Chapitre 1

Découvrir la plateforme Android

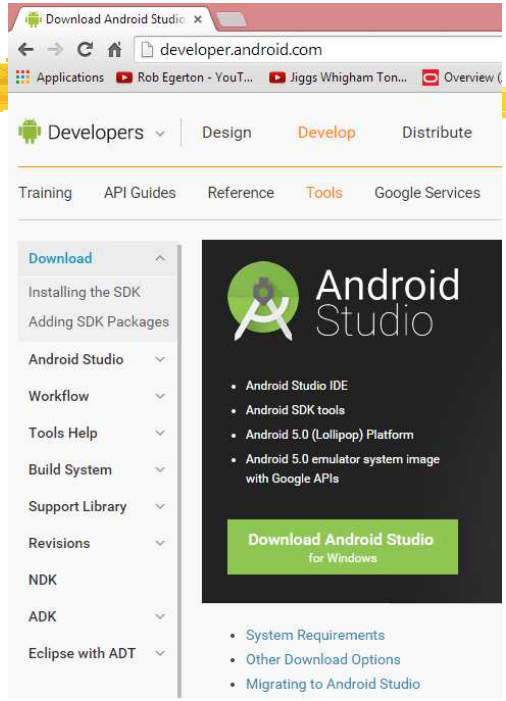
Plan du chapitre 1

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

Android =

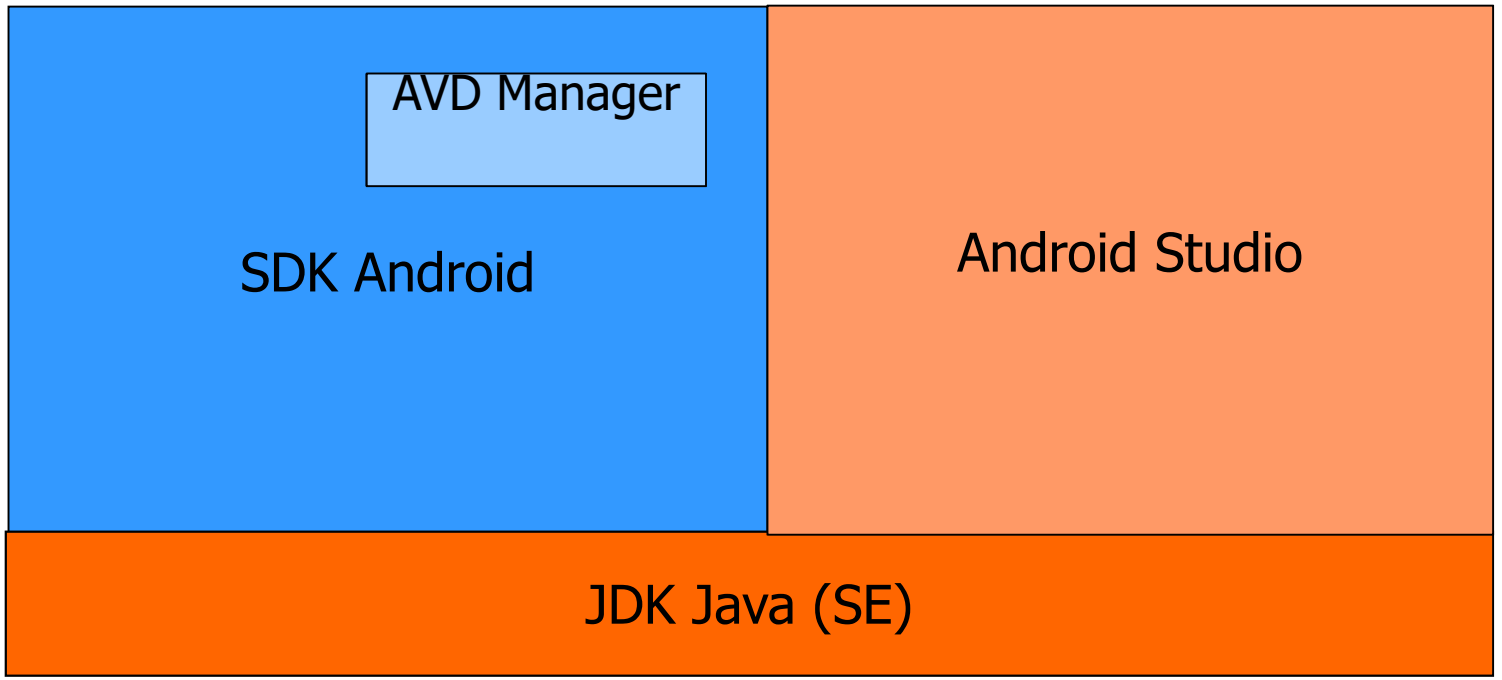
- Android = un système d'exploitation open source pour smartphones, PDA, tablettes : systèmes légers
- = une startup rachetée en août 2005 par Google
- 5 novembre 2007, création de l'OHA (Open Handset Alliance) = un consortium créé à l'initiative de Google réunissant des entreprises opérateurs mobiles, 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

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`

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 !

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

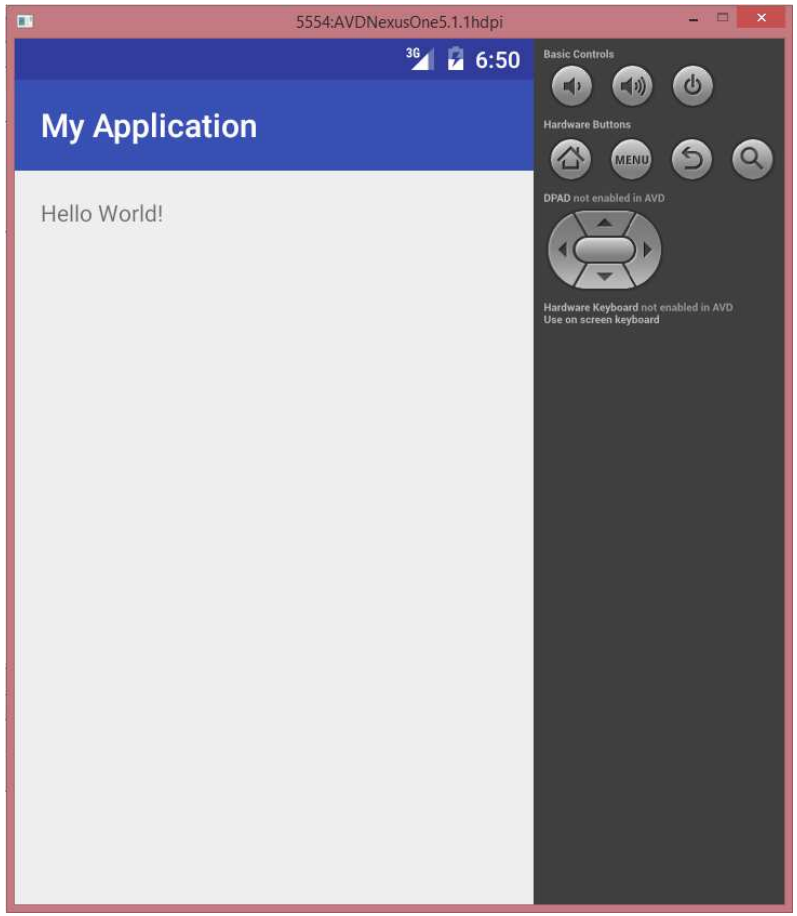
Chapitre 2

Développement Android

Un premier projet : Hello World

Hello World en Android

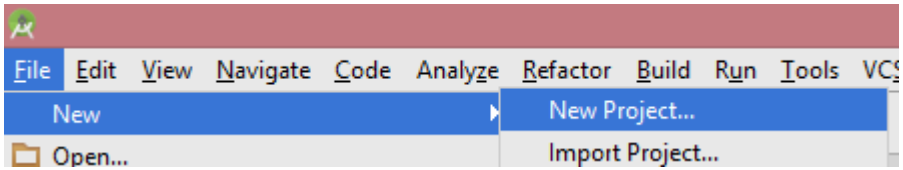
- On veut obtenir :



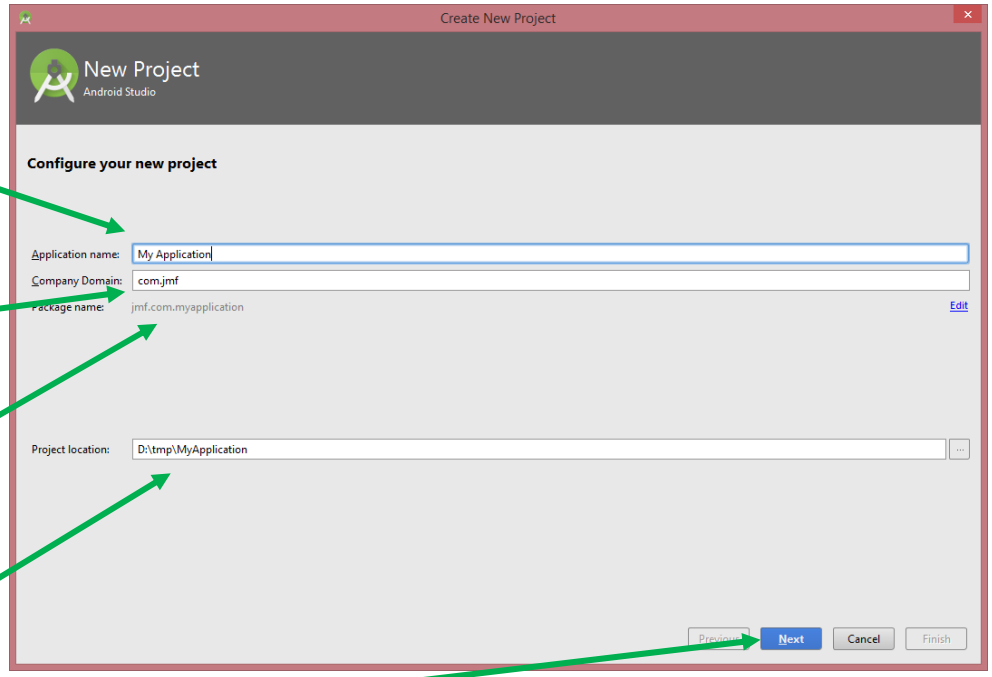
- <u>Remarque :</u>
  Suivant les versions des environnements que vous avez, vous pouvez avoir des fenêtres légèrement différentes des captures d'écran des diapos suivantes

Développement du projet Hello World (1/6)

