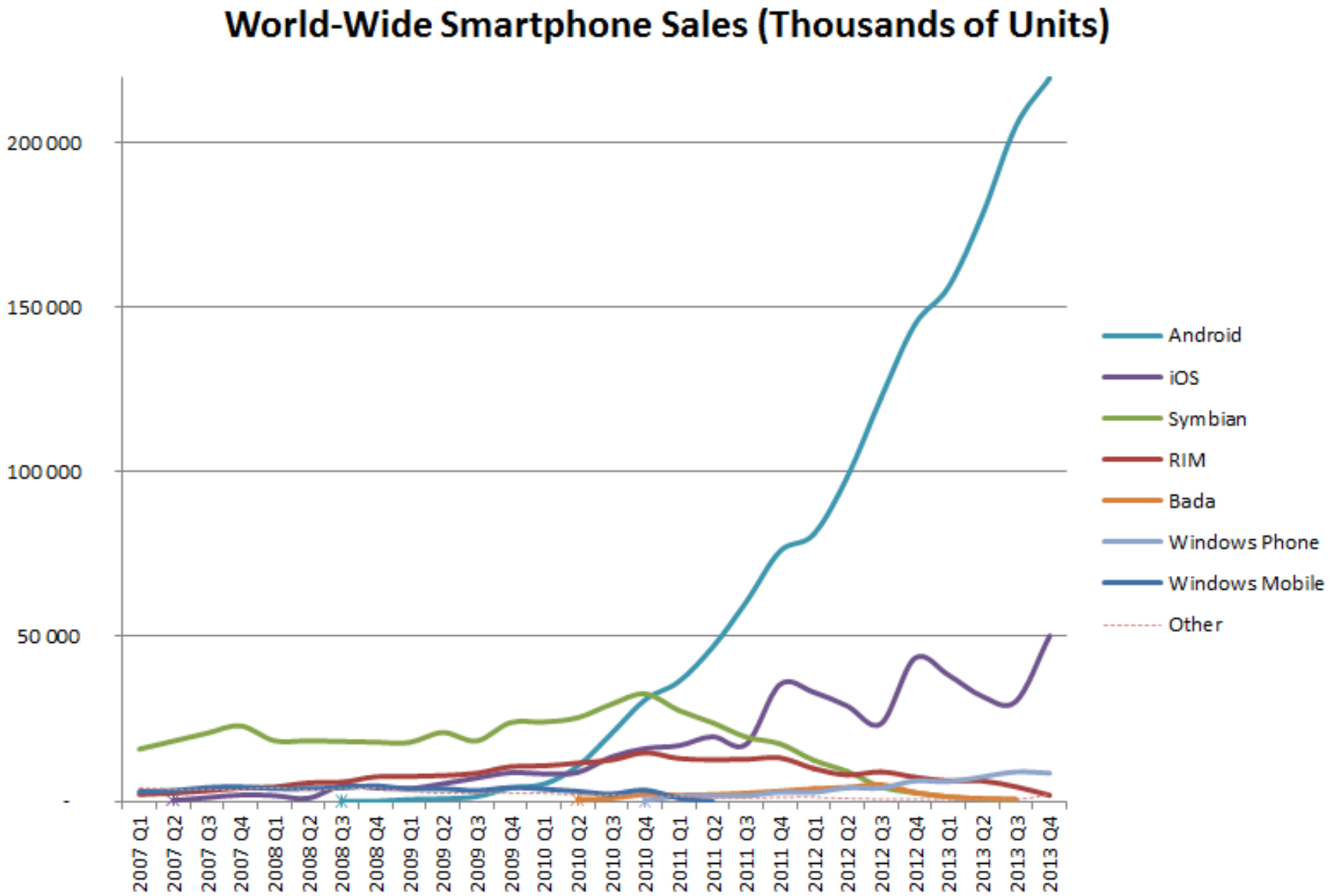
# Android 101

Rémi Forax

# OS/smartphone dans le monde



# Historique

- Octobre 2003 : conception d'un OS mobile par Android Inc. (co-fondé par Andy Rubin)
- Août 2005 : rachat d'Android Inc par Google
- Novembre 2007 : création consortium Open Handset Alliance (Google + industriels) et de l'Android Open SourceProject (AOSP), version beta sous licence Open Source Apache
- Septembre 2008 : 1ère version finale avec le téléphone HTC Dream
- Début 2015 : sortie de la dernière version majeure (Android 5.0 - Lollipop)

# Versions antérieures d'Android

# Lollipop – Android 5.0

# Versions d'Android

| Version | Codename | API | Distribution 2014 (2013) |
|---|---|---|---|
| 2.2 | Froyo | 8 | 0.4 (1.3/8.1) |
| 2.3.3-2.3.7 | Gingerbread | 10 | 6.9 (20.0/45.4) |
| 4.0.3-4.0.4 | Ice Cream Sandwich | 15 | 5.9 (16.1/29.0) |
| 4.1 | Jelly Bean | 16 | 17.3 (35.5/12.2) |
| 4.2 | Jelly Bean (MR1) | 17 | 19.4 (16.3/1.4) |
| 4.3 | Jelly Bean (MR2) | 18 | 5.9 (8.9) |
| 4.4 | KitKat | 19 | 40.9 (1.8) |
| 5.0 | Lollipop | 21 | 3.3 |

https://developer.android.com/about/dashboards/index.html

19 mars 2015

# Android sous le capot

- Système linux (GPL)
  - Quelques patchs spécifiques
- Librairie C particulière (APL)
  - bionic
- Librairies par défaut:
  - SQLite, OpenGL ES, WebKit, ...
- Réutilise la plateforme Java, mais n'est pas Java
  - VM particulière (Dalvik)
  - DEX file (.dex pas .class)

