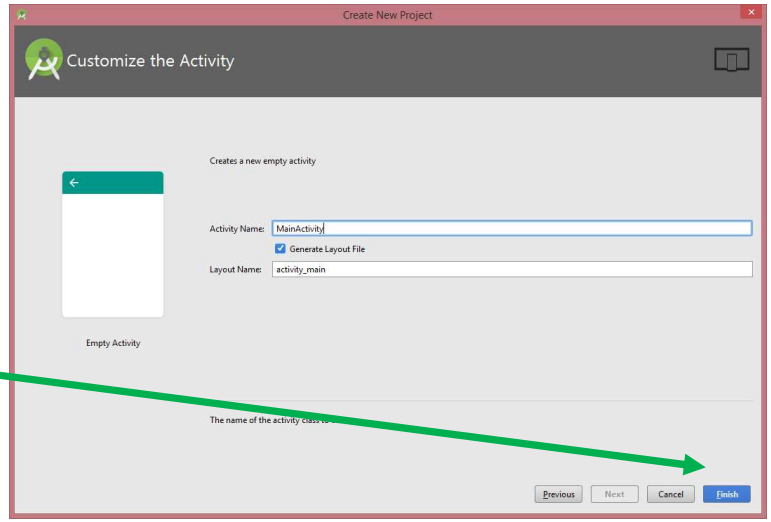
- Dans l'écran suivant, choisir Empty Activity



- Cliquez Next

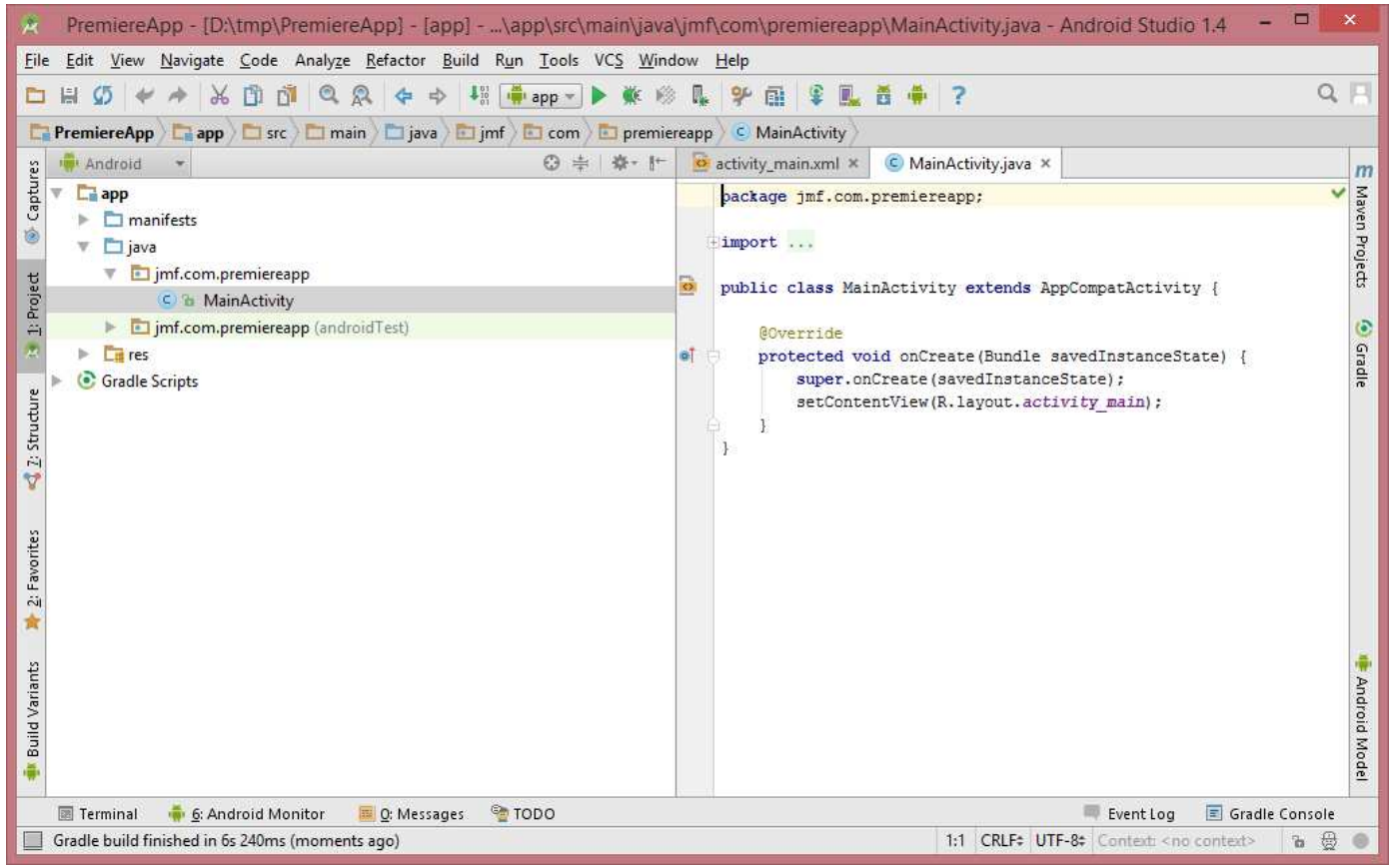
Développement du projet Hello World (4/6)

- Dans l'écran suivant, garder les valeurs pour l'activity et son fichier de configuration d'IHM
- `MainActivity` sera le nom de la première classe chargée (une activité), instanciée et sur laquelle est lancée la première méthode
- `activity_main` est le nom de base du fichier xml qui décrit l'IHM de cette activité
- Cliquer Finish



