

Cours Android 2014



- Souvent présenté comme l'alternative de Google à l'iPhone
- Système d'exploitation pour terminaux mobiles
- Basé sur Linux
- Open Source (licence Apache)



- Framework applicatif avec réutilisation et remplacement possible des composants
- DVM : Dalvik Virtual Machine (machine virtuelle optimisée pour les périphériques mobiles)
- Navigateur intégré basé sur le moteur WebKit (OpenSource) Librairie 2D dédiée
- Gestion de la 3D basée sur une implémentation d'OpenGL ES 1.0 (avec support de l'accélération matérielle)
- Base de données SQLite
- Gestion des écrans tactiles et du Multitouch

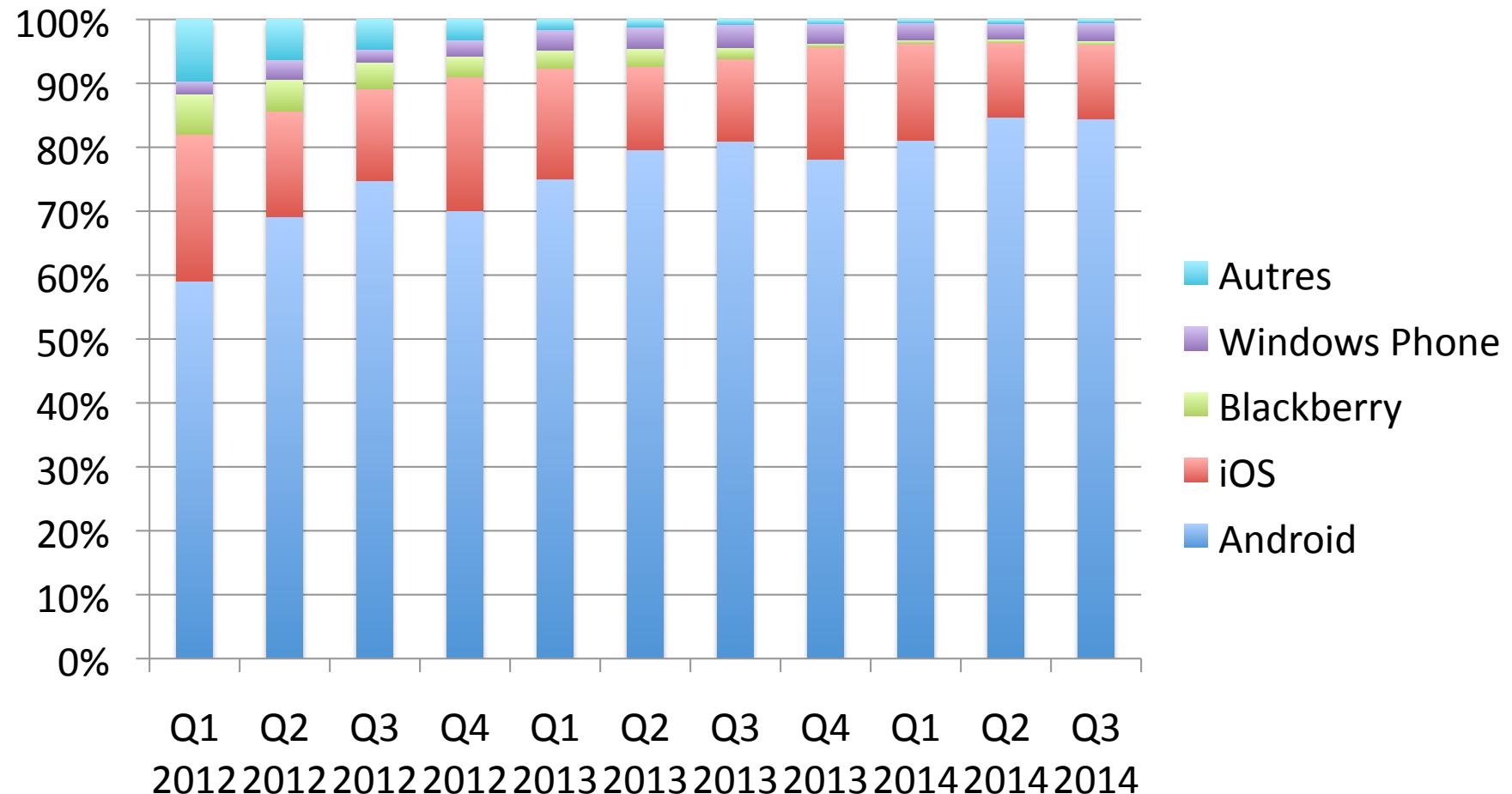


- Multimédia : support de la plupart des formats classiques d'images, de vidéos et audio (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- Téléphonie GSM (selon hardware)
- Bluetooth, EDGE, 3G et WiFi (selon hardware)
- Caméra, GPS, compas et accéléromètre (selon hardware)
- Environnement de développement riche incluant :
 - Un émulateur (avec une interface de contrôle)
 - Des outils de débogage
 - Outils de profilage mémoire et performance
 - Un plugin pour l'IDE Eclipse



- Apple iPhone OS : un des leaders en téléphonie, fermé...
- Windows Phone 7 : En progression avec la chute de Windows mobile 6, fermé...
- Palm : précurseur, en perte de vitesse...
- Blackberry OS : plutôt dédié entreprise, se démocratisant, mais en grosse difficulté.
- Symbian : passage en open source octobre 2009
- Tizen: OS mobile open source développé par Samsung qui viendra remplacer à terme Bada
- Boot to Gecko (B2G) développé par la fondation Mozilla
- Mais la plupart de ses concurrents n'ont pas la flexibilité d'Android qui ne se destine pas uniquement aux téléphones mobiles ! ex : tablette R-link embarquée dans les véhicule Renault.