# Developper sous Android

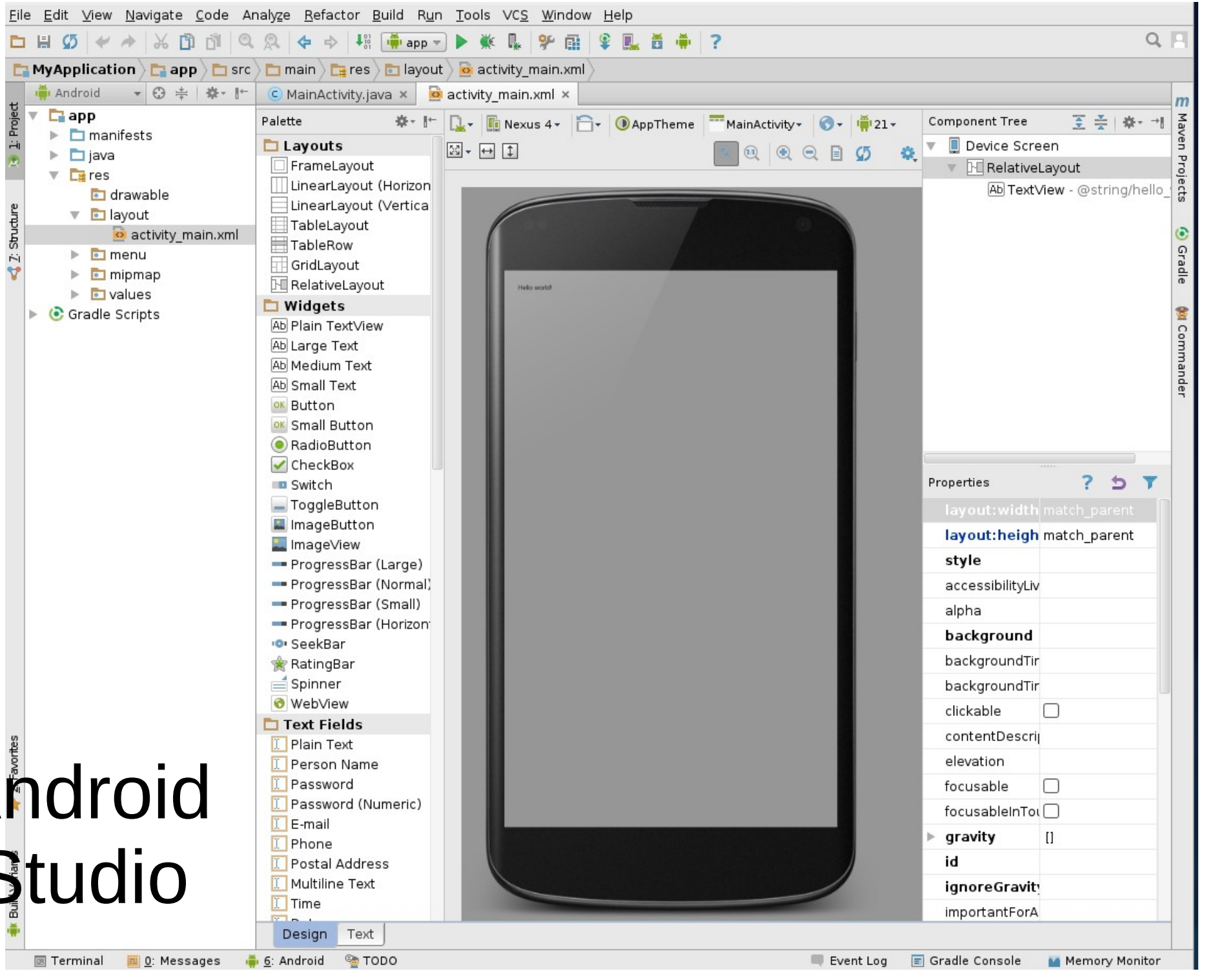
Android ADT: Android SDK + Eclipse

puis download Platform + Virtual Device pour l'émulateur

Android Studio: Android SDK + IntelliJ Idea

puis download Platform + Virtual Device pour l'émulateur

https://developer.android.com/tools/index.html

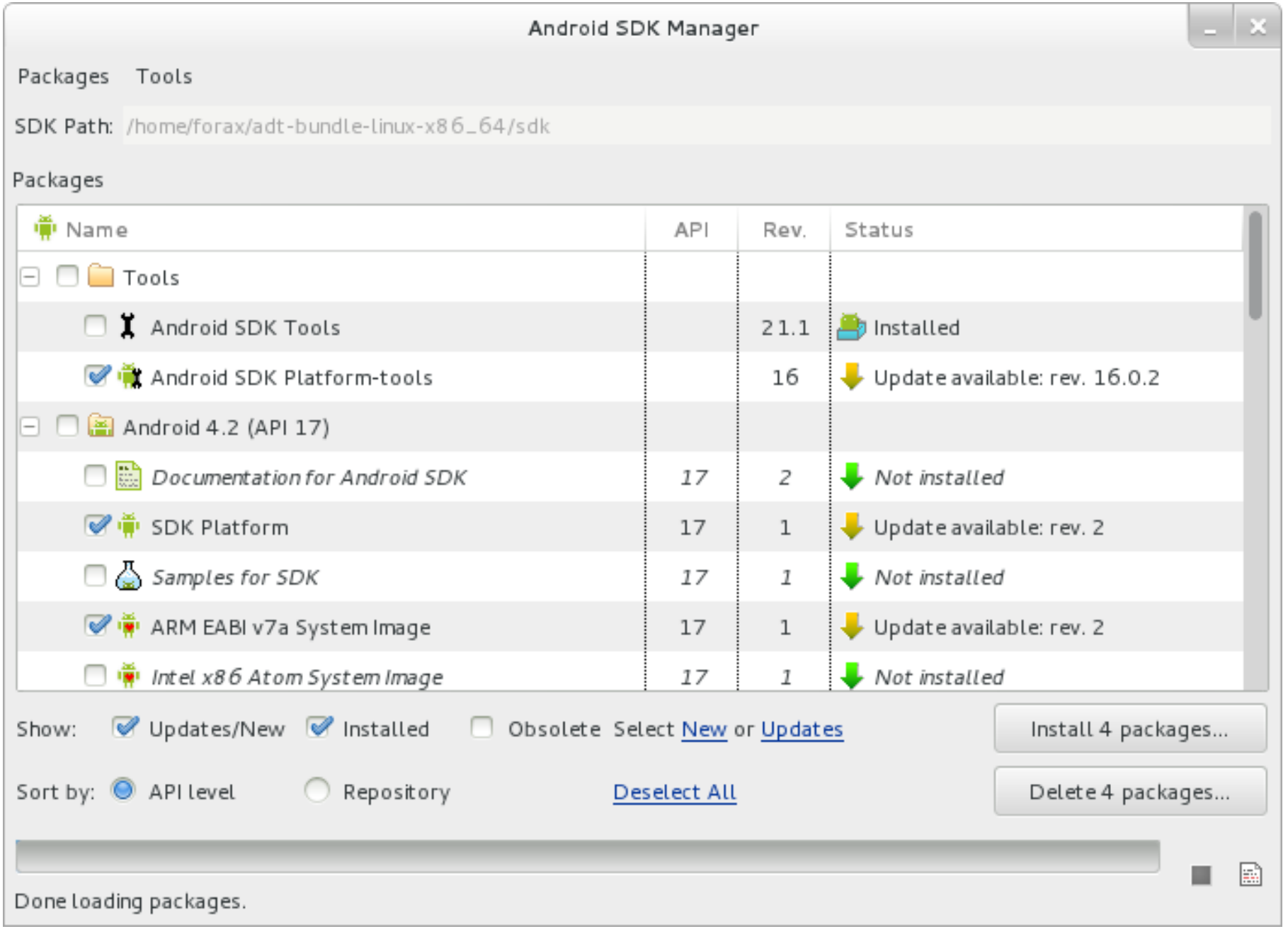



# Android Studio

# SDK Manager (Tools > Android)

- Gère les versions du SDK d'android installés
- Gère les images des virtuals devices
  - Pour ARM, pour INTEL, pour chaque version d'android
- Gère les sources des différents SDK
- Gère les sources des exemples