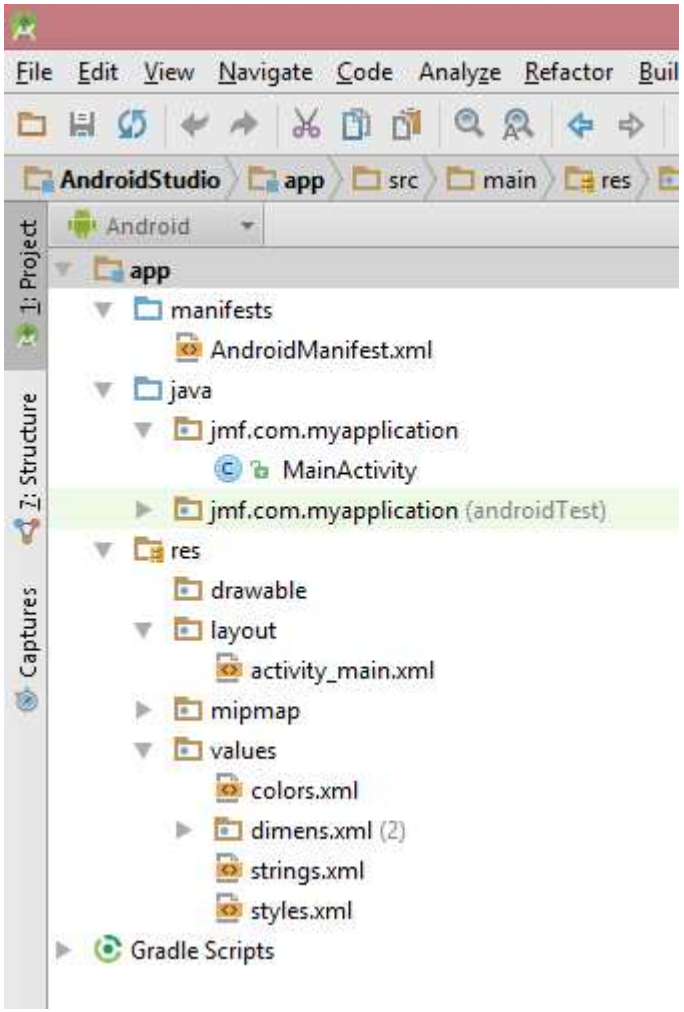
Développement du projet Hello World (5/6)

- L'environnement Android Studio a construit l'app :



Développement du projet Hello World (6/6)

- L'environnement Android Studio et l'Android SDK ont créé plusieurs répertoires et fichiers :
- Sous `manifests` le fichier `AndroidManifest.xml` de configuration de l'application
- Sous `java` le package (`jmf.com.myapplication`) avec les sources de l'application (`MainActivity`)
- Le répertoire `res` (pour ... ressources) contenant, entre autre, `activity_main.xml` (dans le sous répertoire `layout`) et `strings.xml` (dans `values`)



L'activité générée

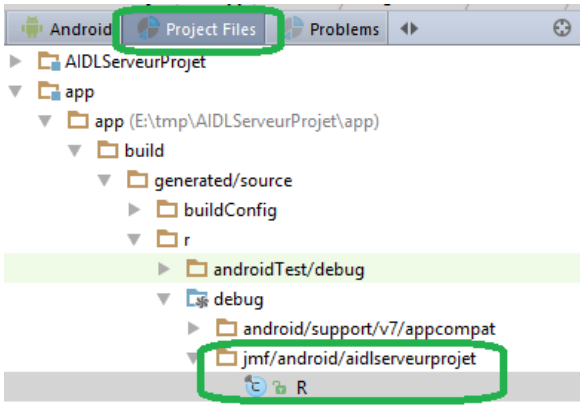
- Une `AppCompatActivity` est proposée : c'est une `Activity` qui permet de créer facilement une `ActionBar` (~ barre de menus)

même pour des appareils Android de version ancienne à l'aide de la bibliothèque de compatibilité v7 (Android 2.1) : "These libraries offer backward-compatible versions of new features, provide useful UI elements that are not included in the framework"

- `Bundle` est utile lorsque une activité est détruite puis recréée (voir cycle de vie d'une activité)

Le `R.java` généré (1/2)

- On utilise le `R.java` ! `R.java` est généré par l'environnement (parfois à la première compilation)
- Android Studio et `R.java` voir à `http://stackoverflow.com/questions/18525374/android-studio-r-java`
- Bref le `R.java` se trouve dans le répertoire *`RepProjet`*`/app/build/generated/source/r/debug/`*`paquetage DuModule.nomDuModule`*
- On peut aussi le lire par l'onglet Project Files en suivant cet arborescence :



Le `R.java` généré (2/2)

- Le fichier est énorme (6300 lignes) mais il contient entre autre des constantes dont :
  - la constante `activity_main` (utile dans `R.layout.activity_main`)
  - `app_name` (`R.string.app_name`)

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package jmf.com.myapplication;
public final class R {
    public static final class layout {
           public static final int activity_main=0x7f040019;
           ...
    }
     public static final class string {
        public static final int app_name=0x7f060014;
     ...
    }
...
}
```

Remarques sur le `R.java` (1/4)

- Il y a correspondance entre les noms dans le fichier `R.java` et les noms utilisés dans le programme Android
- Les identificateurs dans `R.java` font référence à des fichiers se trouvent dans le répertoire `res`. Par exemple `R.layout.activity_main` indique le fichier `activity_main.xml` se trouvant dans le répertoire `layout` (sous répertoire de `res`)
- Comme `activity_main.xml` décrit une interface graphique, on affecte cette IHM à une activité par `setContentView(R.layout.activity_main);`

Remarques sur le `R.java` (2/4)

- Les images sont accessibles par `R.drawable.`*`nomImage`* et corrrespondent à des fichiers images dans le répertoire `res/drawable`. Le fichier `R.java` permet d'associer un `int` à ce nom (de fichier)

R.java

```
 9
10 public final class R {
11     public static final class attr {
12     }
13     public static final class drawable {
14         public static final int apropos=0x7f020000;
```

Remarques sur le `R.java` (3/4)

- Les noms sont accessibles par `R.string.`*nom* et correspondent à des chaînes de caractères dans le fichier `strings.xml` (avec un `s` !) dans le répertoire `res/values`. On récupère ces noms dans le code source par `getResources().getString(R.string.`*nom*`);`
- Le fichier `R.java` permet d'associer un `int` à ce nom (bis)

```
public static final class string {
    public static final int aboutUs=0x7f040004;
    public static final int app_name=0x7f040001;
    public static final int aroundMe=0x7f040003;
    public static final int gpsSettings=0x7f040008;
    public static final int hello=0x7f040000;
    public static final int loadingmessage=0x7f040007;
    public static final int mapkey=0x7f040009;
    public static final int searchStation=0x7f040006;
    public static final int seeAllStations=0x7f040002;
```

strings.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="hello">Hello World, VelibProjetActivity!</string>
4     <string name="app_name">VelibProjet</string>
5     <string name="seeAllStations">Voir les stations</string>
```