Généralités :: Parts de marché mondiales %



Source IDC - via ZDNet.fr/chiffres-clés



Généralités :: L'histoire

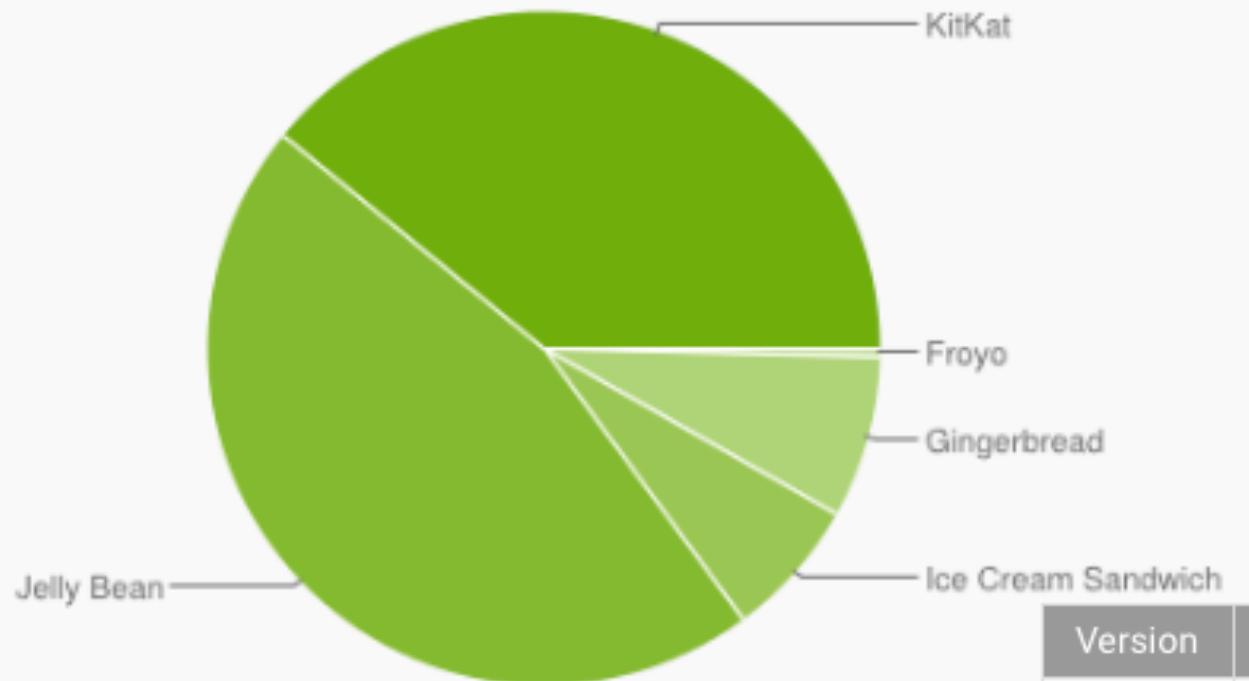
2005	Google achète Android Inc, Le travail sur Davilk commence
2007	Open handset Alliance annonce la sortie prochaine du Software Development Kit.
2008 – 2010	Android devient la plateforme mobile principale
2011 - 2013	Consoles de jeux vidéo, Tablettes, téléviseurs, autoradios, ...
Futur?	Wearable devices: Google Glasses, smartwatches, ...