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



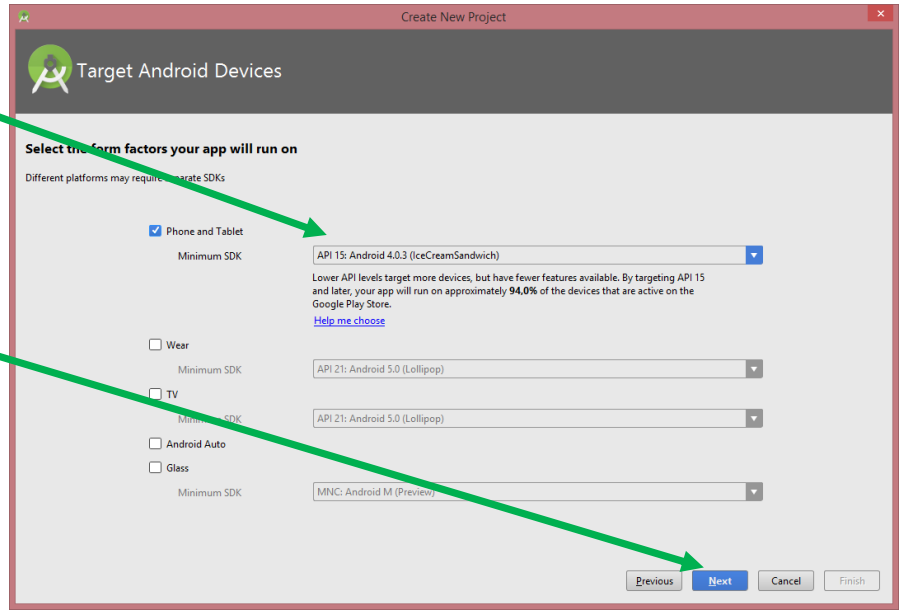
- On obtient la fenêtre :



- 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

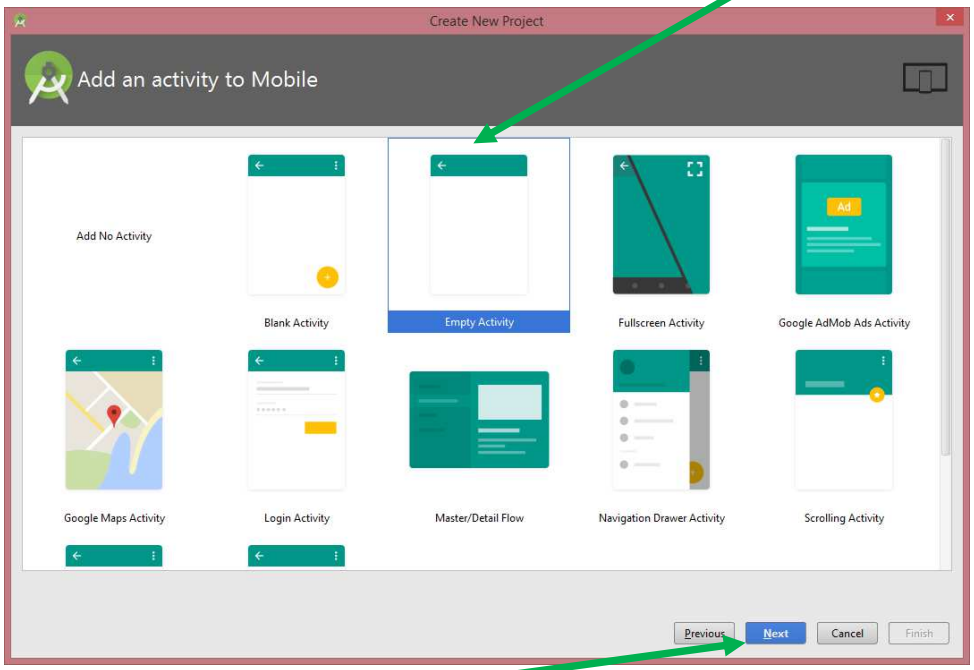
Développement du projet Hello World (2/6)

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



Développement du projet Hello World (3/6)

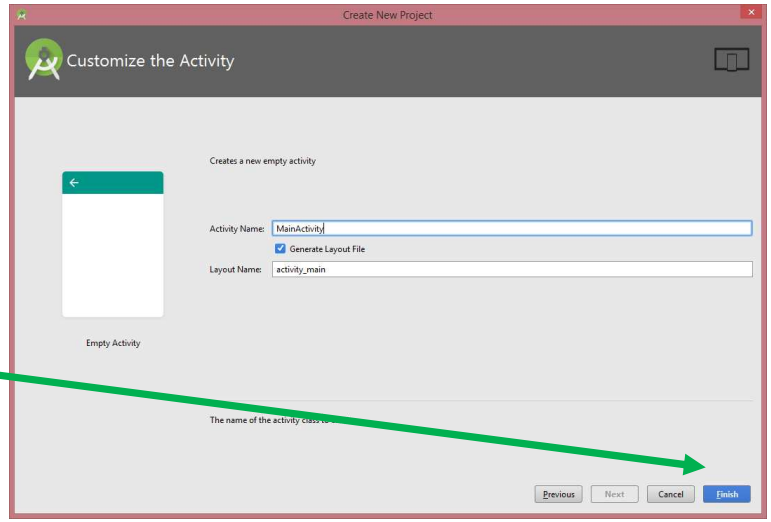
- Dans l'écran suivant, choisir Empty Activity



- Cliquez Next

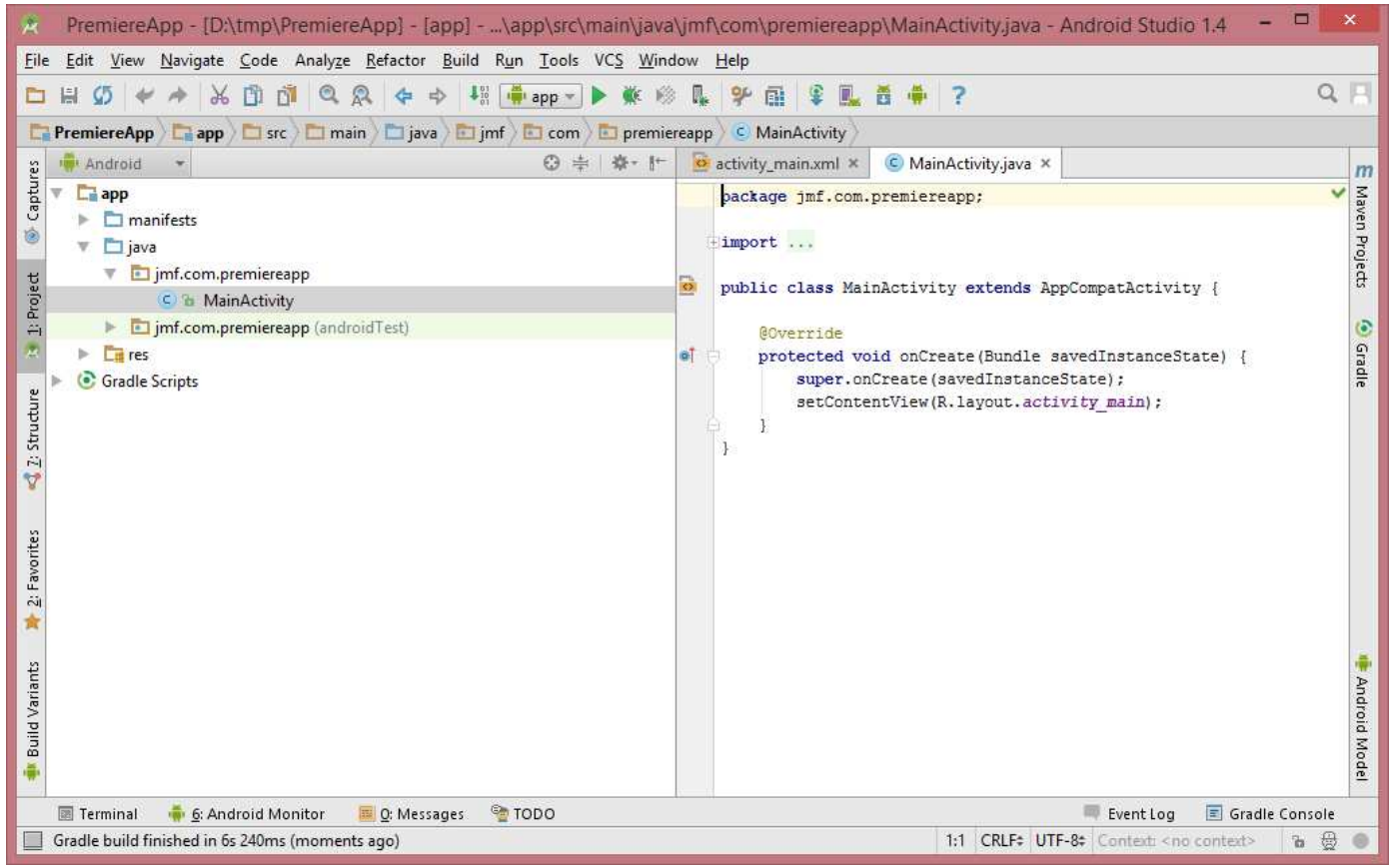
Développement du projet Hello World (4/6)

- 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



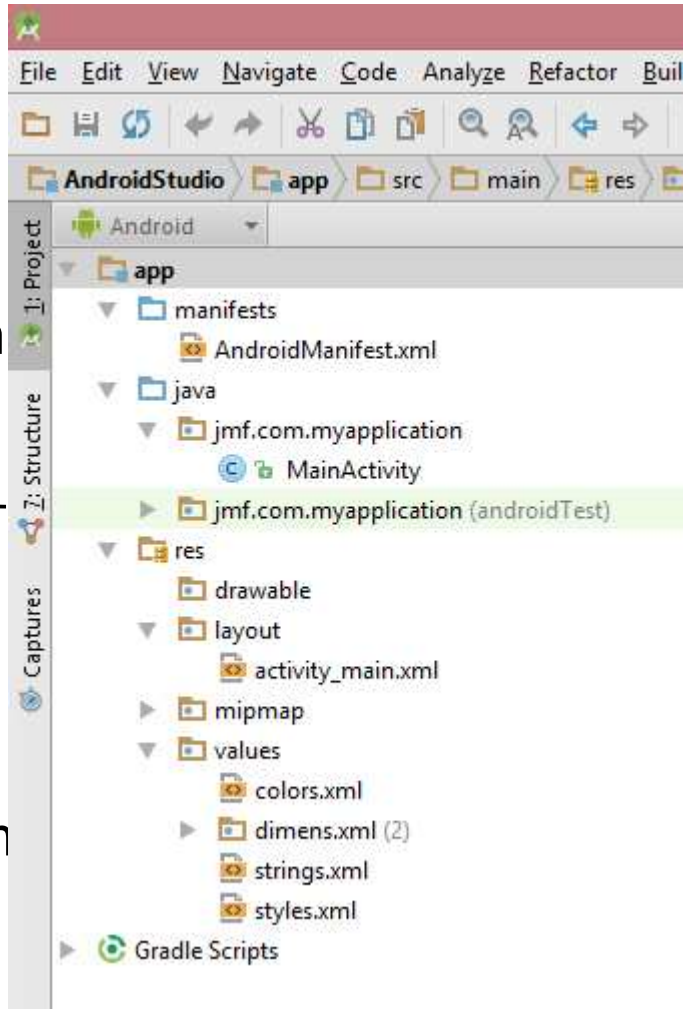
Développement du projet Hello World (5/6)