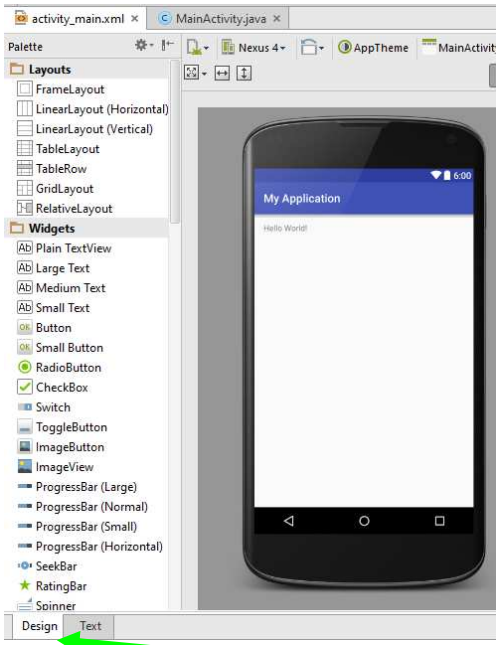
Remarques sur le `R.java` (4/4)

- Les composants graphiques ont un identifiant dans le fichier xml qui les contient (par exemple le `activity_main.xml`). Cet identifiant est la valeur de l'attribut `android:id` et est de la forme "`@+id/leId`". On récupère, dans le code, le composant graphique grâce à cet identifiant par l'appel `findViewById(R.id.leId);`
- Par exemple

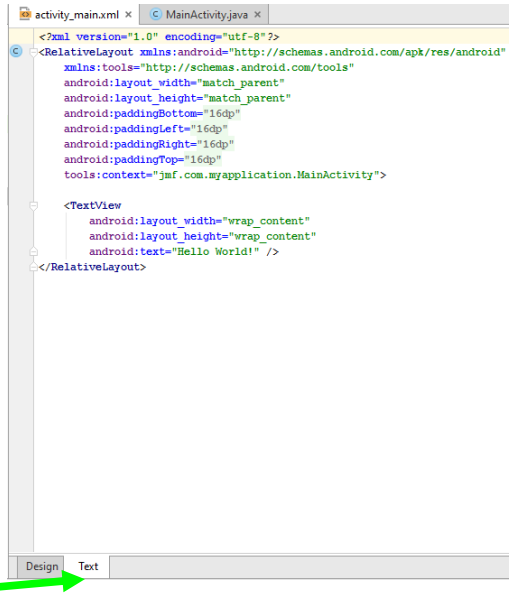
```
Button leBouton =
(Button)findViewById(R.id.leId);
```

Le `activity_main.xml` généré

- = fichier répéré par la constante `activity_main` (`R.layout.activity_main`), argument de `setContentView()` dans l'activité
- Son contenu peut être visualisé par :



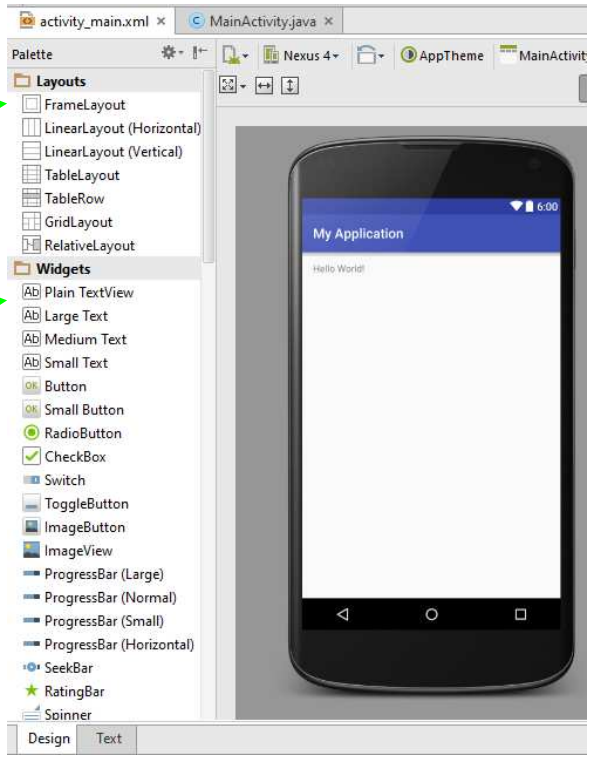
ou par :



suivant l'onglet choisi

activity_main.xml = ?

- Il contient la description de l'IHM
- Souvent, l'IHM est construite par glisser-déposer (onglet Graphical Layout) proposant :
  - les positionnements (Layouts)
  - les composants graphiques (Widgets)



`activity_main.xml` **(suite)**

- = un `RelativeLayout` a été utilisé contenant un `TextView`
- Le texte affiché par le `TextView` (= une zone de texte ~ `Label` de AWT) est la chaîne "Hello World"
- Il est nettement préférable (cf. internationalisation) de mettre une entrée du fichier `strings.xml` en écrivant `android:text="@string/hello_world"`

activity_main.xml × MainActivity.java ×

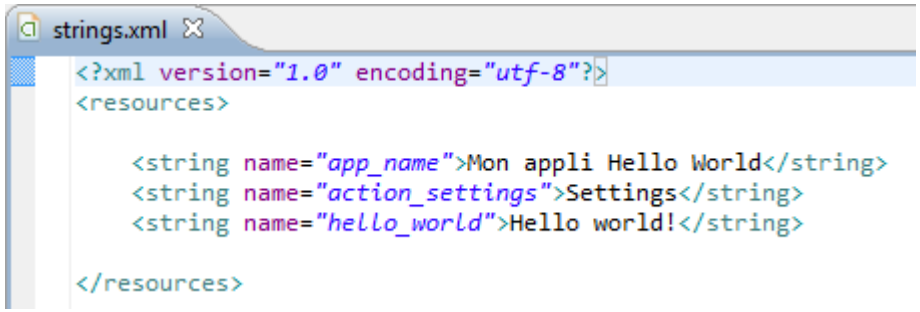
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="jmf.com.myapplication.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