Généralités :: Les versions

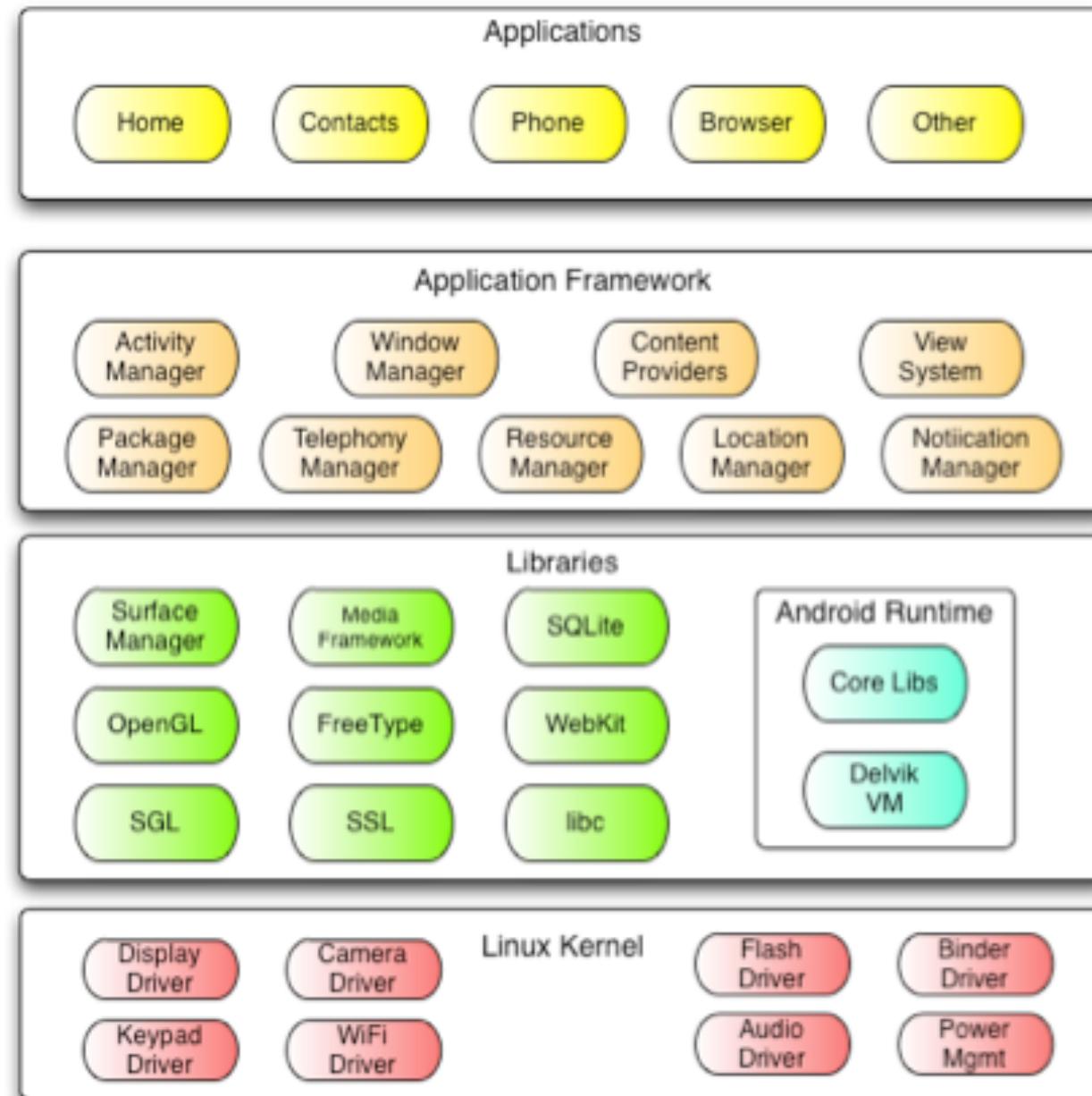
Version	API Level	Nickname
Android 1.0 (09/2008)	1	Android 1.0
Android 1.1 (02/2009)	2	Android 1.1
Android 1.5 (04/2009)	3	Cupcake
Android 1.6 (09/2009)	4	Donut
Android 2.0 (10/2009)	5	Eclair
Android 2.01 (12/2009)	6	Eclair
Android 2.1 (01/2010)	7	Eclair
Android 2.2 (05/2010)	8	Froyo
Android 2.3 – 2.3.2 (12/2010)	9	Gingerbread
Android 2.3.3 -2.3.7 (02/2011)	10	Gingerbread
Android 3.x (02/2011)	11, 12, 13	Honeycomb
Android 4.0 – 4.0.4 (10/2011)	14,15	Ice Cream Sandwich
Android 4.1-4.3 (07/2012)	16, 17, 18	Jelly Bean
Android 4.4 (09/2013)	19	KitKat
Android 4.4w (07/2014)	20	KitKat with wearable extensions
Android 5.0 (11/2014)	21	Lollipop

Généralités :: Distribution des versions



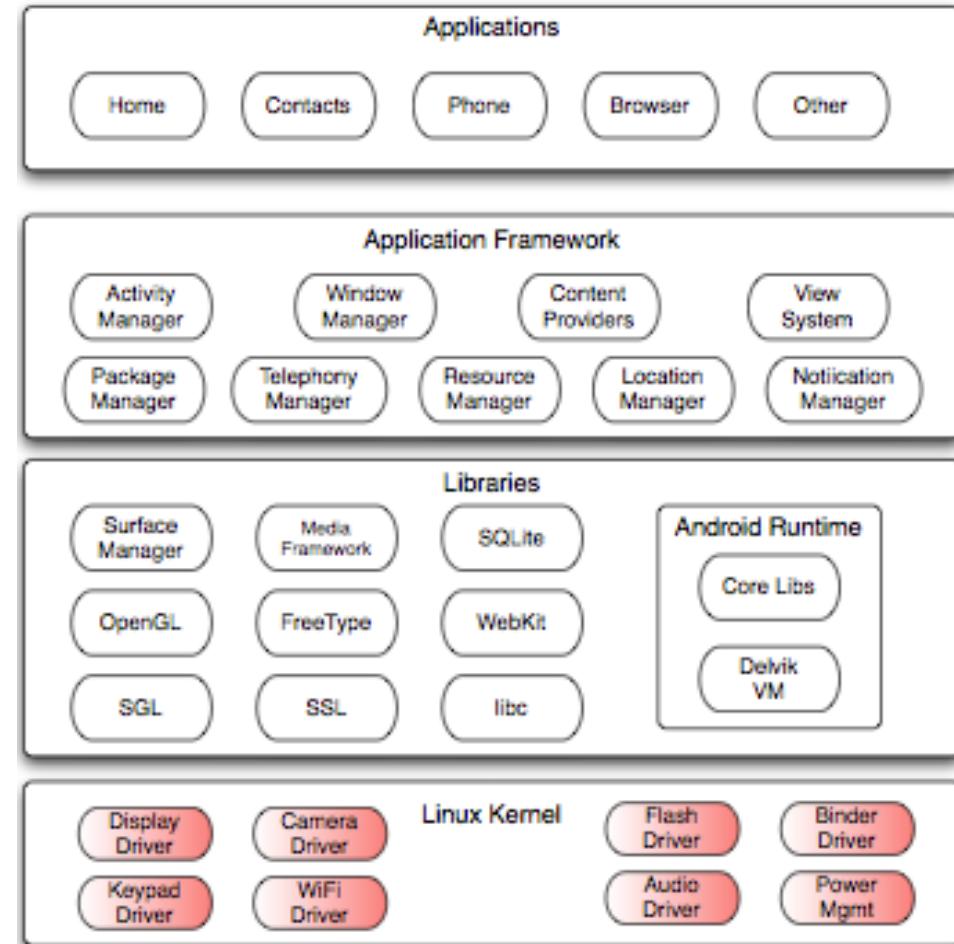
Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	7.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	6.7%
4.1.x	Jelly Bean	16	19.2%
4.2.x		17	20.3%
4.3		18	6.5%
4.4	KitKat	19	39.1%