- 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.myappl` sources de l'application (`MainActivity`)
- Le répertoire `res` (pour ... ressources) contenant, entre autre, `activity_main.xml` (dans le sous répertoire `layout`) et `strings.xml` (dan `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/`*`paquetageDuModule.nomDuModule`*

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>
```

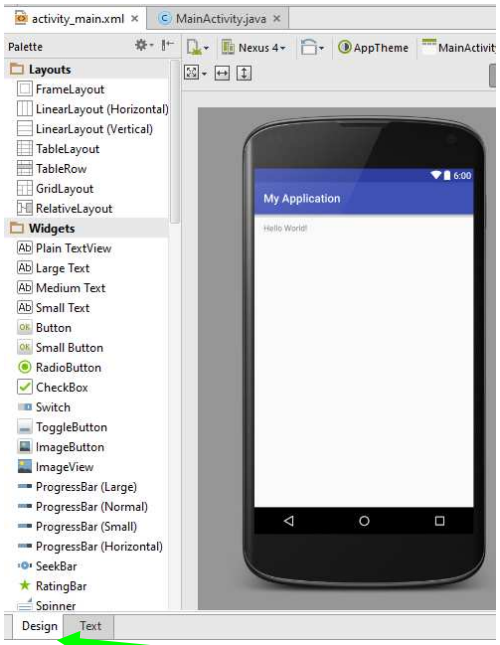
Remarques sur le `R.java` (4/4)

- 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

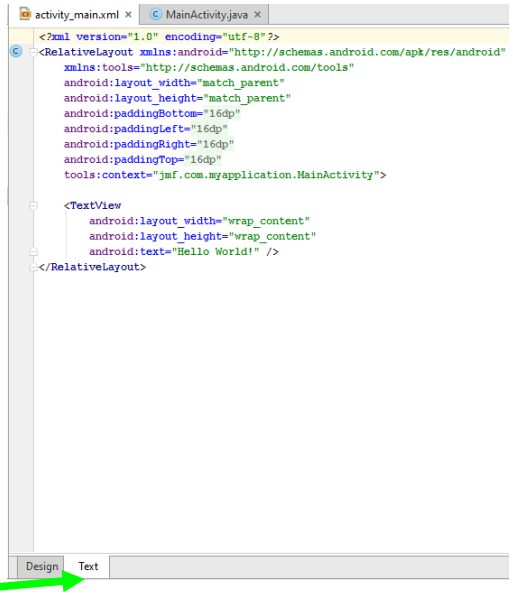
```
Button leBouton =
(Button)findViewById(R.id.leId);
```

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 :



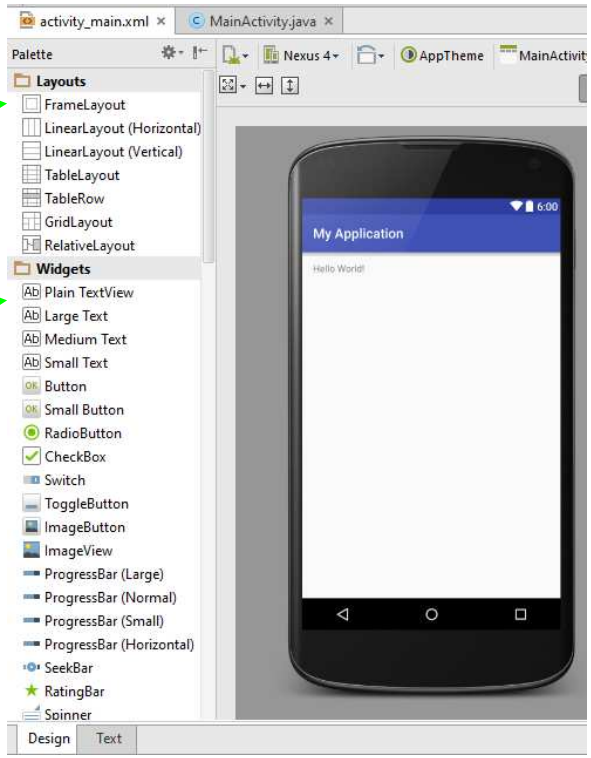
ou par :



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)**

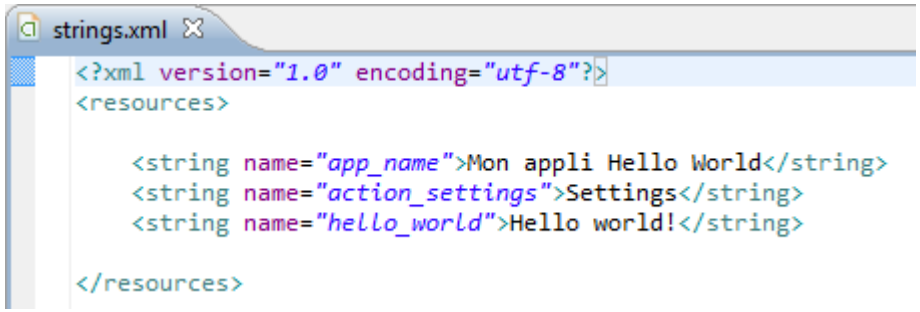
- = un `RelativeLayout` a été utilisé contenant un `TextView`
- Le texte affiché par le `TextView` (= une zone de texte ~ `Label` de AWT) est la chaîne "Hello World"
- Il est nettement préférable (cf. internationalisation) de mettre une entrée du fichier `strings.xml` en écrivant `android:text="@string/hello_world"`

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>
```

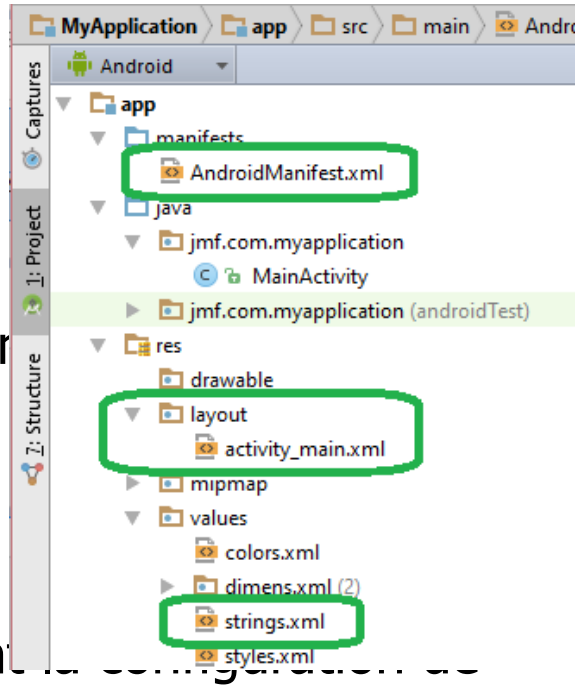
Si problème avec le chemin trop long (sous Windows)

- 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`

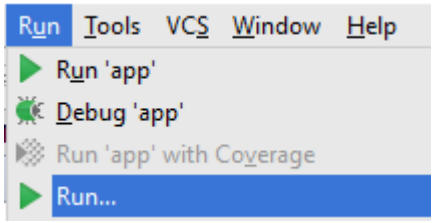
Résumé : les 3 principaux fichiers XML

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



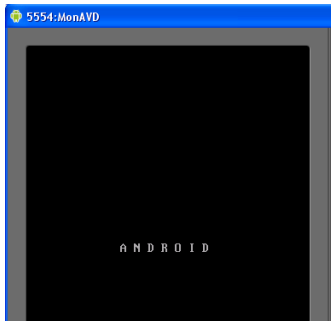
L'exécution (1/3)

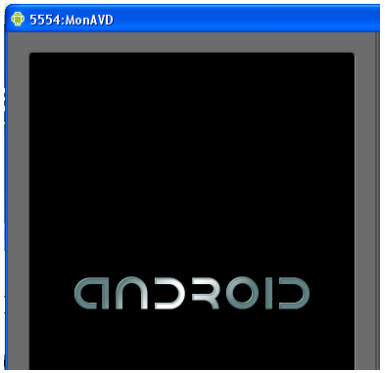
- Sélectionner Run | Run...



- Ou l'icône

L'exécution (2/3)

- La suite des écrans 

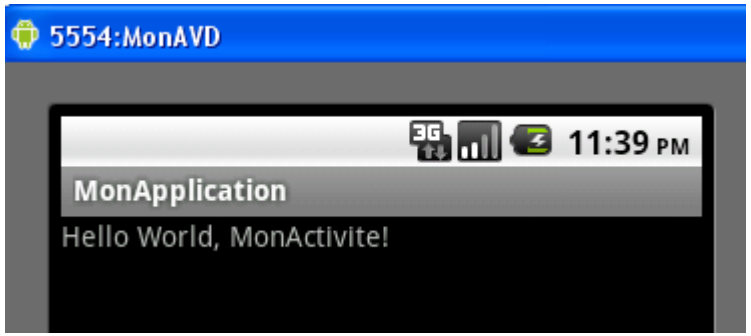
puis 

finit par amener :

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

L'exécution (3/3)

- Apparaît alors votre application Android :



Application Android

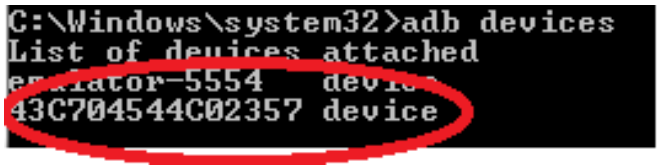
- Pour obtenir cet "Hello World", nous avons eu besoin :
  - de fichiers XML de configuration (`AndroidManifest.xml`),
  - de définitions de constantes (`strings.xml`),
  - d'interfaces graphiques (`activity_main.xml`)
  - en plus de classes Java (presque) créées par le programmeur (`MonActivite.java`) 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"

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 :

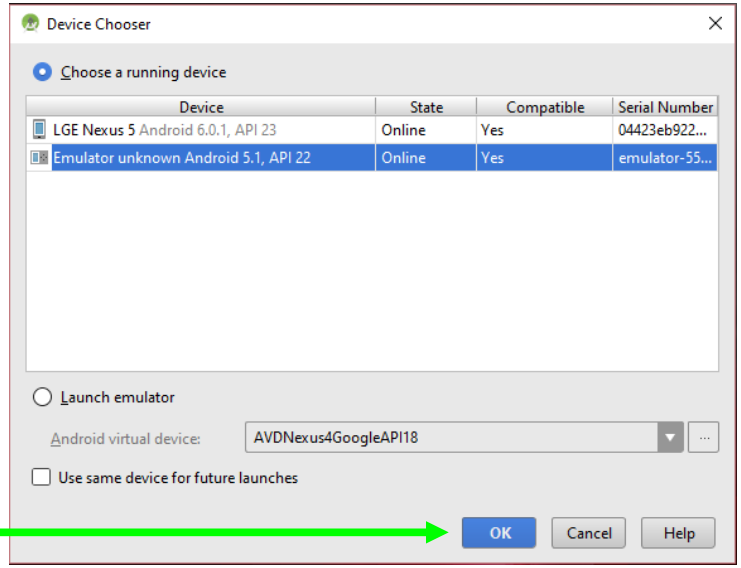


- Remarque : la commande `adb` se trouve dans
  `%RepInstallSDK%\platform-tools`
- Si vous avez une réponse comme :