</RelativeLayout>
```

strings.xml

- On aura par exemple :



- C'est un fichier de correspondance (identificateur, valeur)
- Par exemple, si l'élément `app_name` de ce fichier `strings.xml` a pour corps
  `Mon appli Hello World`
  alors l'identificateur `app_name` correspond à la valeur (chaine de caractères) `Mon appli Hello World`

Accès aux `Strings`

- Une chaîne de caractères mise dans le fichier `strings.xml` par
  `<string name="hello">Coucou</string>`
  est accessible par :
- `@string/hello` dans d'autres fichiers xml
- `R.string.hello` dans le code Java

Le manifeste

- C'est le fichier `AndroidManifest.xml` qui donne des indications sur l'application (paquetage initial, droits nécessaires demandés par l'application, ...)

activity_main.xml × | strings.xml × | AndroidManifest.xml × | MainActivity.java ×

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Si problème avec le `R.java`

- Envisager les traitements suivants :
  - 1°) Sauvegarder tous les fichiers
  - 2°) faire Build | Clean Project
  - 3°) Vérifier les fichiers XML sous `res`. Entre autre des fichiers ont pu être malenconteusement déplacés (sous `layout`, sous `values`, ...). Vérifier la syntaxe de ces fichiers XML
  - 4°) Enlever `import android.R;` dans le code
  - 5°) Quitter Android Studio et relancer le
  - 6°) Finalement ne pas se soucier de l'erreur. Lancer l'application, ça marche (si, si)

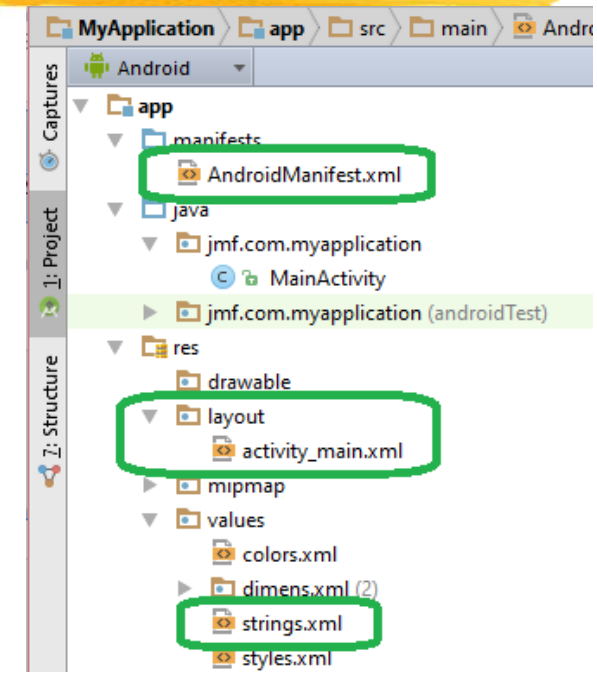
Si problème avec le chemin trop long (sous Windows)

- En cas de message d'erreur dans Android Studio de la forme : `Error: File path too long on Windows, keep below 240 characters : …` ajouter : `buildDir = "`*`unCheminCourt`*`"` comme fils de `allprojects` dans le `fichier build.gradle (Project: XXX)`
- Par exemple :

- biblio : `http://stackoverflow.com/questions/33905687/error-file-path-too-long-on-windows-keep-below-240-characters`

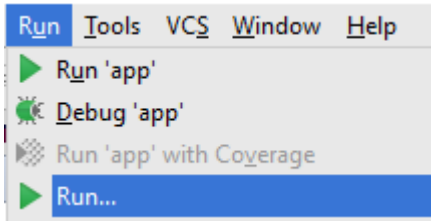
Résumé : les 3 principaux fichiers XML

- Les 3 (types de) fichiers xml :
  - les fichiers sous `layout`,
  - `values\strings.xml`,
  - `AndroidManifest.xml`
- Les fichiers sous `layout` sont les fichiers xml décrivant des (parties d') IHM
- Le fichier `values\strings.xml` contient des valeurs de chaînes de caractères
- Le fichier `AndroidManifest.xml` contient la configuration de l'application Android (sa description, ses demandes de permission, etc.)



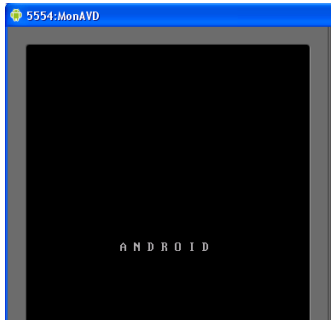
L'exécution (1/3)

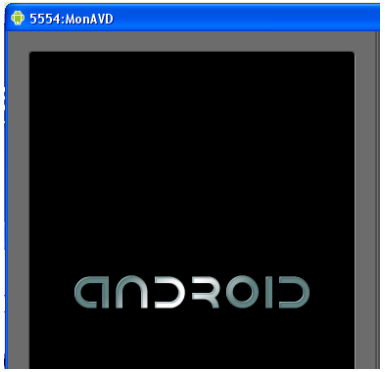
- Sélectionner Run | Run...



- Ou l'icône

L'exécution (2/3)

- La suite des écrans 

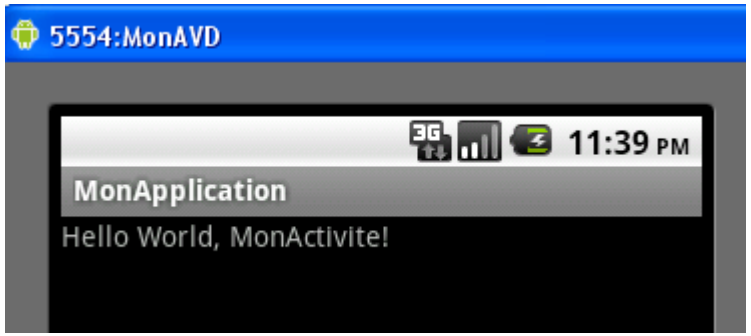
puis 

finit par amener :

- Déplacer alors la glissière (c'est un smartphone !)

L'exécution (3/3)

- Apparaît alors votre application Android :



Application Android

- Pour obtenir cet "Hello World", nous avons eu besoin :
  - de fichiers XML de configuration (`AndroidManifest.xml`),
  - de définitions de constantes (`strings.xml`),
  - d'interfaces graphiques (`activity_main.xml`)
  - en plus de classes Java (presque) créées par le programmeur (`MonActivite.java`) ou par l'environnement (`R.java`)
- Tout cet ensemble constitue une application Android (Android app)
- Pour d'autres applications plus importantes, on aura besoin d'autres "ressources"

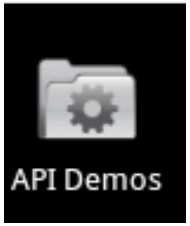
Des exemples d'applications Android (1/2)

- L'environnement Android apporte de multiples exemples d'applications
- Pour les exécuter revenir au l'écran "home"
  Au besoin cliquez sur :

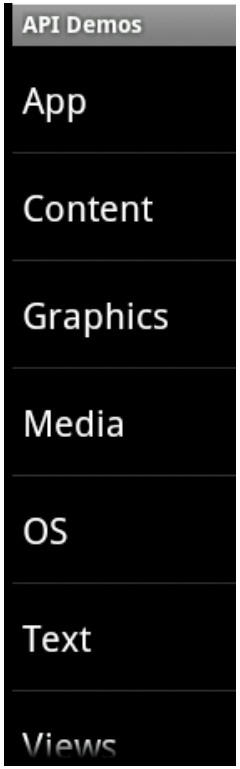
- Cliquez ensuite sur la grille des applications :

Des exemples d'applications Android (2/2)

- Sélectionner les "API Demos"



- Des applications sont accessibles, rangées par thèmes :



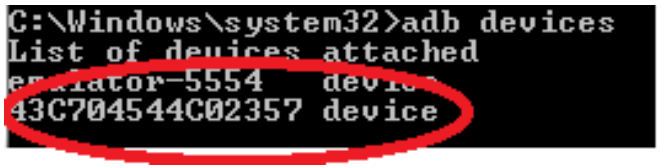
- Le code de ces applications est accessible à partir de : `%repInstalAndroidSDK%\samples\android-XXX\ApiDemos`