Généralités :: Architecture logicielle



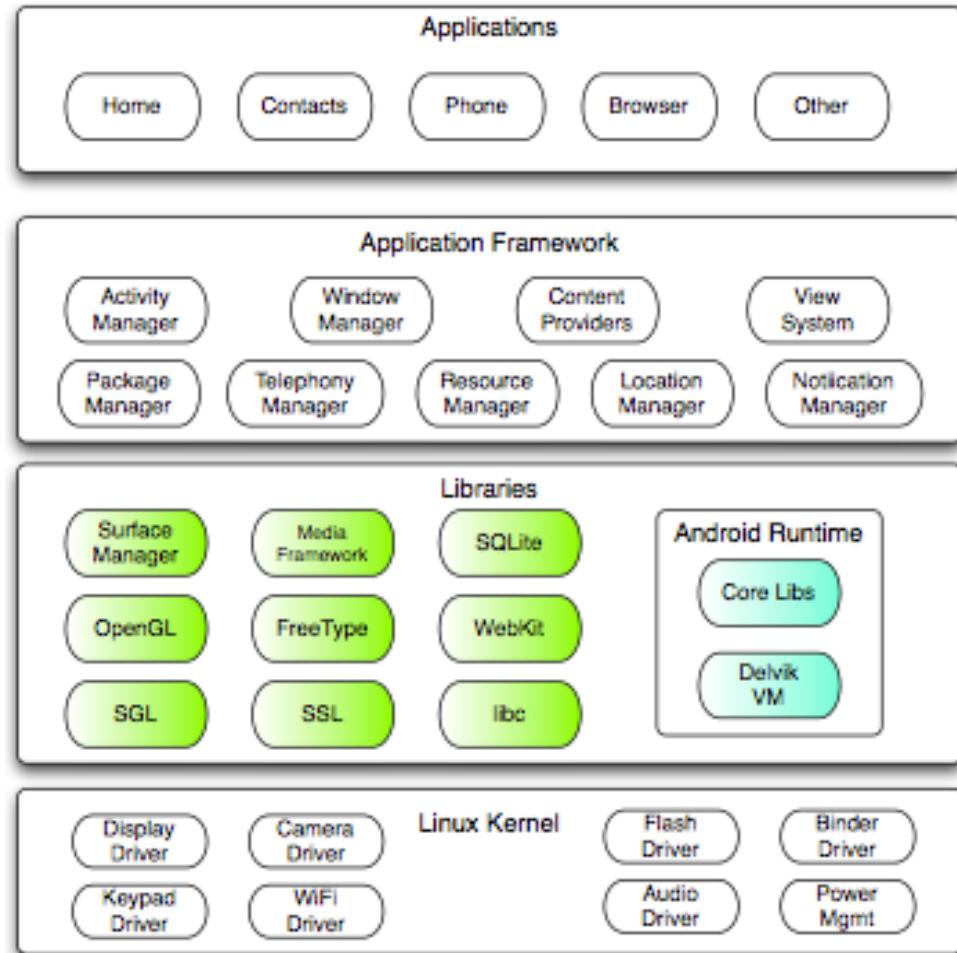
Généralités :: Noyau Linux

- Android fonctionne sur une base Linux
- Linux gère:
 - Couche d'abstraction matérielles
 - Gestion de la mémoire
 - Gestion des processus
 - La couche Réseau
- L'utilisateur n'accède jamais au système Linux
- La commande “ADB Shell” ouvre un shell Linux.





- **Bionic** librairie C optimisée pour Android
- **Webkit** librairie pour le rendu HTML
- **OpenGL** pour les graphiques
- **Codecs Média** qui supportent la majorité des codecs vidéos et Audio du marché
- Base de données **SQLLite**
- Et bien d'autres...