  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

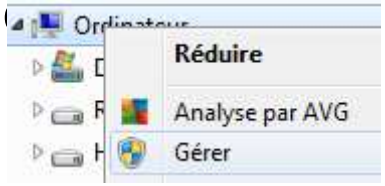
Charger un driver USB de smartphone (1/3)

- 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 Eclipse finisse 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`

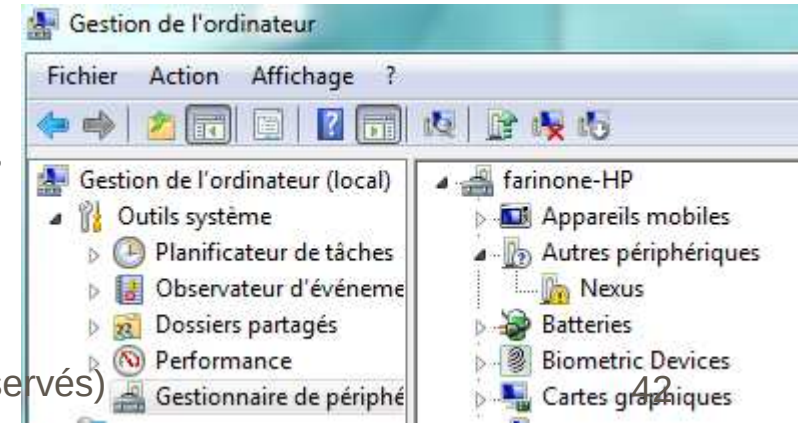
- source : `http://developer.android.com/sdk/win-usb.html`

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

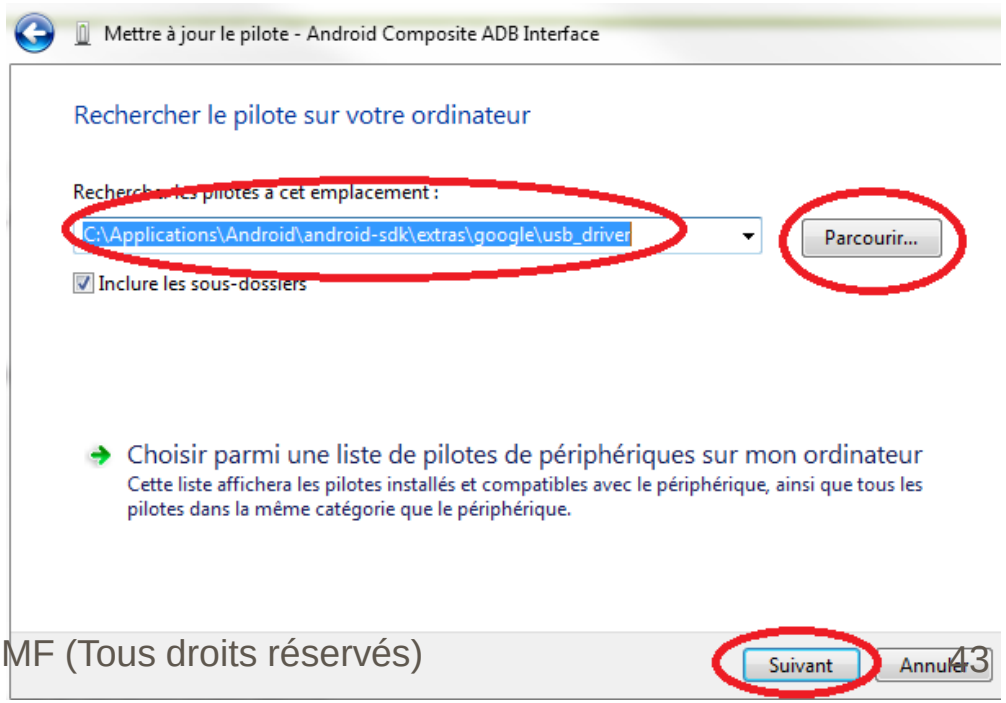


  - Sélectionner Gestionnaire de périphériques dans le panneau gauche, puis Autres périphériques dans le panneau droit



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`

Résumé du chapitre 2

- 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

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)

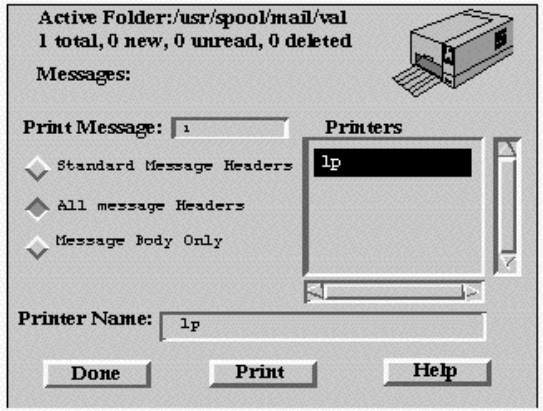
- Correspond à une seule classe Java
- Une activité gère l'affichage et l'interaction d'un écran (IHM)
- Gère les événements, lance l'affichage d'autres écrans, lance du code applicatif
- Suit un cycle de vie déterminé (cf. applets, midlets, servlets, EJB, ...)
- Utilise les `Intent` pour lancer d'autres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

Construction d'une IHM

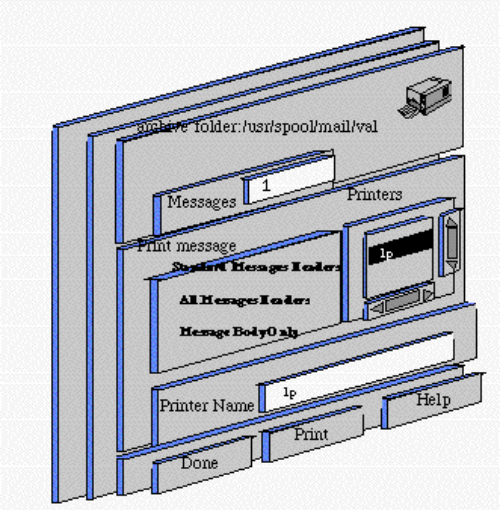
- Plutôt en XML mais
- XML ne peut pas être débogué !
- Tout ne peut pas être fait en XML

Premier principe des IHM (1/4)

- Quand on voit ceci :

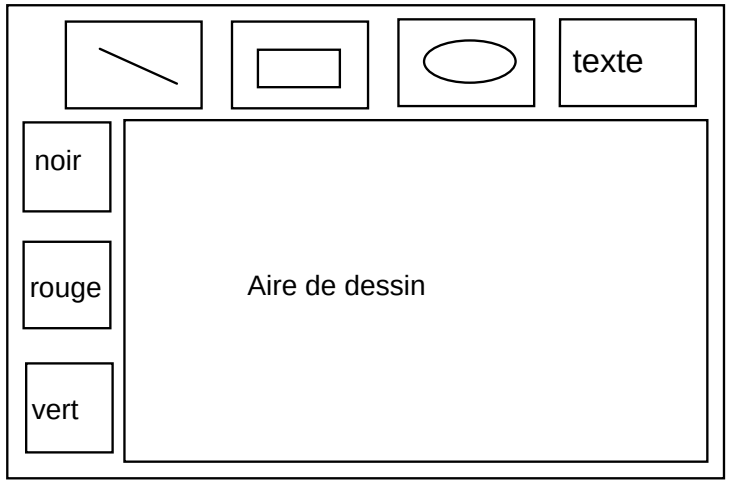


- C'est qu'on a programmé cela :

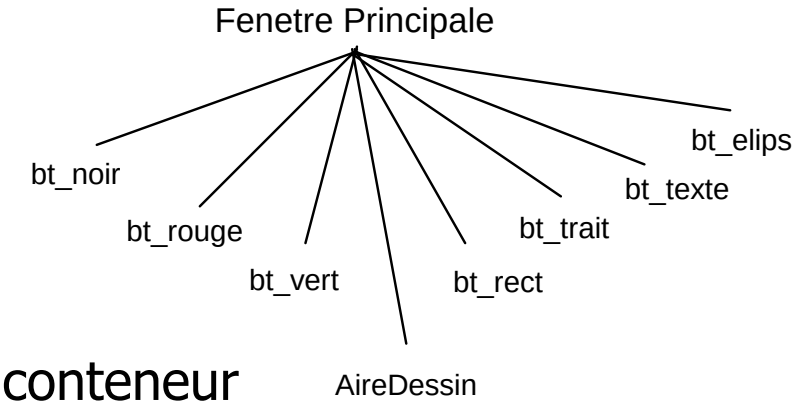


Premier principe des IHM (2/4)

- Si on veut :



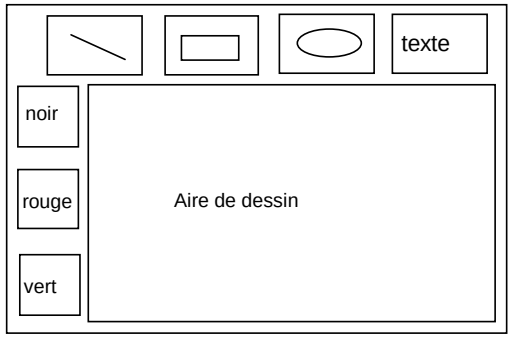
- On doit construire cela :



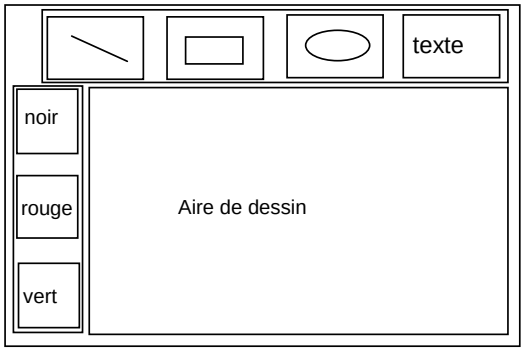
- => Fenetre principale = un conteneur

Premier principe des IHM (3/4)

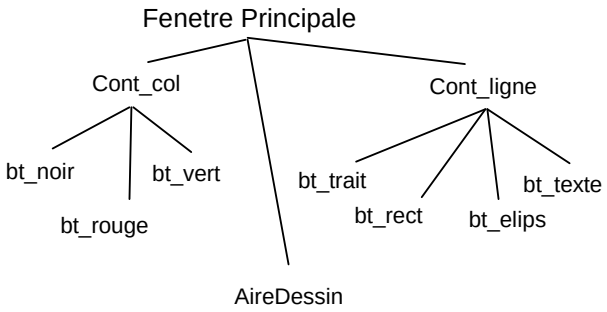
- Plus sûrement, si on veut :



on écrit plutôt :



c'est à dire :