Pour exécuter sur un "vrai" appareil Android (1/3)

- Voir à `https://developer.android.com/tools/index.html`
- Il faut avoir cocher la case "débugage USB" sur l'appareil accessible par le menu "Options pour les développeurs"
- L'appareil étant non connecté,
  - pour une tablette Samsug Galaxy Tab, faire Applications | Paramètres | Applications | Développement | Débogage USB,
  - pour un téléphone Samsug Nexus S, faire Paramètres | {} Options pour les développeurs | Débogage USB,
  - Activer le débugage USB (la case doit être cochée)
- Le gag dans Android 4.2 et suivant. "Options pour les développeurs" est masqué par défaut. Pour rendre cet item visible faire Paramètres | A propos du téléphone et taper 7 fois (au moins) sur Numéro de build
- En général cette étape est faite une seule fois

Pour exécuter sur un "vrai" appareil Android (2/3)

- Connecter le cable USB au PC
- Vérifier dans une fenêtre de commandes, taper
  `adb devices`
  doit afficher la tablette et retourner quelque chose comme :

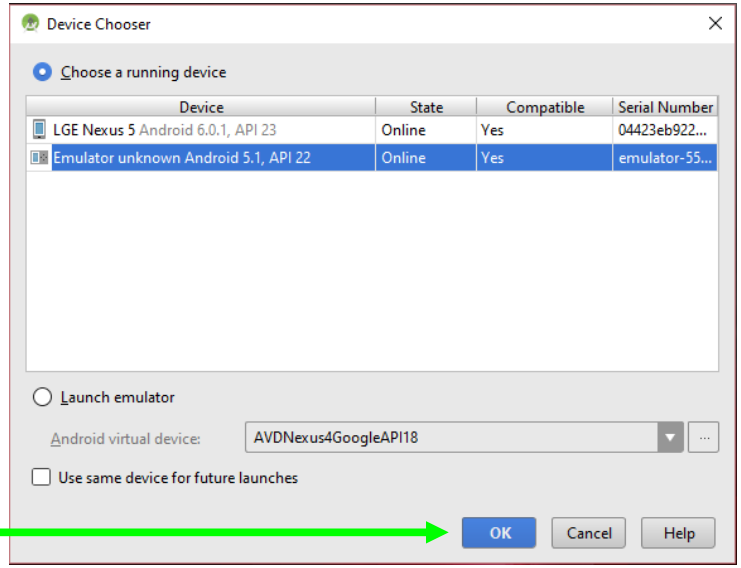


- Remarque : la commande `adb` se trouve dans
  `%RepInstallSDK%\platform-tools`
- Si vous avez une réponse comme :

  autoriser, sur le smartphone,
  qu'il accepte (toujours) les connexions provenant de l'ordinateur

Pour exécuter sur un "vrai" appareil Android (3/3)

- Au moment du lancement du programme, Android Studio présente les diverses machines physiques ou AVD permettant d'éxécuter du code Android
- Faites votre choix (et donc ici une machine physique)



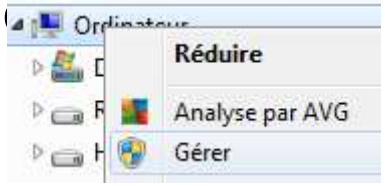
- et cliquer OK

Charger un driver USB de smartphone (1/3)

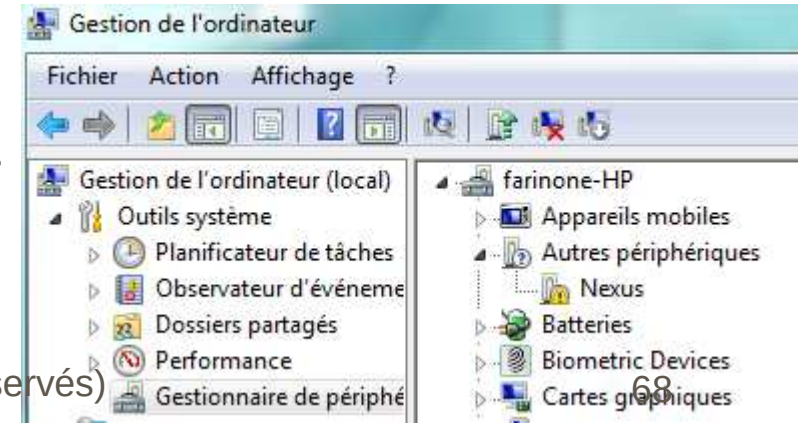
- En général, si la case Débogage USB est cochée, lorsque le cable USB entre le smartphone et l'ordinateur est mis, tout devrait bien se passer, l'ordinateur et Android Studio finissent par trouver le smartphone
- Si ce n'est pas le cas, c'est que votre PC n'a pas le driver de votre smartphone
- Sous Linux, tout devrait bien se passer (communication Linux-Linux)
- Pour windows, voir comment obtenir les drivers à partir de `http://developer.android.com/tools/extras/oem-usb.html#Drivers`
- source : `http://developer.android.com/sdk/win-usb.html`

Charger un driver USB de smartphone (2/3)

- Si vous êtes sous windows, voir ensuite à l'URL `http://developer.android.com/sdk/oem-usb.html#InstallingDriver`
- Les principales étapes sont :
  - Connecter le smartphone à l'ordinateur
  - Sélectionner l'icône de l'ordinateur, cliquer droit, et sélecti

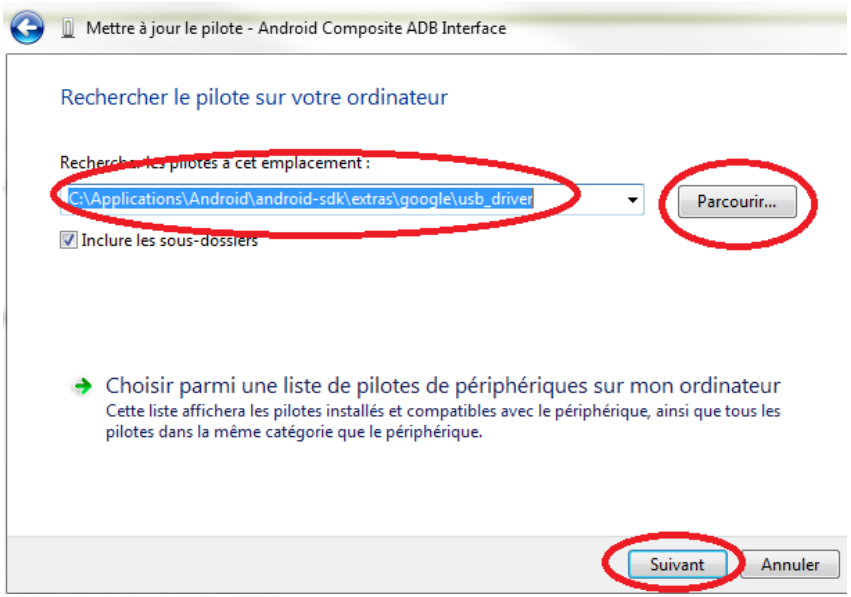


  - Sélectionner Gestionnaire de périphériques dans le panneau gauche, puis Autres périphériques dans le panneau droit



Charger un driver USB de smartphone (3/3)

- Cliquer droit sur l'icône du smartphone puis l'item "Mettre à jour le pilote"
- Dans la nouvelle fenêtre cliquer sur "Rechercher un pilote sur mon ordinateur"
- Aller le chercher dans `%RepInstallAndroid%\extras\google\usb_driver\`
- Cliquer le bouton Suivant



Pb Device unknown

- Parfois un smartphone est visible dans la fenêtre Android Device Chooser mais avec une icône  suivi éventuellement de unknown dans la colonne Target
- Accepter que le smartphone veuille bien communiquer avec l'ordinateur (écran "Autoriser le débogage USB", Empreinte numérique de la clé RSA de l'ordinateur …)
- Ou bien, débrancher et rebrancher le cable USB
- Ou bien, redémarrer alors le téléphone. Voire l'ordinateur
- Bibliographie : `http://stackoverflow.com/questions/10731375/eclipse-target-unknown-in-android-device-chooser`

Pour désinstaller une application

- Sélectionner l'application (en laissant le doigt sur son icône) puis la glisser sur Uninstall. Confirmer la désinstallation (2 fois OK !)

Commandes `adb` utiles

- `adb devices` donne la liste des émulateurs et smartphones connectés à l'environnement de développement