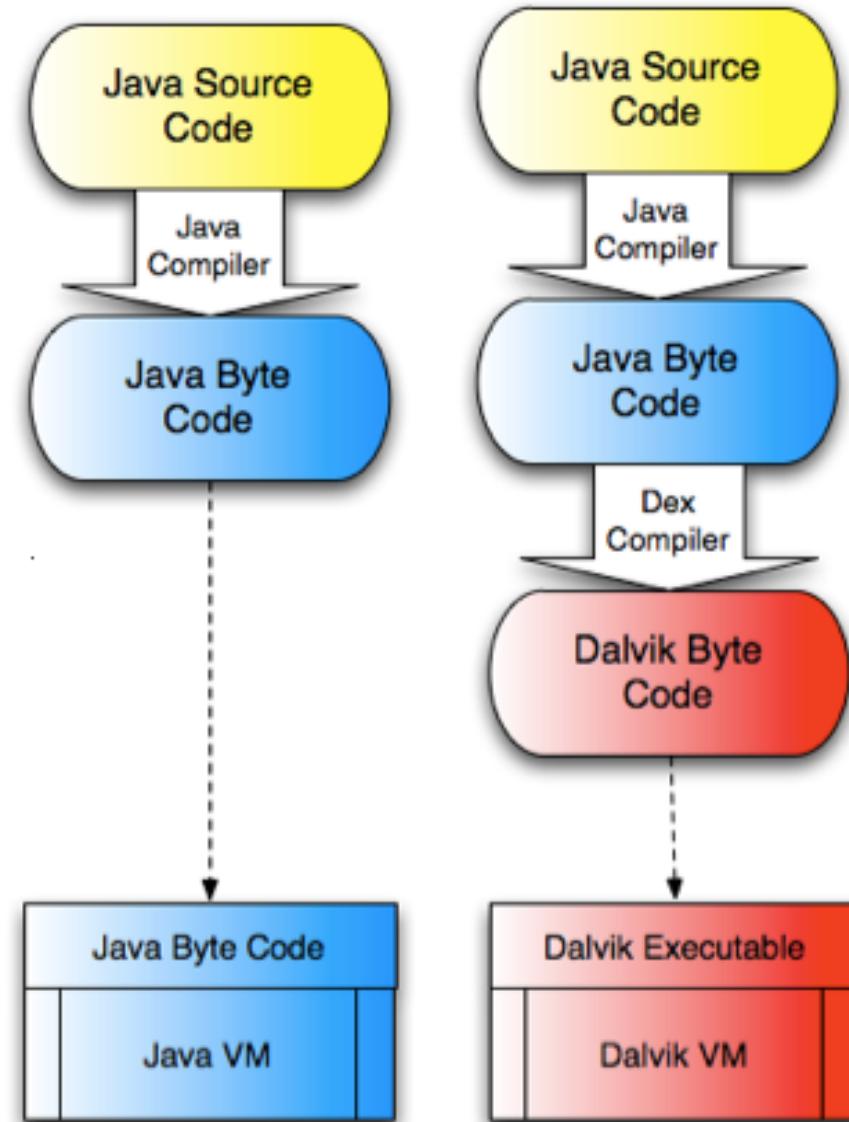




- La **VM Dalvik** est une implémentation Android de la VM Java
- **Optimisée** pour les terminaux mobile:
 - Consommation batterie
 - Capacités CPU
- Principale différences avec la VM
 - Architecture de registre vs Architecture de pile
 - Dalvik exécute des fichiers .dex
 - Implémentation plus efficace et compacte
 - Librairies différentes de celles du SDK

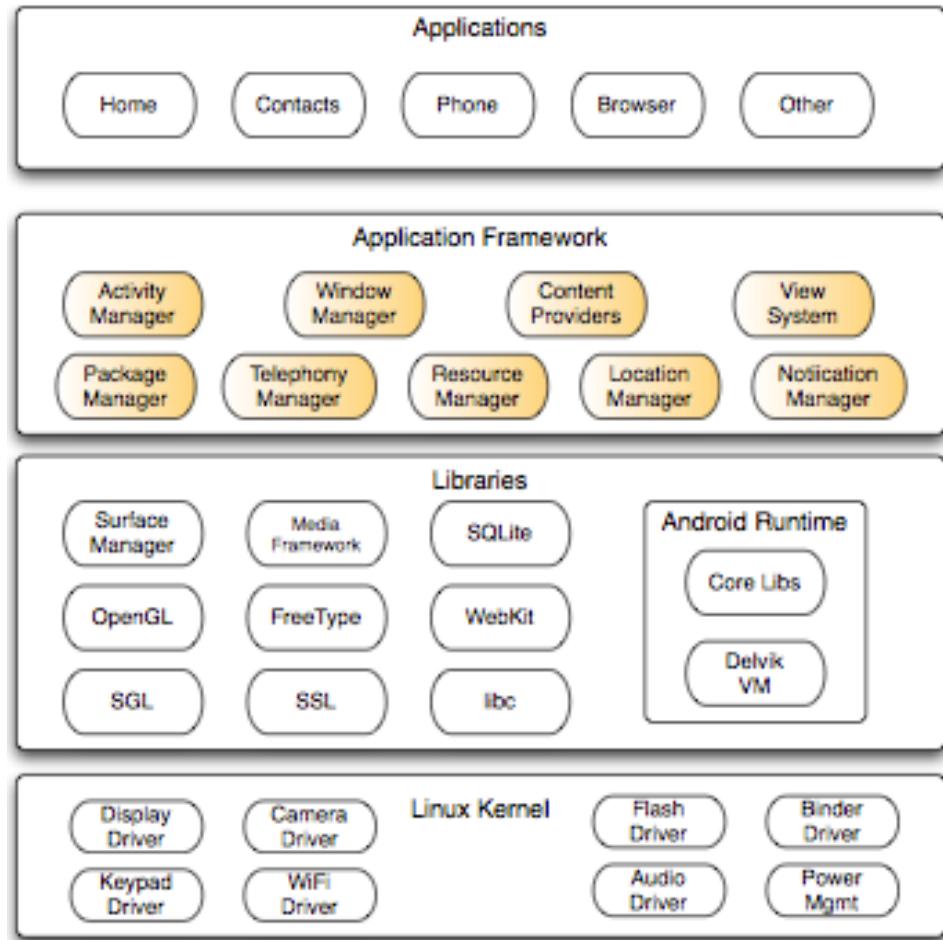


- Android Java = Java SE – AWT/Swing + Android API





- Ensemble de services « riche » contenu dans une api Java Intuitive
- Permet le développement d'applications plus facilement
 - Localisation, web, téléphonie, WIFI, bluetooth, notifications, médias, caméras, etc...





Votre première application

Hello World



- Les outils :
 - SDK Android
 - IDE: Eclipse / Android Studio / Netbeans
 - ADT : Android Development Tools (plugin eclipse)
 - AVD : Android Virtual Device
 - ADB : Android Debug Bridge



Installation de l'environnement de développement :

- JDK Oracle: <http://www.oracle.com/technetwork/java/javase/downloads/index.html> (java -version)
- Android Studio: <https://developer.android.com/sdk/installing/studio.html>
- JDK > 6 : «android studio was unable to find a valid jvm»

```
$ export STUDIO_JDK=/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk  
$ open /Applications/Android\ Studio.app
```



- Installation de l'environnement de développement.
 - Eclipse
<http://www.eclipse.org/downloads/>
 - Sdk Android
<http://developer.android.com/sdk/>
 - Plugin eclipse ADT (Android Development Tools)