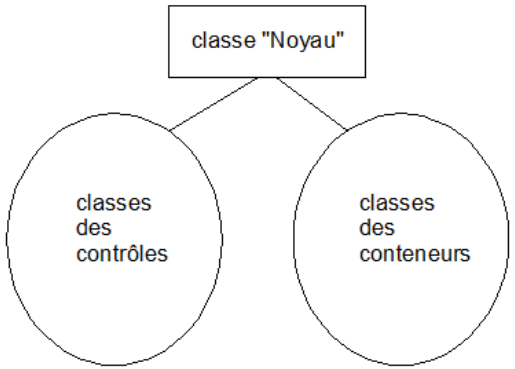
Premier principe des IHM (4/4)

- (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 :

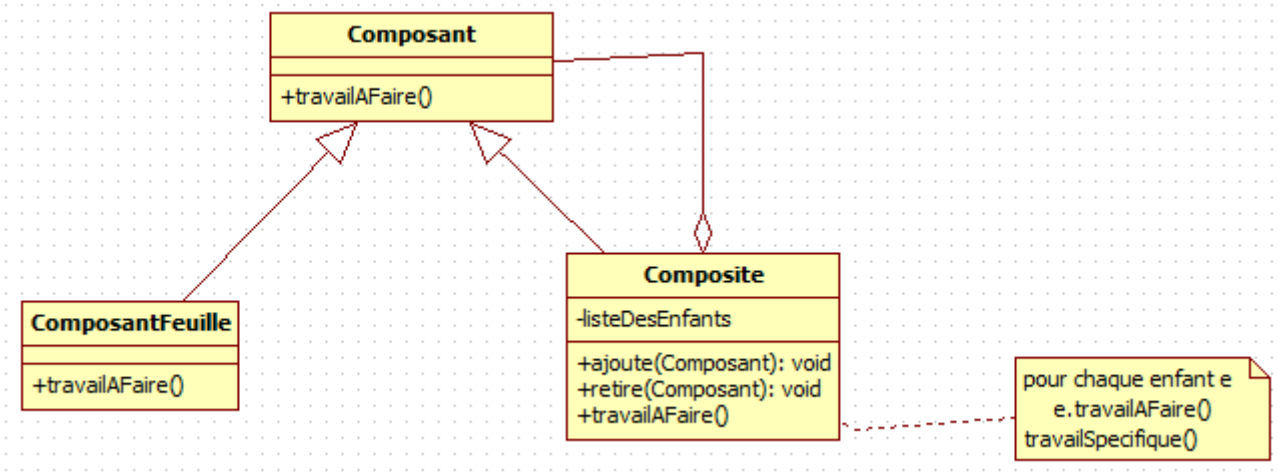


- Question : Comment sont rangées ces classes ?

- 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

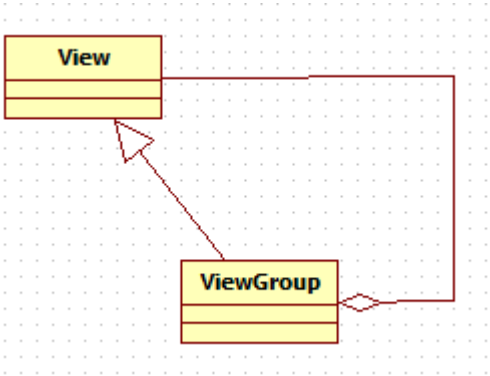
Un composant IHM = 3 parties

- Un composant graphique a 3 parties :
  - les données qu'il représente : le modèle (model)
  - le dessin d'affichage : la vue (view)
  - ce qui prend en charge les actions de l'utilisateur sur ce composant : le controleur (controler)
- C'est l'architecture MVC
- "L'idée est de bien séparer les données, la présentation et les traitements" (*)
- Les traitements sont souvent délégués à un objet autre que le composant graphique lui même (et qui n'est pas, en général, un composant graphique) : programmation par délégation

- (*) source : `http://fr.wikipedia.org/wiki/Paradigme_MVC`

Les fondamentaux d'Android

- La classe "Noyau" de base est la classe `android.view.View` (~ `java.awt.Component` de AWT)
- La classe de base des conteneurs est `android.view.ViewGroup` (~ `java.awt.Container` de AWT)
- On a donc :



- En Android, les conteneurs sont souvent appelés les Layout, les contrôles sont parfois appelés des widgets (window objects)

IHM : construction procédurale vs. déclarative

- En Android on peut construire les IHM en codant en Java des instructions ...
- ... ou en décrivant l'IHM par un fichier XML
- La première solution est celle habituelle de Java (SE Swing et AWT, ME) ou d'autres domaines (Motif, Openwin de Sun, ...)
- La seconde est courante dans certains domaines (Apple, ...)

- Une nouvelle : on peut construire la plupart des IHM en glisser-déposer cf. Interface Builder de NeXT : Jean-Marie Hullot (1989) , Visual Basic, ...

Un second programme

- On veut créer une application Android qui affiche un bouton. Lorsque l'utilisateur actionne le bouton, l'heure est affichée. Le résultat est :

- L'heure est mise à jour lorsqu'on actionne le bouton

Code du second programme (1/2)

- On peut tout coder en Java dans l'activité :

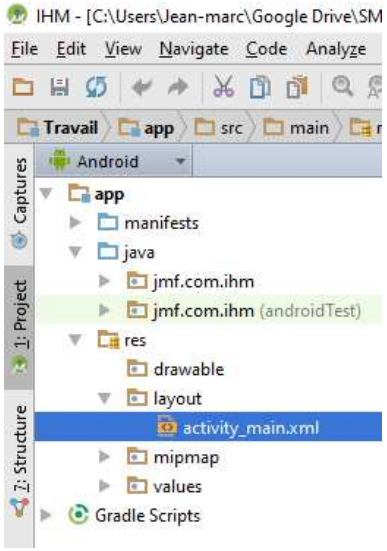
```
package android.jmf;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import java.util.Date;
public class BoutonHeureActivite extends Activity implements View.OnClickListener
{
    private Button btn;
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        btn = new Button(this);
        btn.setOnClickListener(this);
        updateTime();
        setContentView(btn);
    }
    public void onClick(View view) {
        updateTime();
    }
    private void updateTime() {
        btn.setText(new Date().toString());
    }
}
```

Code du second programme (2/2)

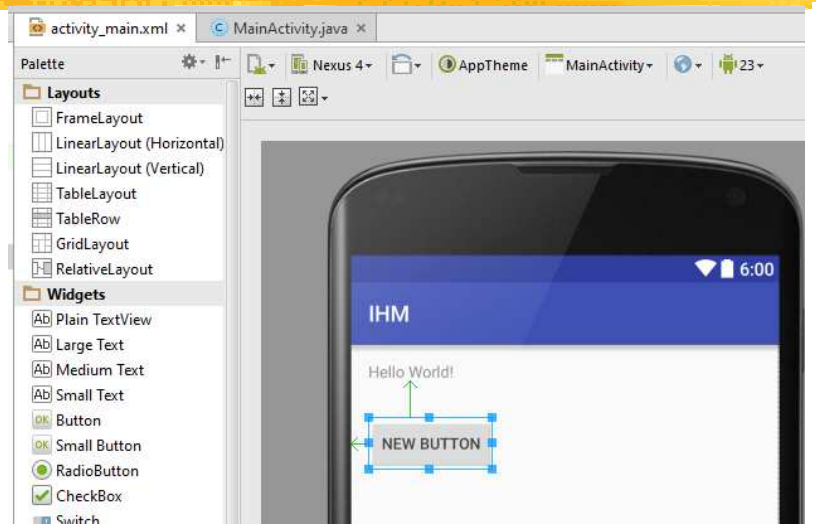
- L'activité est le listener du bouton :
  `public class BoutonHeureActivite extends Activity implements View.OnClickListener`
- Ce qui nécessite d'implémenter la méthode :
  `public void onClick(View view)` qui est lancée lorsqu'on actionne le bouton (technique des listeners cf. événement Java SE)
- Le bouton est mis dans l'activité (`setContentView(btn)`)

Second programme : une autre solution (1/5)

- En général, pour l'IHM on utilise plutôt les fichiers XML
- Par exemple, créer une nouvelle application Android
- Ouvrir le fichier `activity_main.xml` (dans `res/layout/activity_main.xml`)
- Et on peut construire l'IHM en glisser déposer !



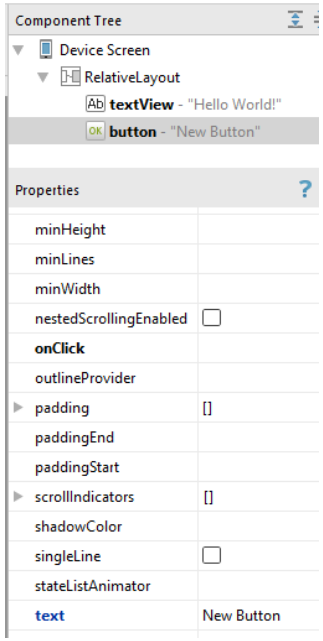
Second programme : une autre solution (2/5)



- La fenêtre de `activity_main.xml` propose deux onglets : l'onglet Design et l'onglet Text
- L'onglet Text donne le code XML de ce fichier
- L'onglet Design (qui met parfois du temps à s'afficher lors de sa première ouverture) permet de visualiser l'IHM correspondant à ce fichier (partie droite) et de construire cette IHM en glisser-déposer avec les éléments graphiques de la partie gauche
- Par exemple, on peut sélectionner le composant `Button` et l'amener dans la partie droite
- Ce constructeur d'IHM est l'ADT Visual Designer

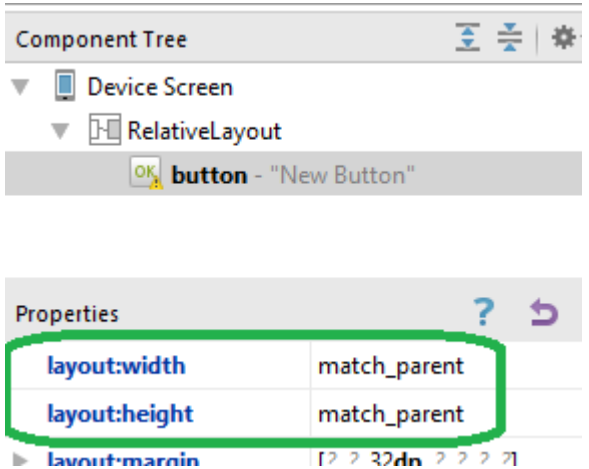
Second programme : une autre solution (3/5)

- L'onglet Component Tree indique l'arborescence des composants graphiques de l'application
- Lorsqu'on sélectionne le composant graphique (quelle que soit la fenêtre !) apparaît l'onglet Properties : ce sont les propriétés du composant graphique



Second programme : une autre solution (4/5)

- On peut changer la largeur et la hauteur d'un composant à l'aide des propriétés `layout:width` et `layout:height`