```
C:\Users\Jean-marc>adb devices
List of devices attached
emulator-5554   device
emulator-5556   device
04423eb9220892b0        device
```

- En cas de problèmes, `adb kill-server` suivi de `adb start-server` est souvent utile
- `adb install` *nomApk* permet d'installer une application Android (il ne faut alors qu'un seul émulateur ou smartphone connecté). Par la suite, il faut autoriser les applications qui ne viennent pas du Play store de s'installer
- Et beaucoup d'autres, voir à `http://developer.android.com/tools/help/adb.html`

Signature d'une application

- Toute application Android doit être signée pour pouvoir être déployée
- Pendant le développement la clé utilisée est dite clé de debug
- Si on veut distribuer l'application il faut construire une clé de release (et la garder dans un keystore = l'entrepôt de la clé)
- Pour créer une clé de release, il faut Build | Generate Signed APK…
- Suivre ensuite les indications
- source : `http://developer.android.com/tools/publishing/app-signing.html#studio`

Les permissions

- Par défaut une application Android ne peut pas faire de communication Internet, de géolocalisation, enregistrer, lire les données de certaines autres applications, ...
- C'est dommage ? Non c'est normal
- Pour le permettre, il faut indiquer vouloir ces permissions. A l'installation de l'application, il est demandé à l'utilisateur l'autorisation d'utiliser ces permissions (jusqu'à l'API 22)
- On peut le faire statiquement dans le fichier manifest
- Par exemple :