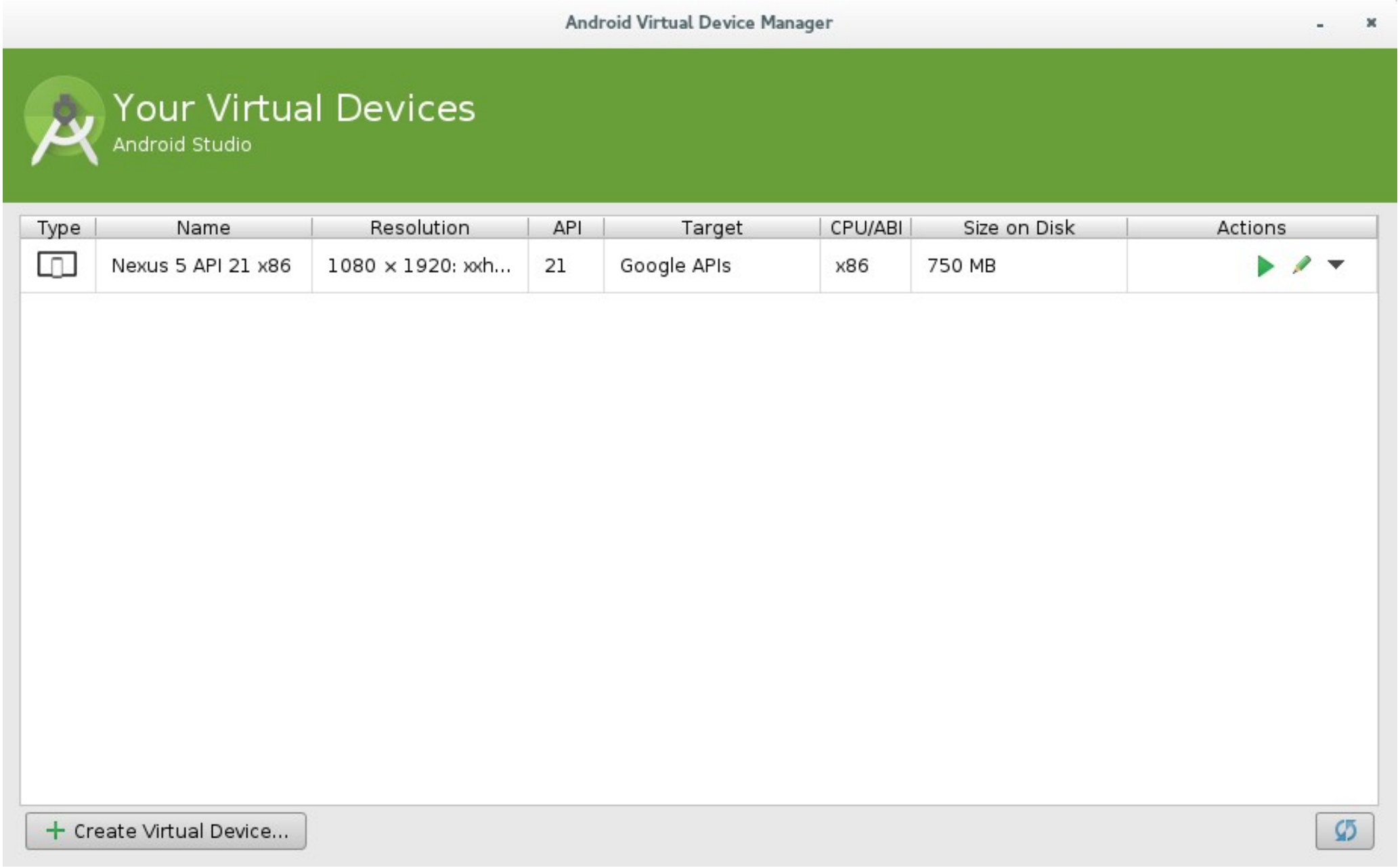
# SDK Manager


Android SDK Manager
Packages
Tools
SDK Path: /home/forax/adt-bundle-linux-x86_64/sdk
Packages
Name
API
Rev.
Status
Tools
Android SDK Tools
21.1
Installed
Android SDK Platform-tools
16
Update available: rev. 16.0.2
Android 4.2 (API 17)
Documentation for Android SDK
17
2
Not installed
SDK Platform
17
1
Update available: rev. 2
Samples for SDK
17
1
Not installed
ARM EABI v7a System Image
17
1
Update available: rev. 2
Intel x86 Atom System Image
17
1
Not installed
Show:
Updates/New
Installed
Obsolete
Select New or Updates
Install 4 packages...
Sort by:
API level
Repository
Deselect All
Delete 4 packages...
Done loading packages.

# Virtual Device Manager

- Permet de créer des configurations de l'émulateur
  - Configure la partie graphique
    - Mode tablette/téléphone, taille de l'écran, etc
  - Configure la partie matérielle (émulé) accessible
  - Configure l'image de l'émulateur à utiliser

# Virtual Device Manager



# Virtual Device en ligne de commande

GUI du Virtual Device Manager : android avd

Nouvelle carte SD vfat :
mksdcard -l carteSD 1024M carteSD.img

Nouvelle machine virtuelle :
android create avd -c carteSD.img -n myAVD --snapshot --target android-17

Lancement de la VM :
emulator @myAVD
Options : -sdcard <file>, -memory <sizeInMBs>, -shell, -logcat <tags>, -tcpdump <file>

Communication avec la VM :

- adb push <source> <dest> : copie un fichier
- adb pull <source><dest> : récupère un fichier
- adb logcat <tags> : affiche les logs courants
- adb shell : ouvre un shell sur la machine
- adb {install <file>, uninstall <package>} : installe ou désinstalle une application
- adb {backup -f <file>, restore <file>} : sauvegarde ou restaure les données utilisateurs
- adb forward <local> <remote> : redirige une socket locale vers une socket de la machine

# APIs de dev

- API Java 6 (pas complète)
  - java.lang, java.util, java.{io,nio}, java.math, java.net, java.text
    - peut compiler en 1.7 avec kitkat
- Package android.* spécifique
- Intégration de JUnit
- Bibliothèques XML et JSON
  - javax.xml, org.w3c.dom, org.xml.sax, org.xmlpull et org.json

# Package android.*

os : accès aux primitives IPC, au gestionnaire d'énergie, d'horloge, aux variables d'environnement, au vibreur...

database : gestion de BDDs privées (implantation SQLite3 fournie)

text : outils pour l'affichage de texte

util : classes utilitaires pour analyse de texte, logging, tableaux creux, caches...

content : gestion de contenus

provider : accès aux ContentProvider de base (CallLog, Contacts, MediaStore...)

app, view, widegets : applications, vues graphiques

webkit : affichage de contenu HTML

media : lecture et enregistrement de données audiovisuelles

graphics : bibliothèque graphique bitmap permettant la manipulation d'images

opengl : API pour le rendu graphique 3D OpenGL

location : API de géolocalisation (par GPS, bornes WiFi et triangulation cellulaire

telephony, bluetooth, net : accès bas-niveau aux interfaces de communication

# Interface Graphique sous Android

android.animation :

interfaces + animations prédéfinies

android.app :

interfaces de base des applications

android.view :

interfaces pour les vues graphiques

android.widget :

implantation de vues utiles

# Structure d'une application Android

# Application Android

Une application android est composée de deux parties

- Une serie de fichier XML décrivant
  - l'application (version de l'API utilisée, activité principale, etc)
  - les ressources (texte, image, **layout**, etc)

- du code Java pour spécifier la partie dynamique de l'application

Les fichiers XML sont plus ou moins obligatoire :(

# Structure d'un projet

- **AndroidManifest.xml** : déclaration des métadonnées du projet (permissions, activités, intent-filter, ...)
- **src/** : sources Java
- **lib/** : jars de bibliothèque
- **res/** : ressources, données statiques utilisées par l'application (chaînes i18n, description des layouts, menus, images, sons...)
- **gen/** : code Java autogénéré (fichier R.java référençant par constantes les ressources)
- **bin/** ou classes/ : classes compilées, fichier apk