- Les valeurs de ces propriétés peuvent être
  - `match_parent` (anciennement nommé `fill_parent` avant l'API 8) indique que le composant graphique sera aussi grand en largeur ou hauteur que son conteneur le lui autorise
  - `wrap_content` indiquant que le composant prend seulement la taille qui lui faut en largeur ou hauteur pour s'afficher correctement

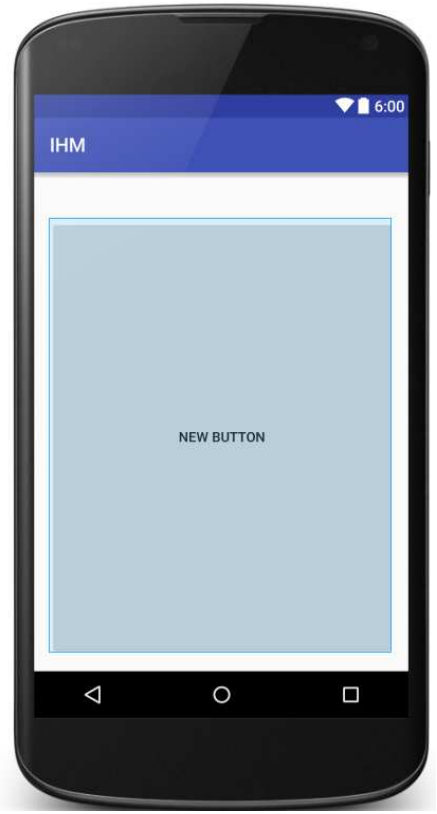
- source : `http://developer.android.com/guide/topics/ui/declaring-layout.html`

Second programme : une autre solution (5/5)

- L'IHM est alors généré cf. les deux onglets de `activity_main.xml` :

activity_main.xml × | app × | AndroidManifest.xml ×

```
<?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:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

    <Button
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="New Button"
        android:id="@+id/button"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="32dp"
        android:layout_alignParentEnd="false" />
</RelativeLayout>
```



Correspondance attribut - méthode

- Pour positionner une propriété (= valeur d'attribut = données = ...) d'un composant graphique on peut le faire soit en XML soit par appel d'une méthode appropriée
- Dans le fichier XML, on positionne une propriété d'un composant graphique à l'aide d'une valeur d'attribut de sa balise XML
- Donc il y a une correspondance entre les noms d'attribut d'une balise XML et une méthode, pour positionner une propriété d'un composant graphique. On a, par exemple :

XML Attributes		
Attribute Name	*Related Method*	*Description*
android:alpha	setAlpha(float)	alpha property of the view, as a value between 0 (completely transparent) and 1 (completely opaque).
android:background	setBackgroundResource(int)	A drawable to use as the background.
android:clickable	setClickable(boolean)	Defines whether this view reacts to click events.
android:contentDescription	setContentDescription(CharSequence)	Defines text that briefly describes content of the view.
android:drawingCacheQuality	setDrawingCacheQuality(int)	Defines the quality of translucent drawing caches.
android:duplicateParentState		When this attribute is set to true, the view gets its drawable state (focused, pressed, etc.) from its direct parent rather than from itself.
android:fadeScrollbars	setScrollbarFadingEnabled(boolean)	Defines whether to fade out scrollbars when they are not in use.
android:fadingEdgeLength	getVerticalFadingEdgeLength()	Defines the length of the fading edges.
android:filterTouchesWhenObscured	setFilterTouchesWhenObscured(boolean)	Specifies whether to filter touches when the view's window is obscured by

- bibliographie :
  `http://developer.android.com/reference/android/view/View.html`

Identifier les composants = la méthode "miracle"

- Le fichier `activity_main.xml` repère les composants par `android:id`

```
<Button
    android:id="@+id/button1"
    android:layout_width="fill_parent"
```

- Dans cet exemple il s'agit de `button1`
- Le composant est manipulé par cet identifiant dans le programme Java à l'aide de la méthode ("miracle") `findViewById(R.id.`*`nomIdentifiant`*`);`
- La valeur *`nomIdentifiant`* est celle qui apparaît dans le fichier `activity_main.xml` après `@+id/`
- Pour cet exemple ce sera `findViewById(R.id.button1);`

Le traitement de `setContentView()`

- `setContentView()` n'est pas banale ! Elle est capable de lire un fichier xml, l'analyser, construire des objets Java (souvent des `View`) à partir de ces données, les agencer pour construire une IHM
- Ce mécanisme est dit inflate

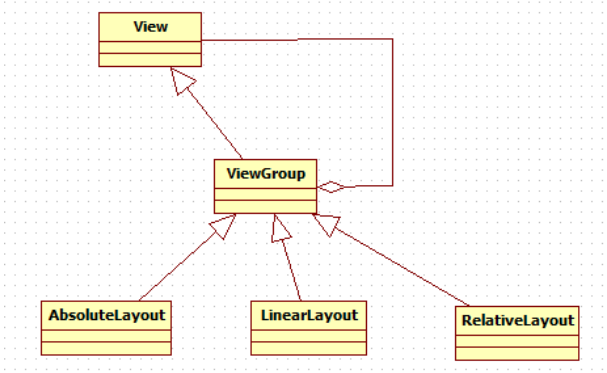
Second programme, solution 2 : le code

- Comme toute l'IHM est faite dans `activity_main.xml`, voici le code de l'activité :

```
package android.jmf;

import java.util.Date;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class BoutonHeure2Activite extends Activity implements View.OnClickListener {
     private Button btn;
     @Override
     public void onCreate(Bundle icicle) {
          super.onCreate(icicle);
          setContentView(R.layout.activity_main);
          btn=(Button)findViewById(R.id.button1);
          btn.setOnClickListener(this);
          updateTime();
     }
     public void onClick(View view) {
          updateTime();
     }
     private void updateTime() {
          btn.setText(new Date().toString());
     }
}
```

Les `Layout`

- Les conteneurs Android sont souvent des `XXXLayout` !
- C'est un peu différent de Java AWT ou Swing. Un Layout Android est un container et un Layout AWT à la fois
- Les principaux Layout Android sont :
  - `LinearLayout` (~ un containeur AWT géré par un FlowLayout AWT)
  - `RelativeLayout`
  - `AbsoluteLayout` (déprécié depuis l'API 3 !)
- On a donc :



LinearLayout

- Les composants à l'intérieur d'un `LinearLayout` sont rangés les uns à la suite des autres horizontalement ou verticalement
- Principales propriétés d'un `LinearLayout` : l'orientation, le mode de remplissage (fill model)

Orientation d'un `LinearLayout`

- L'orientation indique si le `LinearLayout` présente ces contenus sur une ligne (horizontalement) ou sur une colonne (verticalement)
- La propriété d'orientation à utiliser pour un `LinearLayout` dans le fichier XML est l'attribut `android:orientation` de la balise `LinearLayout`. Les valeurs possibles pour cette propriété sont `vertical` et `horizontal`
- La valeur `vertical` indique que les contenus seront les uns en dessous des autres, la valeur `horizontal` indique qu'ils seront les uns à la suite des autres
- L'orientation peut être modifiée à l'éxécution par la méthode `setOrientation()` lancée sur le `LinearLayout` en précisant la valeur `LinearLayout.HORIZONTAL` ou `LinearLayout.VERTICAL`

Mode de remplissage (fill model) d'un `LinearLayout`

- Cela concerne les attributs `android:layout_width` et `android:layout_height` à positionner sur les composants graphiques contenus dans le `LinearLayout`
- Les valeurs possibles de ces propriétés peuvent être :
  - une valeur exacte de pixel (125px pour 125 pixels) : c'est fortement déconseillé
  - `wrap_content` qui indique que le composant prend la taille qu'il faut pour s'afficher correctement entièrement
  - `match_parent` (anciennement `fill_parent` avant l'API 8) indique que le composant remplit complètement la dimension indiquée du `LinearLayout`

Le `RelativeLayout`

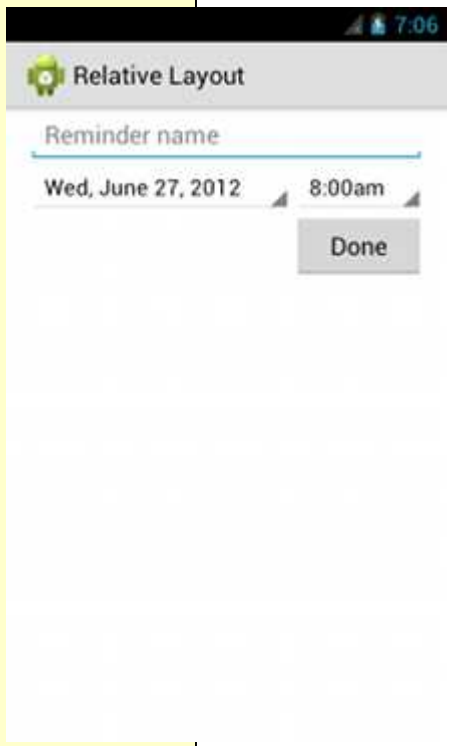
- = un conteneur qui permet de placer ses contenus les uns par rapport aux autres
- C'est le conteneur proposé dans le (nouveau) outil de construction d'IHM
- Les `Views` contenues dans le `RelativeLayout` indiquent leur positionnement à l'aide de leurs attributs (dans le fichier XML de l'IHM)
- Il ne doit pas y avoir de dépendance cyclique (bon sens)
- Les `Views` indiquent leur position par rapport à la vue parente ou leurs `Views` soeurs (en utilisant leur `id`)
- Les valeurs des attributs sont soit des `boolean`, soit l'`id` d'une autre `View`

Attributs possibles pour une `View` dans un `RelativeLayout`

- Les `Views` dans un `RelativeLayout` peuvent utiliser les attributs :
  - `android:layout_alignParentTop` : si `true`, le haut de la `View` est calé sur le haut de la vue parente
  - `android:layout_centerVertical` : si `true`, la `View` est centrée verticalement à l'intérieur de la vue parente
  - `android:layout_below` : le haut de la vue est en dessous de la `View` indiquée (par son l'`id`)
  - `android:layout_toRightOf` : le coté gauche de la vue est à droite de la `View` indiquée (par son l'`id`)
- Il y a d'autres valeurs possibles voir à `http://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html`