```
<manifest ...>
     <application>
     ...
     </application>
     <uses-permission android:name="android.permission.INTERNET"/>
     <uses-permission android:name="android.permission.RECORD_AUDIO"/>
     <uses-permission android:name="android.permission.READ_CALENDAR"/>
     ...
</manifest>
```

Les permissions depuis Marshmallow

- Depuis l'API 23 (Android 6.0), les permissions sont gérées à l'exécution et pas (seulement) à l'installation de l'app
- On peut avoir beaucoup plus de souplesse. Exemple pouvoir utiliser la caméra sauf dans certaines zones terrestres (militaires, ...)
- On peut toujours déclarer ces permissions dans le manifeste mais elles sont aussi gérées à l'exécution
- Par exemple si l'app veut écrire dans l'application calendrier, l'app doit avoir le code :

```
int permissionCheck = ContextCompat.checkSelfPermission(thisActivity,
        Manifest.permission.WRITE_CALENDAR);
```

- Si l'app a la permission (dans son manifeste), cette méthode retourne `PackageManager.PERMISSION_GRANTED`. Si cette permission n'est pas dans le manifeste `PERMISSION_DENIED` est retourné et l'app demande explicitement à l'utilisateur cette permission
- Bibliographie : `https://developer.android.com/training/permissions/requesting.html`

Résumé du chapitre 2

- Android utilise une architecture de composants : c'est l'environnement d'exécution qui gère ces composants (leur cycle de vie, le lancement de méthodes spécifiques du composant sous certaines situations, ...). Les composants fondamentaux d'une application Android sont les `Activitys`, les `Services`, les `ContentProviders`, les `BroadcastReceivers`
- Les composants communiquent entre eux grâce aux `Intents`. Ceux ci peuvent être utilisés pour demander à l'environnement d'exécution de faire une certaine tâche : ce sont des intentions
- Développer une application Android nécessite de créer plusieurs fichiers de code (`.dex` compilés de `.java`) de description (`.xml`, ...). L'ensemble est mis dans un `.apk`
- Le fichier de description d'une application Android est son manifeste `AndroidManifest.xml`

Chapitre 3

Les interfaces utilisateurs avec Android

Plan du chapitre 3

- IHM des smartphones, IHM pour Android
- Les deux principes des IHM
- Un second programme : IHM par programmation, par description
- Les `Layout`, les contrôles
- La gestion des événements
- Enchaîner les écrans
- `Toast` et traces
- L'internationalisation
- Les icônes, menus, notifications et boîtes de dialogues
- Le multi-plateformes

Smartphone != ordinateur

- Android tire partie des particularités des smartphones :
  - interface homme machine adapté (tactile, multitouch)
  - divers modes : vibreur, sonnerie, silencieux, alarme
  - notifications (d'applications, d'emails, de SMS, d'appels en instance)
  - de boussole, accéléromètre, GPS
  - divers capteurs (gyroscope, gravité, baromètre)
  - NFC, RFID
  - téléphonie (GSM) et réseau EDGE, 3G, 4G, SMS, MMS
  - Appareil photo, caméra vidéo (enregistrement et rendu)
  - une base de données intégrée (SQLite)
- En plus de ce qu'on peut avoir sur un ordinateur : navigateur, bibliothèques graphiques 2D, 3D (Open GL), applications de rendu multimédia (audio, vidéo, image) de divers formats, réseau Bluetooth et Wi-Fi, caméra

Les IHM graphiques avec Android

- Bibliothèque propre
- Pas AWT, ni Swing, ni Java ME / LCDUI
- Décrit par fichier XML
- Ecran en Android géré par une activité

Activité (Activity)

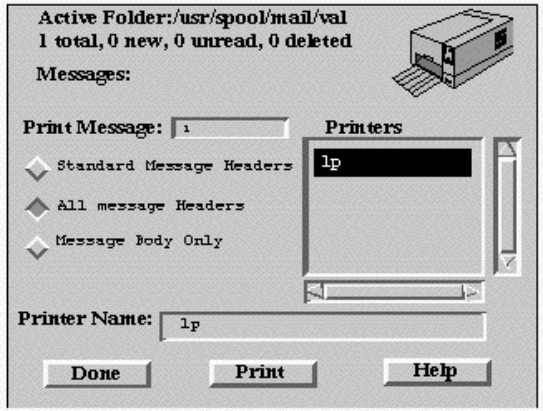
- Correspond à une seule classe Java
- Une activité gère l'affichage et l'interaction d'un écran (IHM)