# Exemple de manifeste

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="fr.umlv.android.signon"
      android:versionCode="1"
      android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="fr.umlv.android.signon.MainActivity"
            android:label="@string/app_name" >

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>
    </application>
</manifest>
```

Les resources commencent par @...

Oh, une activité

Oh, un intent-filter

# Création d'un projet

Création possible depuis ADT sous Eclipse

Création en ligne de commande :

```
android create project
  target <version> \
  name <your_project_name> \
  path path/to/your/project \
  activity <your_activity_name> \
  package <your_package_namespace>
```

Mise à jour depuis un précédent SDK :

```
android update project \
  name <project_name> \
  target <version> \
  path <path_to_your_project>
```

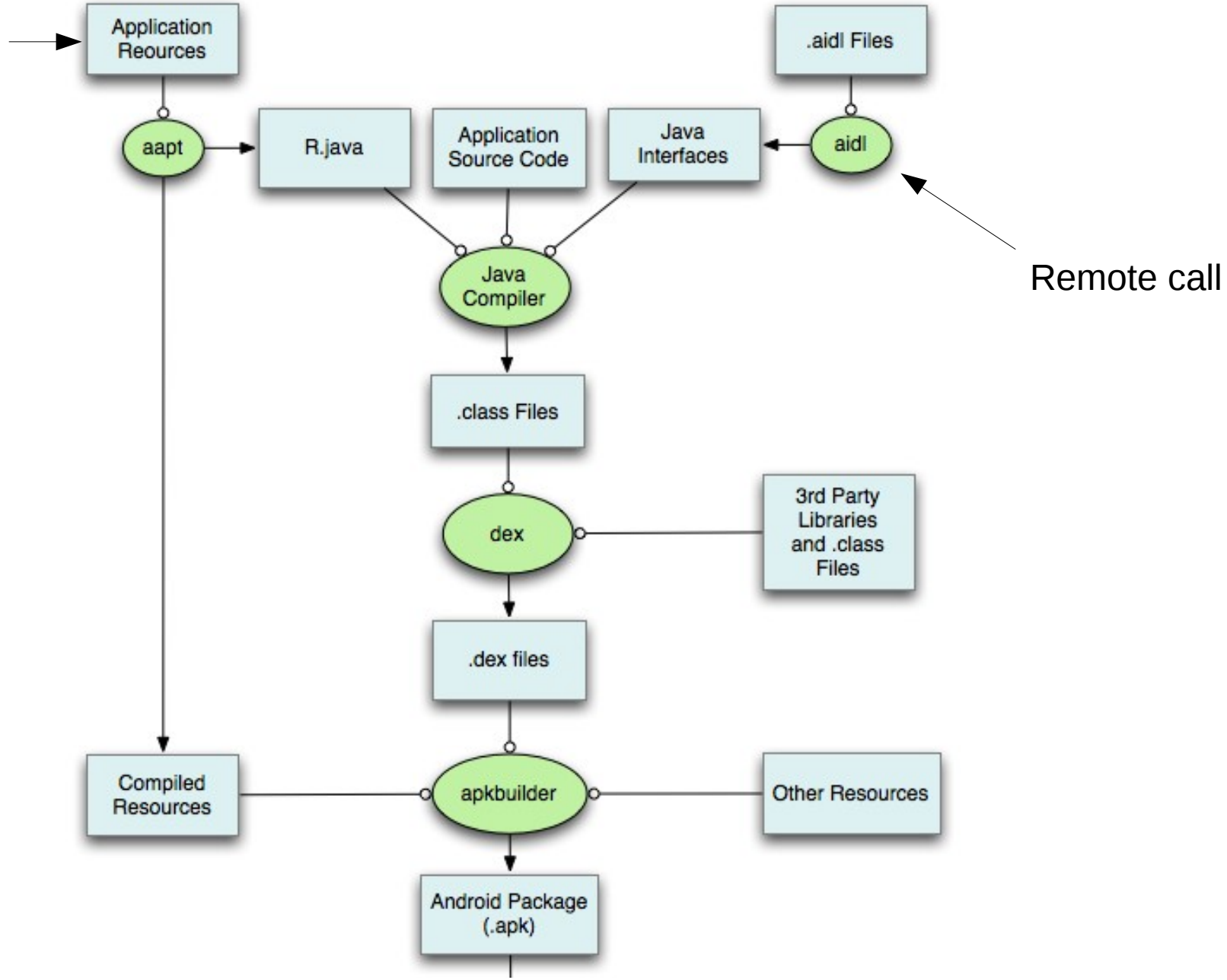
# Compiler un projet

Eclipse ou script Ant auto-généré : ant debug, ant release

- Resources XML vers R.java avec aapt
- Génération du BuildConfig (informations de déverminage pour la compilation)
  - Traditionnel javac
  - Utilisation optionnelle de ProGuard pour optimiser et obfusquer le bytecode
  - Conversion du bytecode Java en bytecode Dalvik (DEX) avec dx
  - Empaquetage et compression des ressources avec aapt
  - Empaquetage final du bytecode et des ressources avec apkbuilder
  - Signature optionnelle de l'apk avec jarsigner (obligatoire pour Google Play)

# Compiler un projet

Les fichiers XML

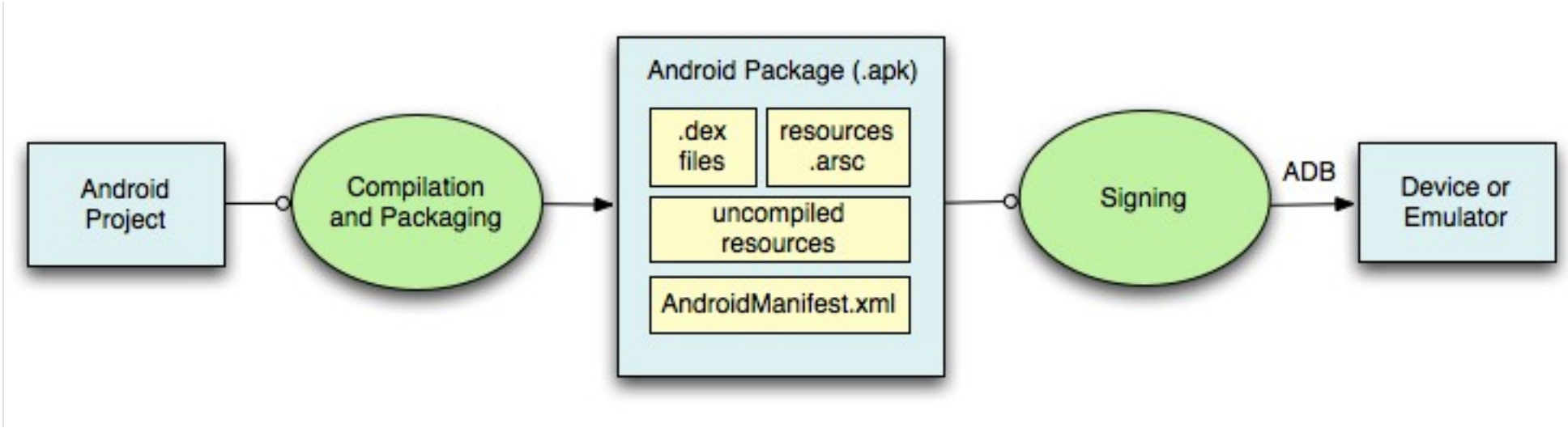


# Tester un projet

- Analyse statique avec lint <projectdir>, avant de compiler
- Compiler le projet : ant {debug, release}
- Lancer un appareil de test sous Android:
  emulator @myAVD
  ou connecter un vrai smartphone en USB
  (vérifier la connexion avec adb devices)
- Installer l'application (préalablement empaquetée dans un apk) :
  adb install HelloWorld.apk
- Lancer l'activité principale :
  adb shell am start -n fr.umlv.helloworld/HelloWorld
- Analyse de l'exécution
  - Surveiller les logs avec logcat (avec tag ActivityThread et de niveau INFO minimum) :
    adb logcat ActivityThread:I
  - Utiliser la GUI Dalvik Debug Monitor (DDMS)

# Cycle de dev

En résumé, le cycle de dévelopement est à peu près celui-là



# Les Activity

# Développer une application android

Contrairement à une application classique, une application android est découpé en écran

- Chaque écran est géré par une Activity
- Une Activity peut avoir plusieurs fragments (version > 3.0)

Le système Android contrôle les Activity, pas le developpeur

- Le code d'une activité ré-agit à des évènements du système Android

# Intent

Comme chaque activité est indépendante et gérée par le système, il existe un mécanisme d'appel inter-activité

- Exécuter une autre activité