RelativeLayout : un exemple

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```



`@+id` **vs.** `@id`

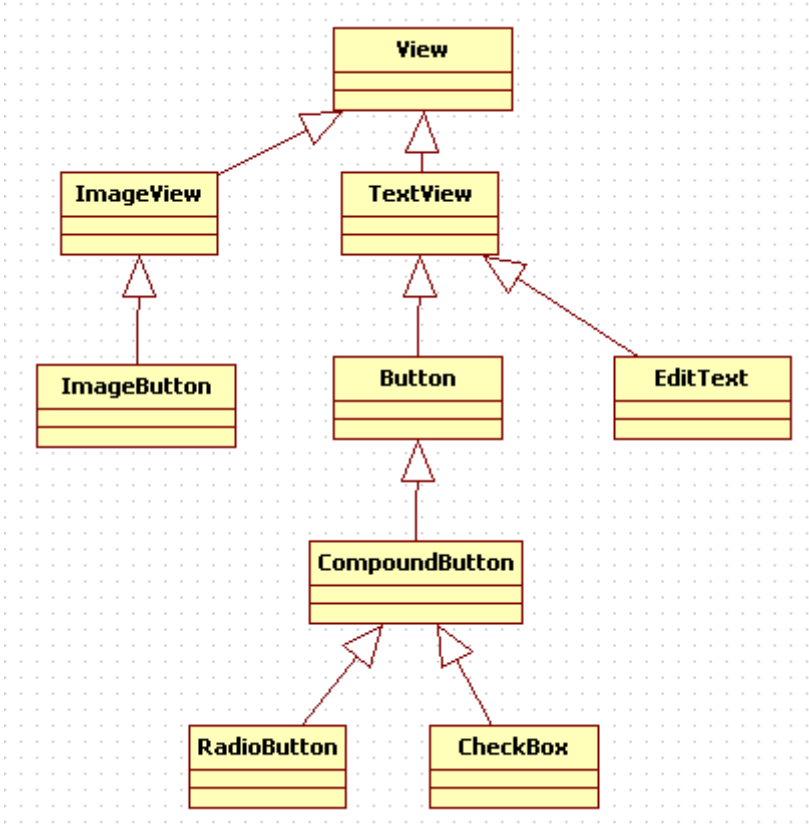
- Dans le fichier xml précédent, on utilise parfois `@+id` et parfois `@id`
- `@+id` indique qu'il faut créer, si nécessaire, une nouvelle entrée dans `R.java` (et sa classe interne `id`)
- `@id` repère simplement l'identificateur `id` et il n'y a pas de création dans `R.java`
- En fait cela fonctionne si on met toujours le `+` !

Les contrôles Android

- Ce sont les composants graphiques que voient l'utilisateur, avec lesquels il agit sur (contrôle) l'interface graphique
- Appelés dans certains domaines, les contrôles
- En Android ce sont (par exemple) :
  - les zones de texte non éditable (~ `Label` AWT) ou éditable (~ `TextComponent` AWT) : `TextView`
  - les boutons (~ `Button` AWT) : `Button`
  - les zones de texte éditables (~ `TextField` et `TextArea` de AWT) : `EditText`
  - les cases à cocher (~ `Checkbox` AWT) : `CheckBox` et les boutons radio `RadioButton` à regrouper dans un ensemble
- Toutes ces classes sont dans le package `android.widget` et dérivent de `android.view.View`

Arborescence des principaux contrôles Android

- Les classes de composants non conteneurs (contrôles) sont rangées dans l'arbre d'héritage :



TextView

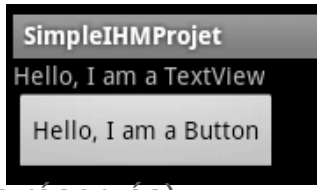
- Peut servir de zone de texte non éditable (~ `Label` AWT) et dans ce cas, sert souvent pour présenter les widgets qui suivent
- Propriétés importantes :
  - `android:text` : le texte du `TextView`
  - `android:typeface` : le type de police utilisée (monospace, ...)
  - `android:textStyle` : le style (`italic` pour l'italique, `bold_italic` pour gras et italique, ...)
  - `android:textColor` pour la couleur d'affichage du texte. Les valeurs sont en hexadécimal en unité RGB (par exemple `#FF0000` pour le rouge)

Arborescence des composants graphiques dans une IHM

- Faire une IHM c'est mettre des composants dans des composants dans des composants ... On peut le faire par programmation en utilisant `addView(View)` d'un `ViewGroup` ou dans un fichier XML
- Par exemple le fichier :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

= un `TextView` et un `Button` dans un `LinearLayout` représente l'IHM :



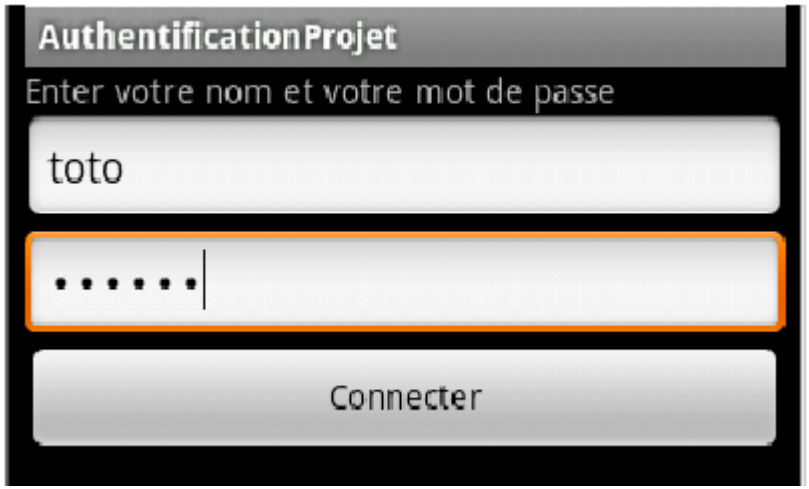
ScrollView

- Si le contenu d'une zone de texte est trop important par rapport à l'écran, on ne voit pas tout ce contenu
- L'environnement d'exécution doit pouvoir ajouter des barres de défilement à cette zone (=scrollbars). Pour cela, on met la zone de texte dans une `ScrollView`
- Le fichier XML d'IHM contient alors :

```
<ScrollView
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     ...
     android:paddingTop="8dp" >

     <TextView
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:padding="8dp"
          ...
     />
</ScrollView>
```

Exercice

- Construction d'une IHM avec des composants Android
- Construire et faire afficher l'activité :



- Remarque : le troisième (si, si !) composant n'affiche pas l'écho des caractères

La gestion des événements

- Deux moyens :
  - 1°) créer un auditeur d'événements (classe qui implémente une interface connue) et l'enregistrer auprès du composant (`View`)
  - 2°) les `View` sont elles mêmes auditrices de certains événements : (touché de l'écran). Spécialiser la méthode adaptée et lancée lorsque l'événement survient
- 1°) est classique (Java SE, Java ME). Les interfaces sont des interfaces internes à la classe `View` et de nom `OnXXXListener` (donc des interfaces de nom `View.OnXXXListener`). Cela nécessite d'implémenter une méthode de nom `onXXX()`. On enregistre un auditeur par `setOnXXXListener(View.OnXXXListener l)`
- 2°) permet d'écrire directement la gestion de certains événements qui peuvent se produire dans la `View`

Créer un auditeur d'événements : exemple

- Le code peut être :

```
private OnClickListener lAuditeurDuBouton = new OnClickListener() {
    public void onClick(View v) {
      // code lancé lorsque le bouton est cliqué
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Récupération du Button à partir de l'IHM en XML
    Button button = (Button)findViewById(R.id.leBeauBouton);
    // Enregistrer l'auditeur auprès du bouton
    button.setOnClickListener(lAuditeurDuBouton);
    ...
}
```

Méthodes lancées par les auditeurs d'événements

- `onClick()` (de `View.OnClickListener`) est lancée lorsque l'utilisateur touche le composant graphique, ou après appui sur enter alors que le composant a le focus
- `onLongClick()` (de `View.OnLongClickListener`) : idem que si dessus mais après un appui de plus de 1 seconde
- `onKey()` (de `View.OnKeyListener`) est lancée après appui et relachement d'un touche clavier
- `onTouch()` (de `View.OnTouchListener`) est lancée pour toute action de toucher (appui, relachement, mouvement de l'utilisateur sur l'écran)
- `onCreateContextMenu()` (de `View.OnCreateContextMenuListener`) est lancée pour créer un menu contextuel

L'attribut `android:onClick`

- On peut indiquer dans le fichier `.xml` de description d'IHM, la méthode qui sera lancée sous une certaine action sur un composant graphique
- Par exemple, l'attribut `android:onClick` d'un composant graphique indique le nom de la méthode qui sera lancée si on clique sur cette `View`
- Par exemple, dans le fichier de description de l'IHM, on écrit

```
<Button
    android:id="@+id/push"
    ...
    android:onClick="onClickEmpiler">
</Button>
```

- Dans l'activité chargeant l'IHM contenant ce `Button`, on pourra écrire :

```
public void onClickEmpiler(View v){
    // traitement lancé lorsque l'utilisateur clique sur le Button d'id push
}
```

Plus sur les `Intents`

- Un `Intent` est une description abstraite d'une opération à faire.
- Un `Intent` peut être utilisé avec :
  - `startActivity()` pour demander à lancer une activité
  - `broadcastIntent()` pour demander à contacter un `BroadcastReceiver`
  - `startService()` ou `bindService()` pour communiquer avec un `Service`
- Essentiellement un Intent est formé :
  - d'une description d'action à effectuer
  - de données utiles pour cette action
- source : `https://developer.android.com/reference/android/content/Intent.html`

"Enchaîner" les écrans (1/2)

- Pour passer d'un écran à un autre, il faut écrire le code

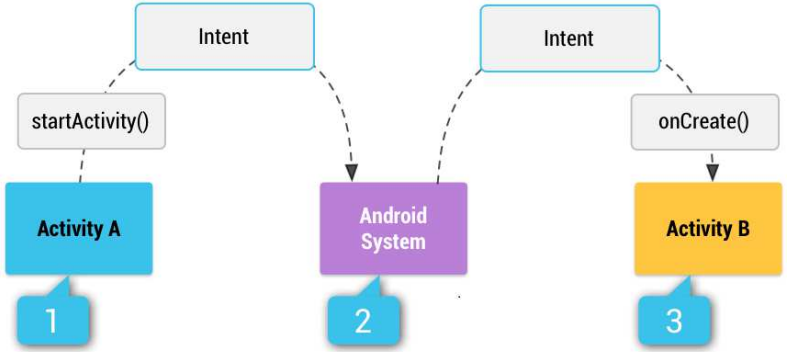
```
Intent i0 = new Intent(getApplicationContext(), NouvelleActivity.class); //1
startActivity(i0);
```

  et déclarer la nouvelle activité `NouvelleActivity.class` (le futur écran) dans `AndroidManifest.xml`

- `public void startActivity (Intent intent)` est une méthode de la classe `Activity` permettant de lancer une autre Activity (qui affichera un nouvel écran). `intent` est l'`Intent` (l'intention) qui prépare ce lancement

"Enchaîner" les écrans (2/2)