- Gère les événements, lance l'affichage d'autres écrans, lance du code applicatif
- Suit un cycle de vie déterminé (cf. applets, midlets, servlets, EJB, ...)
- Utilise les `Intent` pour lancer d'autres activités

Construction d'une IHM

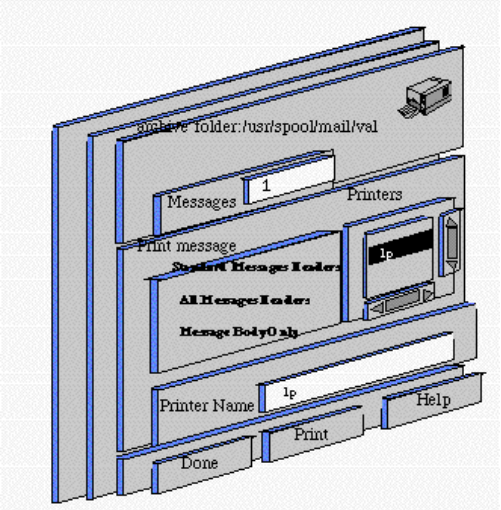
- Plutôt en XML mais
- XML ne peut pas être débogué !
- Tout ne peut pas être fait en XML

Premier principe des IHM (1/4)

- Quand on voit ceci :

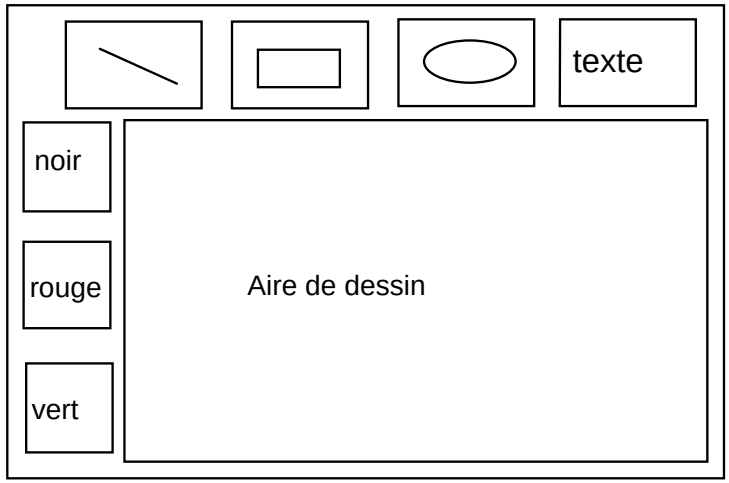


- C'est qu'on a programmé cela :

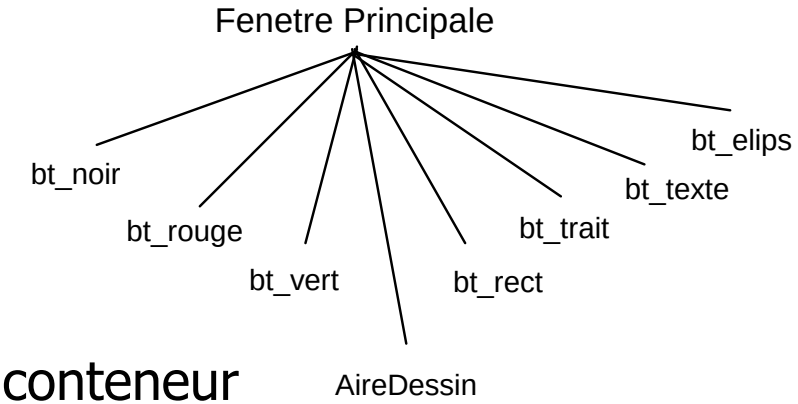


Premier principe des IHM (2/4)

- Si on veut :



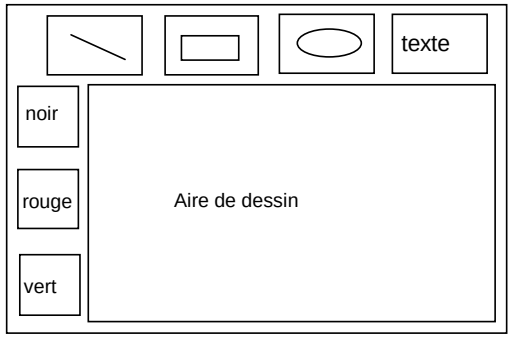
- On doit construire cela :



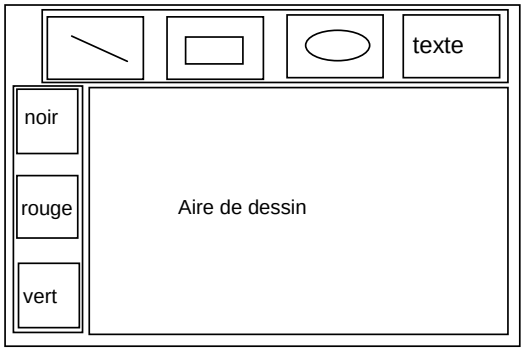
- => Fenetre principale = un conteneur

Premier principe des IHM (3/4)

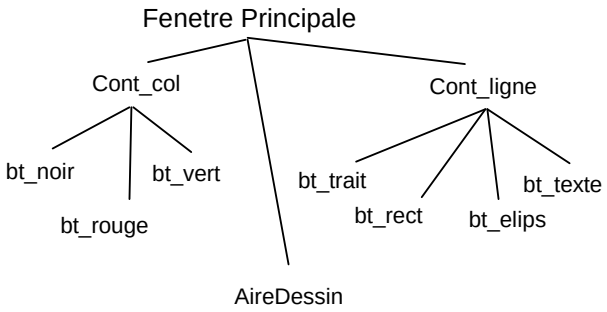
- Plus sûrement, si on veut :



on écrit plutôt :



c'est à dire :



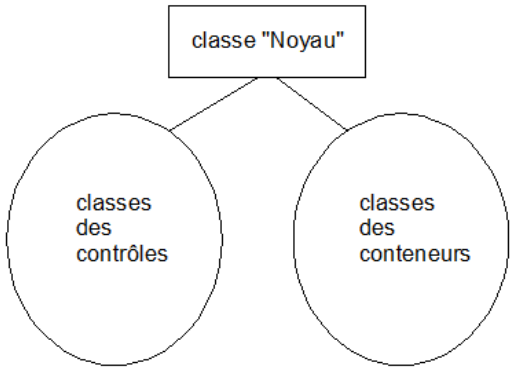
Premier principe des IHM (4/4)

- (Premier principe) : construire une IHM, c'est mettre des composants graphiques les uns à l'intérieur des autres
- Il y a donc, dans une IHM à présenter à l'utilisateur, un arbre de composants graphiques
- Les éléments de cet arbre sont des composants graphiques (redite !)
- Etre "fils de" dans cet arbre signifie "être contenu dans"

- Voilà pour les composants graphiques ! (et le premier principe des IHM)

Second principe des IHM (1/3)

- Les ensembles de composants graphiques sont des classes. On aura la classe des boutons, la classe des cases à cocher, etc.
- Un composant graphique particulier sera une instance particulière d'une classe. Par exemple le bouton "Quitter" et le bouton "Sauvegarder" d'une IHM seront deux instances de la classe des boutons : merci l'OO !
- Il y a une famille de conteneurs et une famille de non conteneurs
- D'où les classes de composants graphiques :

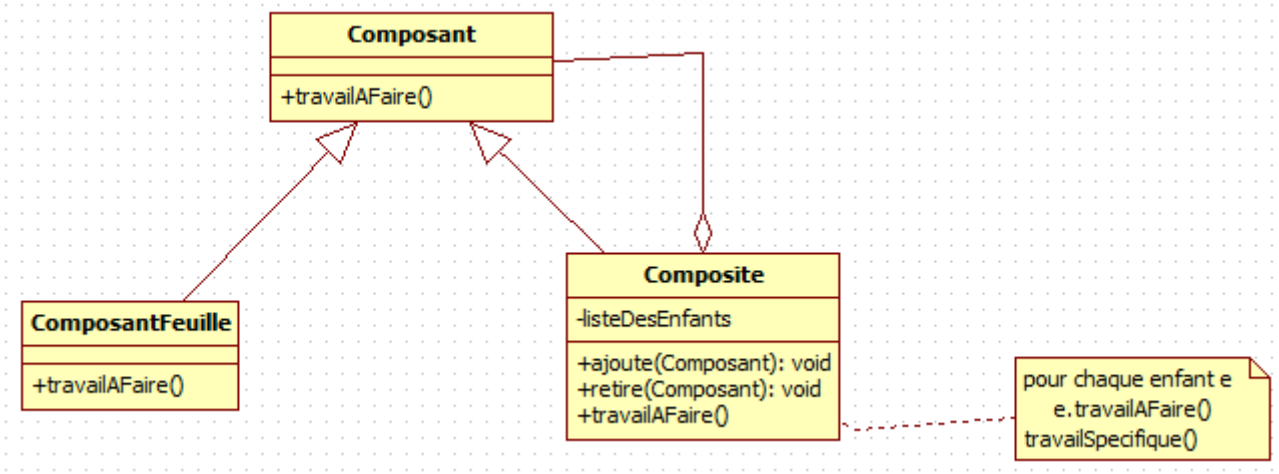


- Question : Comment sont rangées ces classes ?

- Réponse : dans un arbre d'héritage de classes : merci l'OO (bis) !

Second principe des IHM (2/3)

- Plus précisément le point de départ de l'arborescence des classes est :



- C'est le design pattern Composite
- remarque : `travailAFaire()` est répercuté sur tout l'arbre des instances

Second principe des IHM (3/3)

- (Second principe) : les bibliothèques pour construire des IHM sont, en Java, (et souvent !) des classes rangées par arborescence d'héritage
- Les éléments de cette arborescence sont des classes (redite !)
- Etre "fils de" dans cette arborescence signifie "hérite de"
- Voilà pour les classes (et le second principe des IHM)

Les deux principes des IHM

- Lorsqu'on parle d'IHM, il existe deux arborescences et donc deux "principes"
- 1er principe : Une IHM est construite en mettant des composants graphiques les uns à l'intérieur des autres
- 2ième principe : les composants graphiques sont obtenus comme instances de classes. Ces classes sont rangées dans une arborescence d'héritage
- Remarque : Ces deux arbres (celui des composants graphiques et celui des classes) ont peu de choses à voir l'un l'autre
  - Le premier est l'architecture de l'interface i.e. le placement des divers composants graphiques les uns par rapport aux autres,
  - le second est un arbre d'héritage de classes donné une bonne fois par le concepteur de la bibliothèque graphique

Résumé du chapitre 3

- Les interfaces humain machine (IHM) sont plus riches sur les smartphones que sur les ordinateurs. Elles nécessitent des API particulière : c'est le cas pour Android
- La programmation des IHM suit deux principes, l'un sur les composants graphiques, l'autre sur les classes

Fin