```
Intent intent = new Intent(this,OtherActivity.class);
this.startActivity(intent);
```

- Exécuter par rapport à une action

```
Intent intent = new Intent(Intent.ACTION_VIEW,
                           Uri.parse("www.playboy.com"));
this.startActivity(intentImplicit);
```

# Déclarer que l'on sait réagir à une action

On déclare dans le AndroidManifest.xml un intent-filter

Par exemple, ajout d'un intent-filter pour permettre le lancement depuis le launcher:

```
<activity ... >
  <intent-filter ... >
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
  ...
</activity>
```

# Le BackStack

Regroupement logique d'activités en pile (le BackStack) permet la navigation temporelle

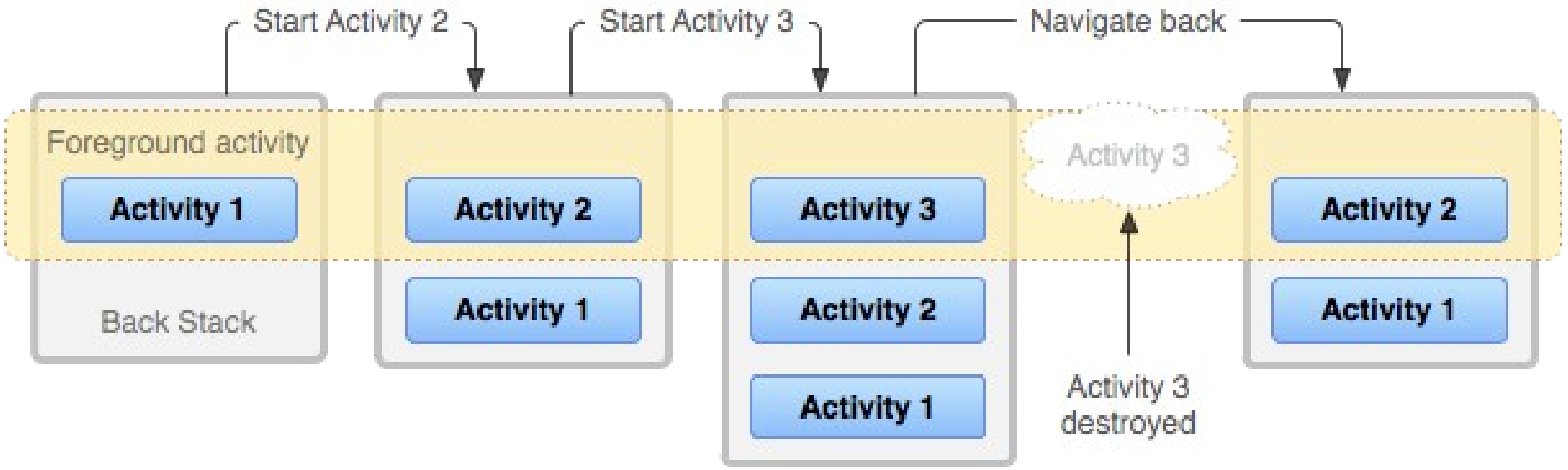
- Création d'une nouvelle Activity : typiquement depuis le launcher
- Coexistence possible de plusieurs activity, mais une unique Activity affichée

Manipulation du BackStack :

- Activité A lance activité B : B est empilé sur A
- Touche "retour" : B est détruite et dépilée
- Touche "home" : la tâche est mise en arrière-plan ; les activités peuvent être détruites en fonction des ressources utilisées

# Le BackStack

Un exemple avec 3 activités



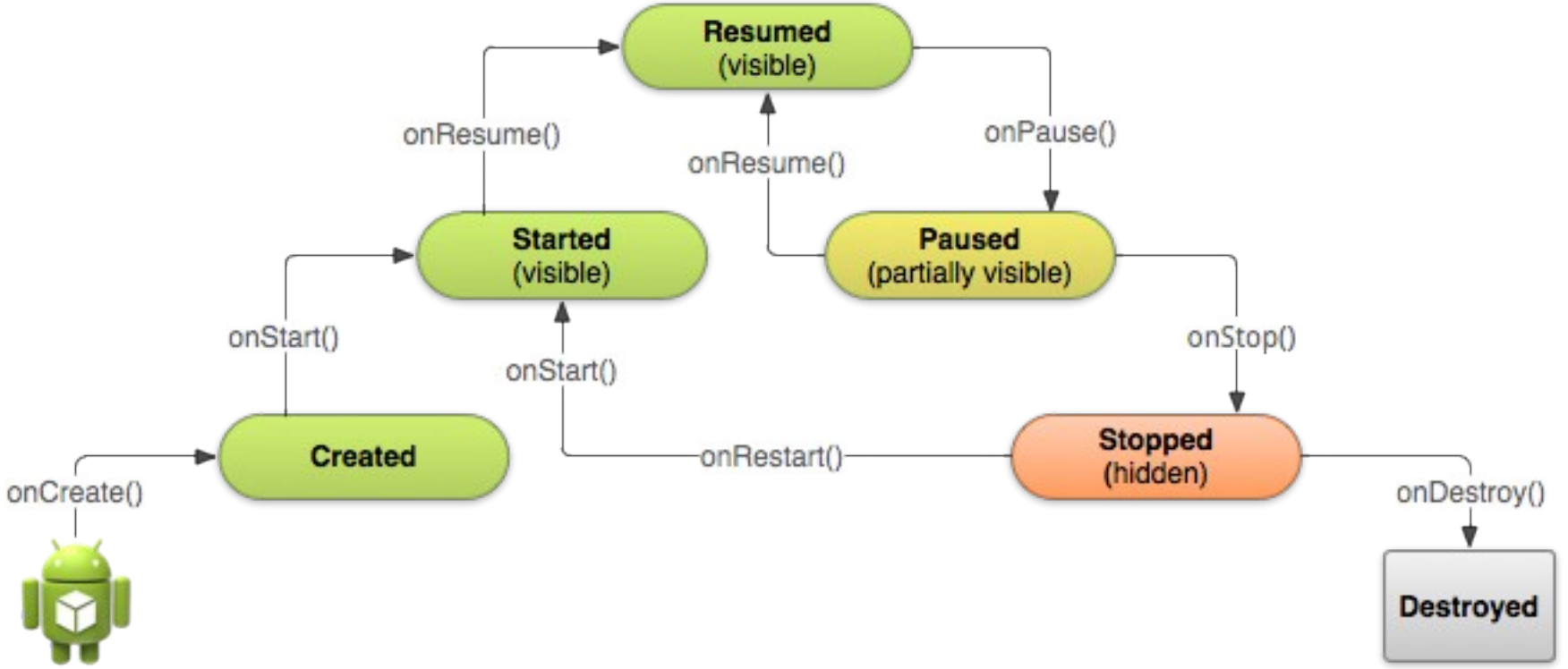
# Cycle de vie d'une activité

- Une seule activité visible à l'écran en avant plan (sommet du BackStack)
- Activités invisibles (autres tâches, fond du BackStack) :
  - RAM : présentes (ou pas si pénurie de mémoire)
  - CPU : ne doivent pas exécuter des threads de calcul (tâche dévolue aux services)
- Passage de l'état visible à invisible :
  - En lançant une nouvelle activité masquant l'activité courante
  - En détruisant l'activité courante (finish(), bouton "retour")

    Envoi d'évènements à chaque changement de statut : gestion par listener onXXX() dans Activity

# Cycle de vie d'une activité

L'activité réagie au demande du système android (qui lui même réagi au demande de l'utilisateur)



# Activity.onXXX()

Les méthodes appelées lors des changements d'état :

- **onStart**() : affichage de l'activité à l'écran ; s'il s'agit d'un redémarrage, onRestart() est appelé avant
- **onResume**() : passage au 1er plan
- **onPause**() : remplacement de l'activité courante par une autre (sauvegarde de champs en cours d'édition, arrêt de threads d'animation)
- **onStop**() : activité rendue invisible, peut être ensuite tuée si pénurie de mémoire
- **onCreate**(Bundle savedInstanceState) : initialisation des structures, (re)chargement des données dans le Bundle
- **onDestroy**() : destruction imminente de l'activité, libération de ressources

On redéfinie les méthodes que l'on veut et

on oublie pas d'appeler en premier **super.onXXX**()

# Il n'y a pas que des Activités !

Une application Android est composée de plusieurs

- **activity** : brique élémentaire pour interface graphique d'interaction avec l'utilisateur

  fragment : élément de GUI réutilisable
- **service** : démon réalisant une tâche en arrière plan
- **provider** : fournisseur de contenu pour gérer des données accessibles à d'autres applications (ex : carnet d'adresse, agenda...)
- **receiver** : Récepteur d'évènements diffusés pour réagir à des évènements systèmes

# Processus et Thread

## Utilisateurs

1 utilisateur unix pour une application

## Processus

- Mapping activity, service, etc. → processus modifiable dans le manifeste (android:process)

  Par défaut, 1 application = 1 processus

- Processus le moins important tuable par le système si pénurie

  Hiérarchie : 1er plan > visible > service > arrière plan > vide

## Threads

- Thread principale crée pour l'affichage graphique
- Création par l'utilisateurs de nouvelles threads :

  new Thread(new Runnable() { ... }).start()

https://developer.android.com/guide/components/processes-and-threads.html

# Thread & interface graphique

Thread principale automatiquement crée pour l'affichage graphique

- Ne pas la bloquer avec de longs calculs ou communications réseau