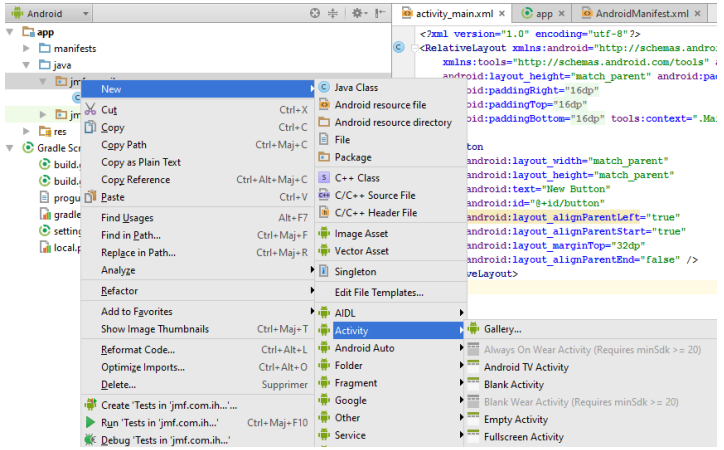
- En fait cet `Intent` est envoyé à l'environnement d'exécution. Celui-ci le redirige vers l'activité concernée



- Le premier argument du constructeur de l'`Intent` doit être le `Context`. Si l'appel est fait dans une activity `MonActivity`, `this` est souvent utilisé et convient car `Activity` dérive de `Context`
- On utilise souvent `MonActivity.this` quand on est dans un listener d'événement. Mais, la méthode `getApplicationContext()` est la plus pratique

Créer une nouvelle activité

- Dans un projet, on peut créer une nouvelle activité comme indiqué dans les diapos précédentes : écrire une classe dérivant de `Activity`, redéfinir les méthodes `onCreate()`, ... et déclarer cette activité dans l'`AndroidManifest.xml`
- Avec l'environnement de développement, si on utilise clic droit sur le répertoire java paquetage | New | Activity,



en complétant les écrans qui suivent, l'activité est en partie créée et sa déclaration correcte (fils de l'élément `application`) dans l'`AndroidManifest.xml` aussi !

```
    </intent-filter>
</activity>
<activity android:name=".Main2Activity" >
</activity>
```

Passer de données entre activités grâce aux `Intent`

- Les `Intent` servent parfois d'enveloppes pour passer des informations d'une `Activity` à une autre. On utilise pour cela une des méthodes `public Intent putExtra(String nomDeLExtra, unType valeur)`
- Par exemple

```
Intent i = new Intent(leContexte, MapStationActivity.class);
i.putExtra("latitude", latitudeDuPointCourant);
i.putExtra("longitude", longitudeDuPointCourant);
startActivity(i);
```

- Dans une `Activity`, on récupére l'`Intent` qui a lancé l'`Activity` par `getIntent()`. On peut alors récupérer tous les extras de l'`Intent` par `getExtras()`, et, par la suite, un extra associé à une entrée par `get`*`TypeEntrée`*`(nomEntrée, valeurParDefaut)`, `valeurParDefaut` est la valeur retournée si il n'y a pas d'extra associé à `nomEntrée` dans l'`Intent`
- Par exemple :

```
double laLatitudeDeLaStation =
getIntent().getExtras().getDouble("latitude", 0);
```

Un simple avertissement : `Toast`

- Une fenêtre de dialogue qui affiche un message pendant 2 (`Toast.LENGTH_SHORT`) ou 5 (`Toast.LENGTH_LONG`) secondes est un composant graphique Android : le Toast
- On le construit et on l'affiche avec le code

```
Toast leToast = Toast.makeText(leContexte, "texteAAfficher", Toast.LENGTH_LONG);
leToast.show();
```

- Une des méthodes qui construit un `Toast` est la méthode statique :
  `public static Toast makeText (Context context, CharSequence text, int duree)`
  `context` est le contexte à utiliser. En général on passe l'activité courante
  `text` est la chaîne de caractères à afficher
  `duree` est la durée d'affichage `LENGTH_SHORT` ou `LENGTH_LONG`
- Attention construire le `Toast` ne l'affiche pas : il faut utiliser `show()` pour cela
- Et donc finalement on écrit souvent :

```
Toast.makeText(leContexte, "texteAAfficher", Toast.LENGTH_LONG).show();
```

Code de "trace" en Android

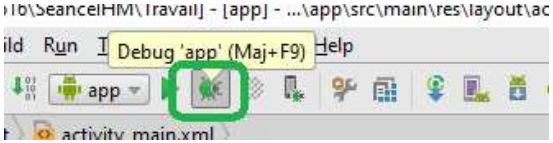
- La classe `android.util.Log` propose plusieurs méthodes de trace (de Log) hierarchisées. Ces méthodes ont pour nom une seule lettre. Ce sont, dans l'ordre les méthodes `v()` (verbose), `d()` (debug), `i()` (information), `w()` (warning) et `e()` (erreur)
- Ces méthodes ont deux arguments : (`String tag, String msg`)
- Elles permettent de visualiser des traces lors de l'exécution en utilisant l'onglet LogCat. On peut filtrer ces traces en ne laissant afficher que les log de balise `tag`
- Par exemple le code

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
    case R.id.new_game:
        Log.v("JMF", "Une nouvelle partie ?");
        Log.d("JMF", "Une nouvelle partie ?");
        Log.i("JMF", "Une nouvelle partie ?");
        Log.w("JMF", "Une nouvelle partie ?");
        Log.e("JMF", "Une nouvelle partie ?");
    return true;
    // ...
    }
}
```

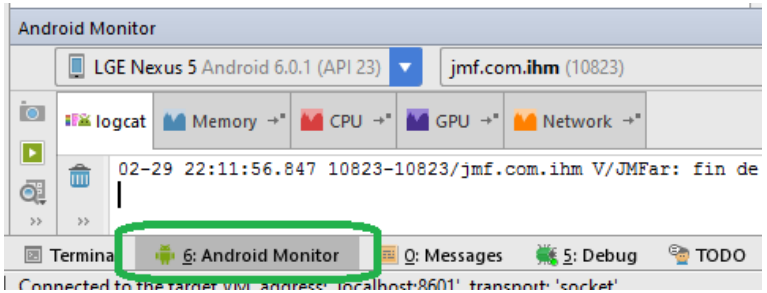
permet de voir les sorties dans l'onglet LogCat

L'onglet logcat

- Les traces de Log sont affichés dans l'onglet logcat
- Pour afficher cet onglet, il faut lancer l'exécution en mode Debug : bouton



- Faire afficher l'onglet 6: Android Monitor (en bas de l'IDE)



- Il y a alors plusieurs onglets disponibles utiles lors de l'exécution de l'app dont l'onglet logcat
- On peut faire des filtres sur les sorties dans la zone de texte de recherché de cet onglet

Les Intents implicites et explicites

- On peut construire un Intent en indiquant explicitement la classe de l'objet qui gérera l'Intent. Par exemple, on écrit :

```
new Intent(getApplicationContext(), ActivityClassName.class)
```

- C'est un Intent explicite
- Mais on peut aussi indiquer au système l'intention à faire une action. L'environnement d'exécution (= le système), recherche alors les activités qui peuvent satisfaire cette action, les propose dans une liste si il y en a plusieurs, ou lance l'activité s'il est unique
- On utilise pour cela les Intents implicites
- Le programmeur peut se construire des Intents implicites
- Si aucune activité ne peut gérer l'Intent, l'application émettrice est arrêtée. Pour éviter cela, il est bon d'écrire :

```
Intent intent = new Intent(...)....
if (intent.resolveActivity(getPackageManager()) != null) {
     startActivity(intent);
}
```

Les Intents implicites courants

- Un smartphone Android possède des applications pré-installées
- Ces applications sont sensibles à des actions standards, pouvant être lancées par des Intents implicites. Par exemple :
  - Envoi d'un SMS :

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("sms:"));
intent.putExtra("sms_body", "hello world");
startActivity(intent);
```

  - Lancer une écoute de musique :

```
Intent intent = new Intent(Intent.ACTION_VIEW);
File hello = new File("/sdcard/hello.mp3");
intent.setDataAndType(Uri.fromFile(hello), "audio/mp3");
startActivity(intent);
```

  - Appeler un numéro de téléphone :

```
String url = "tel:43556";
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse(url));
startActivity(intent);
```

  - Lancer un navigateur :

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.google.com"));
startActivity(intent);
```

Action et catégorie d'un `Intent`

- Les `Intent` sont classés par catégorie (ensemble d'`Intent`) et d'action (un élément dans cette ensemble)
- Catégorie et action d'un `Intent` apparaissent comme sous élément de l'élément `intent-filter` d'un composant Android dans le fichier `AndroidManifest.xml`
- Exemple de catégories :

CATEGORY_TAB	Activité prévue pour fonctionner en tant qu'onglet d'une autre activité
CATEGORY_HOME	Affichage de l'écran d'accueil au démarrage du mobile ou après pression du bouton *Home*
CATEGORY_LAUNCHER	Activité initiale d'une tâche listée dans le gestionnaire d'applications
CATEGORY_DEFAULT	Obligatoire pour recevoir des intents implicites. Ainsi, elles pourront être trouvées par startActivity
CATEGORY_BROWSABLE	Activité qui peut être invoquée par un navigateur pour afficher les données référencées par un lien

Les Intents implicites (1/2)

- On peut se construire des Intent implicites. Par exemple :

```
<activity android:name=".HelloWorldActivity"
     android:label="@string/hello_world_name">
     <intent-filter>
          <action android:name="bonjour.ACTION" />
          <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>
</activity>
```

- Cette activité HelloWorldActivity est sensible à l'action "bonjour.ACTION". Elle peut être lancée par :

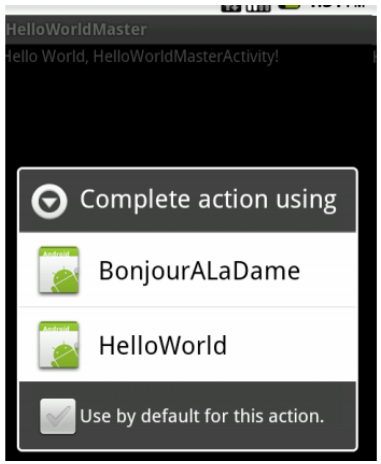
```
Intent intent = new Intent();
intent.setAction("bonjour.ACTION");
startActivity(intent);
```

Les `Intent`s implicites (2/2)

- Si un autre composant peut être invoqué par la même action, par exemple :

```
<activity android:name=".DisBonjourALaDameActiviy"
     android:label="@string/dis_bonjour_a_la_dame_name">
     <intent-filter>
          <action android:name="bonjour.ACTION" />
          <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>
</activity>
```

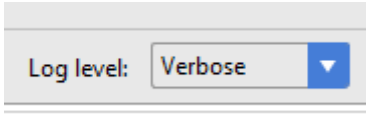
lorsque l'activité `HelloWorldMasterActivity` demande à traiter cette action, le système présente :



- ...y compris (et surtout ?!) si cet autre composant (= autre activité ici) appartient à une autre application

Traces (log) en Android

- On indique le niveau de trace dans la liste Log level :



- Le niveau verbose affichera toutes les traces du filtre, le niveau info n'affichera que les traces info, warning, erreur et assert

Exercice

- Gérer les actions des utilisateurs sur une IHM
- Bonus : traiter les `Intents` implicites

Fin