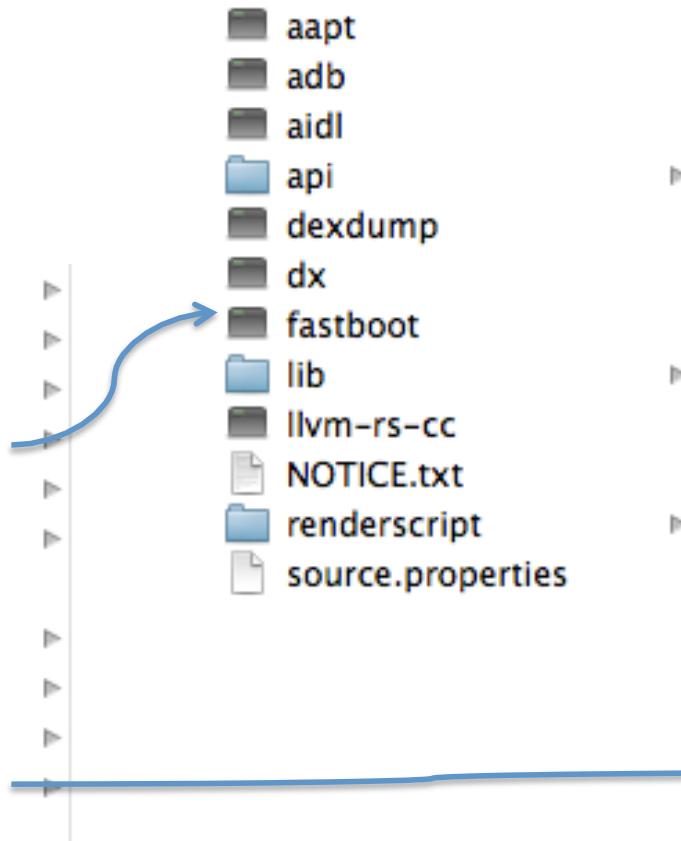




Hello World :: Environnement de développement

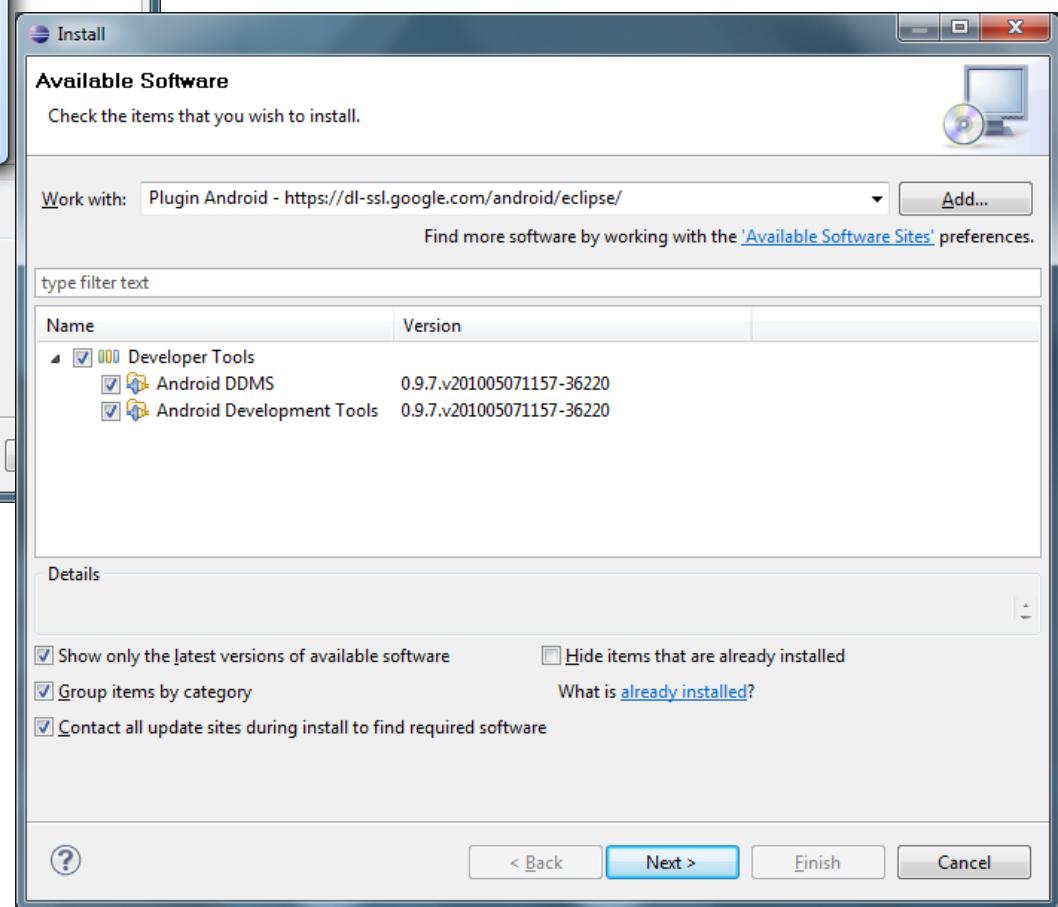
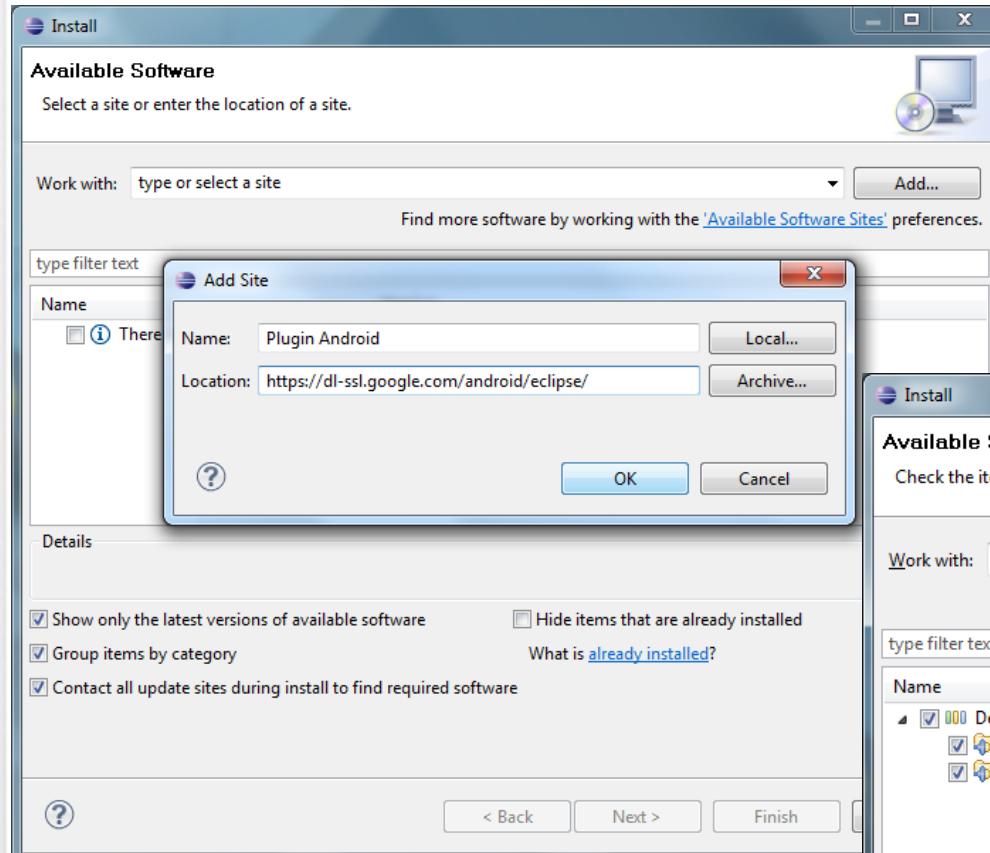
- Le SDK