  sinon **A**pplication **N**ot **R**esponding

- Les autres threads créer par l'utilisateur n'ont pas le droit de modifier l'interface graphique
  (cf cours sur les traitements long)

# Les Ressources

# Resources

Définies dans des fichiers XML (ou binaires) dans res/*

res/**values** : déclaration XML (avec i18n) de string, color, dim, style...

res/**drawable** : fichiers multimédias binaires (images, sons)

res/**layout** : vues graphiques aborescente

res/**anim** : définition d'animations

- Animation d'interpolation (changement de transparence, échelle, angle de rotation...)
- Animation pour séquences d'images : liste des images avec leur durée d'affichage

res/**xml** : pour des fichiers XML divers

res/**raw** : pour d'autres ressources sous la forme de fichiers binaires

# Répertoires resources

Les suffixes d'un répertoire indiquent la version des resources
critères de version utilisés

1. Mobile Country Code et Mobile Network Code (MCC et MNC) :
mcc208-mnc00 (réseau Orange en France), mcc310 (réseau aux USA)...
2. Langue et région : en, fr, fr-rCA...
3. Direction d'agencement : ldltr (direction de gauche à droite), ldrtl (de droite à gauche)
4. Min(largeur, hauteur) : sw<N>dp (N en pixels)
5. Largeur d'écran : w<N>dp
6. Hauteur d'écran : h<N>dp
7. Taille d'écran : parmi smal, normal, large et xlarge
8. Ratio largeur/hauteur : long, notlong
9. Orientation de longueur(peut changer) : port, land
10. Mode d'interface : car, desk, television, appliance
11. Mode nuit : night, notnight
12. Densité de l'écran : ldpi, mdpi, hdpi, xhdpi, nodpi, tvdpi
13. Tactilité : notouch, finger, stylus
14. Disponibilité du clavier : keysexposed, keyshidden, keyssoft
15. Clavier physique : nokeys, qwerty, 12key
16. Touches deat navigation : navexposed, navhidden
17. Méthode primaire de navigation non-tactile : nonav, dpad, trackball, wheel
18. Version de plate-forme : vN (e.g. v17 pour Android 4.2)

Inutile d'apprendre
cette liste par coeur !

# Exemple, internationalization

- Valeur par défaut:

  dans res/**values**/string.xml

  ```
  <resources>
    <string name="helloworld">Hello World</string>
  </resources>
  ```

- Valeur pour le francais:

  dans res/**values-fr**/string.xml

  ```
  <resources>
    <string name="helloworld">Bonjour monde</string>
  </resources>
  ```

# Référence aux resources

- Une ressource dans un fichier XML d'une autre ressource:
  @[paquetage:]type/nomDeLaRessource
  par exemple @**string/helloworld**
- Ressources **système** en utilisant le paquetage android
  (par ex. @**android**:color/blue)
- Dans un fichier source Java, par une constante (int) générée dans le fichier R.java, par ex. **R**.**string**.**helloworld**
- Utilisation de Resources context.getResources()
  (Activity hérite de Context) pour obtenir un getter de ressources
- Quelques exemples :
  R.id.loginButton, getString(R.string.helloworld),
  getColor(R.color.my_nice_color), getLayout(R.layout.activity_layout),
  getDimension(R.dimen.world_width),
  getDrawable(R.drawable.card_picture),
  openRawResource(R.raw.bindata) (retourne un InputStream)

# Les composants graphiques

# Android.widget.View

Classe de base de tous les composants graphiques d'Android

regroupe l'affichage ainsi que la gestion des évènements liés à la zone d'affichage en un seul objet

# ViewGroup

Il existe une View particulière qui permet de contenir elle même d'autre View

design pattern composite

ViewGroup : conteneur des vues enfants (arbre de vues) + placement des enfants

# Les layouts

Vues (LinearLayout, RelativeLayout, etc.) héritant de ViewGroup qui gére le placement de vues filles à l'intérieur du ViewGroup

Chaque layout permet d'associer à une View (fille du Layout) un ensemble de contraintes de placement

Arbre de vues statique doit être définie en XML dans le répertoire ressource layout

# res/**layout**/activity_x.xml

Fichier XML décrivant l'interface graphique d'une activité

```
<RelativeLayout
   xmlns:android=”http://schemas.android.com/apk/res/android“
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin">

  <EditText
     android:id="@+id/name"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content" />
</RelativeLayout>
```

génère moi un id SVP

# View en Java

id calculé à partir d'un nom dans R.java

Depuis une activité :

- Activity.setContentView(int)

  demande l'affichage de la hierarchie à partir d'un layout/foo.xml
- View context.findViewById(int)

  demande d'un composant **après** création hiérarchie

```
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    EditText editText = (EditText)findByView(R.layout.name);
    ...
}
```

# View basique

## Éléments de formulaire

- TextView : affiche une chaîne
- EditText : permet la saisie d'une chaîne (propriété inputType pour le type d'entrée attendu)
- Button : bouton cliquable, variante de type interrupteur avec ToggleButton
- CheckBox : case à cocher
- RadioButton : bouton radio regroupable dans un RadioGroup
- CheckedTextView : chaîne cochable (implante Checkable)
- ProgressBar : barre de progression (horizontale, circulaire), variante avec étoiles de notation avec RatingBar
- SeekBar : barre de réglage
- SearchView : champ de recherche avec proposition de suggestions

## Éléments multimédias

- ImageView : affichage d'une ressource image
- ImageButton : bouton avec image
- VideoView : affichage contrôlable de vidéo

# EditText

Champs texte editable par l'utilisateur

attributs XML les + fréquents:

- android:id, identifiant pour un findViewById dans l'Activity
- android:ems, taille en caractère (adapté à la fonte)
- android:text, texte
- android:hint, conseil qui sera affiché si pas de texte
- android:inputTyp, type valeur attendu,
  textPassword, textEmailAddress, date, time, phone

exemple,

```
<EditText
  android:id="@+id/password"
  android:ems="16"
  android:hint="password"
  android:inputType="textPassword"/>
```

# WebView

- View permettant d'afficher du HTML en utilisant le moteur de rendu Webkit
  (utilisé par ~~Chromium~~, Safari, Opera)
- Configurable par WebView.getSettings()
  - setJavaScriptEnabled() pour activer/désactivé le JavaScript
  - setPluginState(PluginState.OFF) pour désactiver les plugins
  - setAllowContentAccess(false) pour désactiver le chargement d'URL
  - setAllowFileAccess(false) pour l'accès au fichiers locaux
- Utilisation
  - loadData(String data, String mimeType, String encoding) affiche le contenu
  - loadUrl(String url) poaur charger le contenu à une URL spécifique

# WebView

- Interception des clics sur les liens hypertextes en redéfinissant
  - shouldOverrideUrlLoading(WebView, String Url)
- Possibilité de communication d'un objet Java à un script JavaScript
  - addJavaScriptInterface(Object object, String name)

# Les layouts de base

## LinearLayout

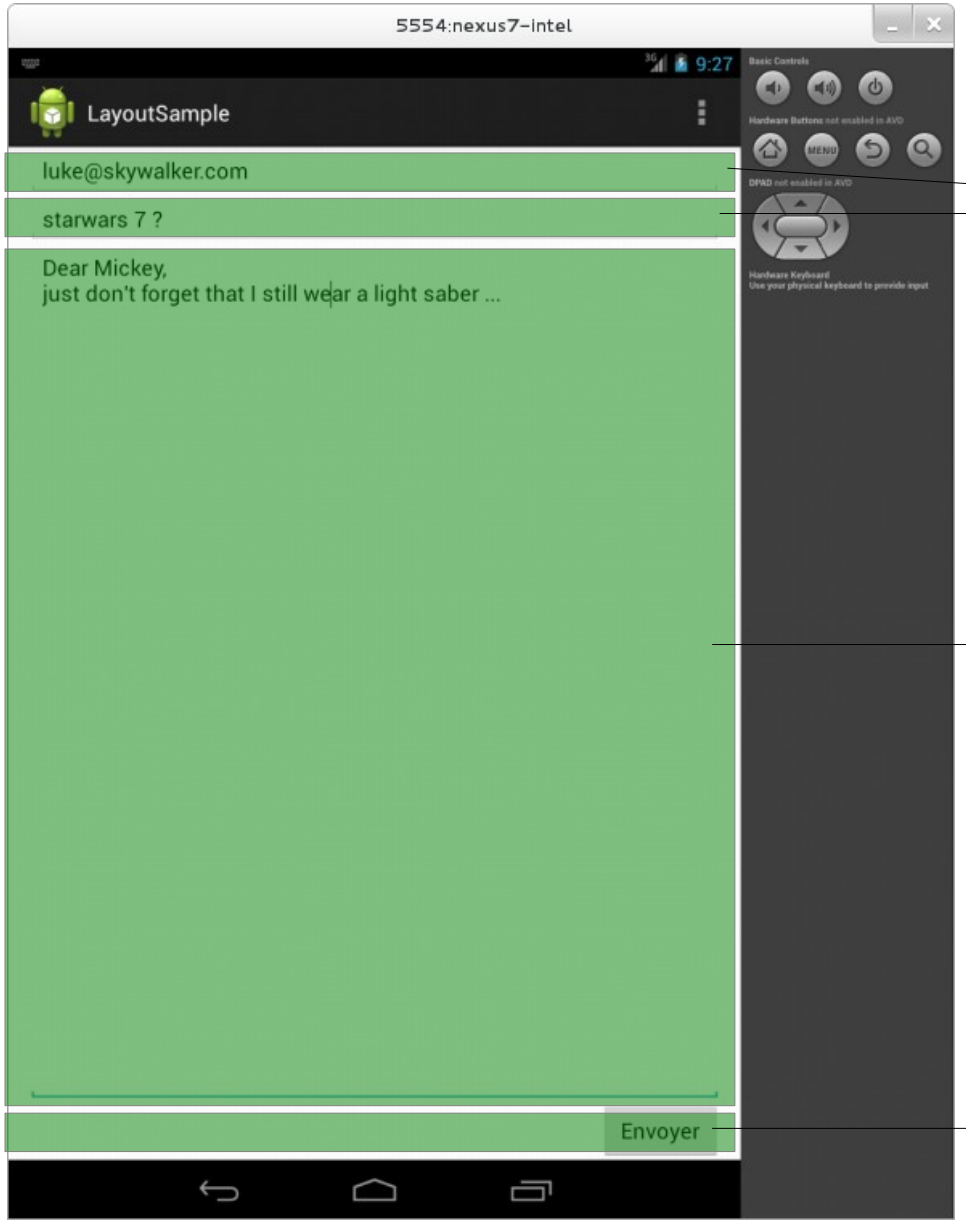
place les composants verticalement/horizontalement les uns derrières les autres

## RelativeLayout

place les composants relativement les uns par rapport aux autres

il y a d'autres layouts (cf plus tard)

# LinearLayout




layout_width = fill_parent
layout_height = wrap_content

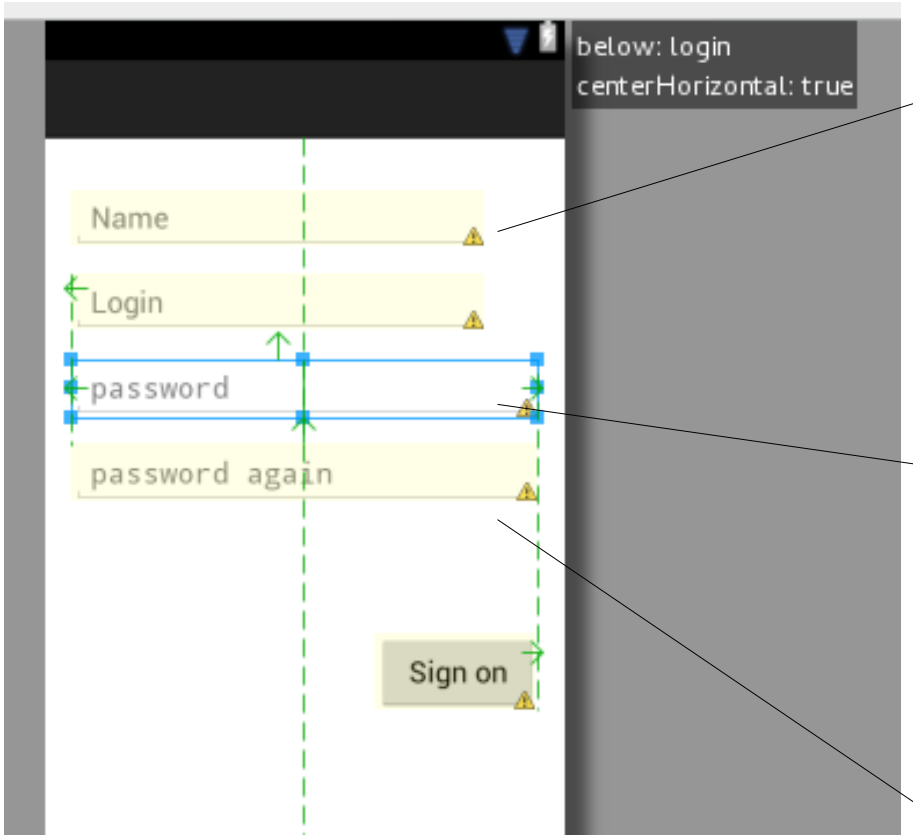
layout_width = fill_parent
layout_height = 0dp
layout_weight = 1
layout_gravity = top

layout_width = 100dp
layout_height = wrap_content
layout_gravity = right

# Contraintes du LinearLayout

- width/height: controle l'occupation de la case
  - fill_parent: prend la taille du parent
  - wrap_content: prend la taille du contenu
- weight: poid pour le dimensionnement
- gravity: placement du contenu dans sa case
  - left, top, bottom, right

# RelativeLayout




layout_width = wrap_content
layout_height = wrap_content
layout_below = @+id/name
layout_marginTop = 16dp
layout_alignLeft = @+id/passwd

layout_width = wrap_content
layout_height = wrap_content
layout_below = @+id/login
layout_marginTop = 16dp
layout_centerHorizontal = true

layout_width = 100dp
layout_height = wrap_content
layout_below = “@+id/passwd2
layout_marginTop = 80dp
layout_alignRight = @+id/passwd

# Contraintes du RelativeLayout

- width/height: controle l'occupation de la case
- below, above: placement relatif à un autre composant (en dessous/au dessus)
- alignLeft, alignRight: alignement relatif entre composants
- marginTop, marginLeft, marginBottom, marginRight: marges autour du composant

# Listener d'évènements

Il existe 3 façons de récupérer un évènement suite à une actions d'un utilisateur

- Redéfinition de la méthode on*XXX*() du composant (pas une bonne idée)
- Enregistrement d'un listener avec setOn*XXX*Listener(*XXX*Listener)
- Dans le XML, propriété android:onClick dans layout/foo_activity.xml

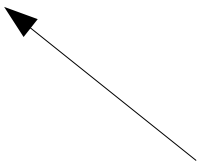
# Listener en code

On s'enregistre pour être prévénu ultérieurement

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button button = (Button)findViewById(R.id.signon);
    button.setOnClickListener(new ButtonOnClickListener(button));
}

static class ButtonOnClickListener implements OnClickListener {
    private final Button button;
    ButtonOnClickListener(Button button) { this.button = button; }
    @Override
    public void onClick(View v) {
        Log.e(MAIN_ACTIVITY, button.getId()+" is clicked");
    }
}
```



android.util.Log, apparait dans le logCat

# Listener en code

On utilise une classe annonyme

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    final Button button = (Button)findViewById(R.id.signon);
    button.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        Log.e(MAIN_ACTIVITY, button.getId()+" is clicked");
      }
   });
}
```

On doit déclarer la variable local final car on est pas en 1.8 :(

# Activité et champ

Ce code est **idiot**, il n'est pas nécessaire  déclarer les composants graphiques en tant que champ de l'activité !

```
public class DemoActivity extends Activity {
    private Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        button = (Button)findViewById(R.id.signon);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Log.e(MAIN_ACTIVITY, button.getId()+" is clicked");
            }
        });
    }
}
```

devrait être dans une variable locale

# Evènements courants

void **OnClickListener**.**onClick**(View)

clic tactile, par trackball ou validation

boolean **OnLongClickListener.onLongClick**(View)

clic long (1s)

void **OnFocusChangeListener**.**onFocusChange**(View v, boolean hasFocus)

gain ou perte de focus

boolean **OnKeyListener**.**onKey**(View v, int keyCode, KeyEvent e)

appui sur une touche matérielle

boolean **OnTouchListener**.**onTouch**(View v, MotionEvent e)

dispatch d'un événement de touché (appelé avant le transfert de l'évènement à la vue enfant concernée). Les gestures peuvent être composées de plusieurs MotionEvent.

Valeur de retour boolean : permet d'indiquer si l'évènement a été consommé, i.e. s'il ne doit plus être communiqué à d'autres listeners (de vues parents)

# Listener en XML

On utilise l'attribut android:onClick, l'appel se fait par réflection

```
<Button
    android:id="@+id/signon"
    ...
    android:onClick="signOnClicked" />


public class MainActivity extends Activity {
  ...
  public void signOnClicked(View view) {
    Button button = (Button)view;
    Log.e(MAIN_ACTIVITY, button.getId()+" is clicked");
  }
}
```

# Interception global d'évènements

Pratique pour débugger

- Activity.dispatch*XXX*Event(*XXX*Event) :
  dispatch de l'évènement de l'activité vers la vue concernée (*XXX*={GenericMotion | Key | KeyShortcut |PopulateAccessiblity |Touch | Trackball})
- ViewGroup.onIntercept*XXX*Event(*XXX*Event) et ViewParent.requestDisallowInterceptXXXEvent(MotionEvent) :
  vol d'évènement par la vue parent (*XXX*={Touch | Hover})