- add-ons
- docs
- extras
- platform-tools
- platforms
- samples
- SDK Readme.txt
- sources
- system-images
- temp
- tools



- adb_has_moved.txt
- android
- ant
- apkbuilder
- apps
- ddms
- dmtracedump
- draw9patch
- emulator
- emulator-arm
- emulator-mips
- emulator-x86
- emulator64-arm
- emulator64-mips
- emulator64-x86
- etc1tool
- hierarchyviewer
- hprof-conv
- Jet
- jobb
- lib
- lint
- mksdcard
- monitor
- monkeyrunner
- NOTICE.txt
- proguard
- sdcard1.iso
- source.properties
- sqlite3
- support

Environnement de développement :: Eclipse

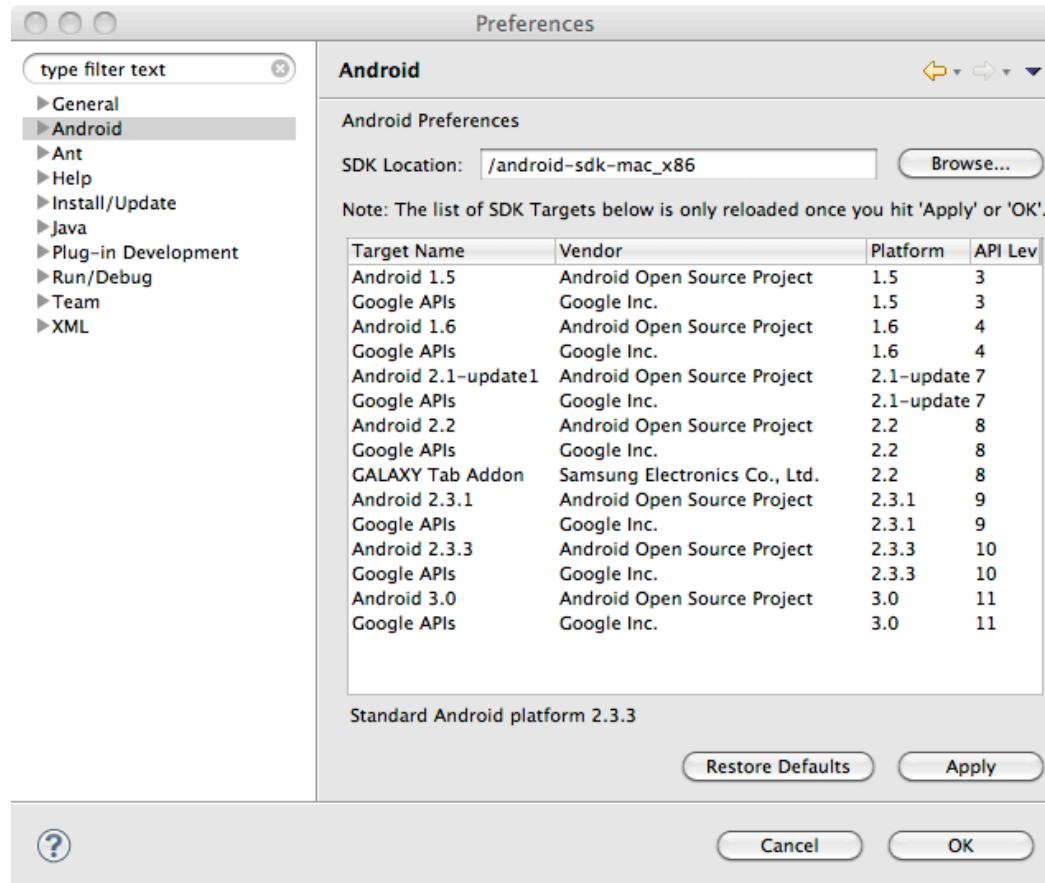


Installation du plugin ADT:

<http://dl-ssl.google.com/android/eclipse>



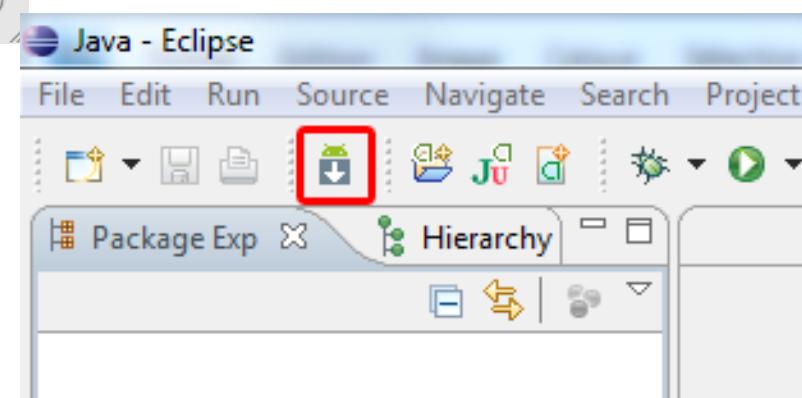
Environnement de développement :: Eclipse



Executer le SDK

Configuration du plugin ADT

- Configurer le « chemin » du SDK





Environnement de développement :: Eclipse

Android SDK and AVD Manager

List of existing Android Virtual Devices located at /Users/patrice/.android/avd

AVD Name	Target Name	Platform	API Level
✓ WVGA800_7	Google APIs (Google Inc.)	2.1-update1	7

New... Edit... Delete...

✓ A valid Android Virtual Device. A repairable A
✗ An Android Virtual Device that failed to load. Click

Edit Android Virtual Device (AVD)

Name: WVGA800_7

Target: Google APIs (Google Inc.) – API Level 7

SD Card:

Snapshot:

Skin:

Hardware:

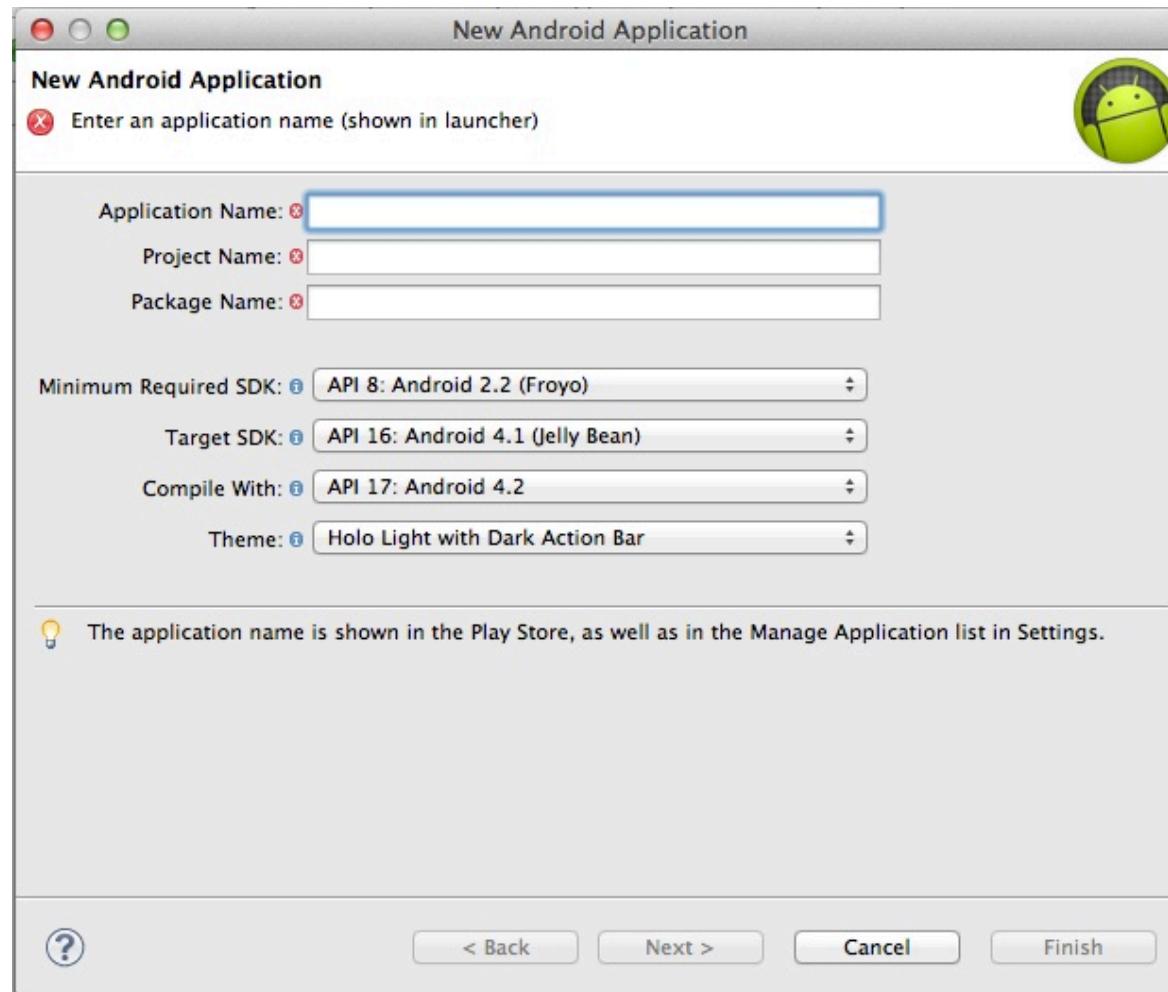
Property	Value
SD Card support	yes
Abstracted LCD density	240
Max VM application heap size	100
Camera support	no

Override the existing AVD with the same name

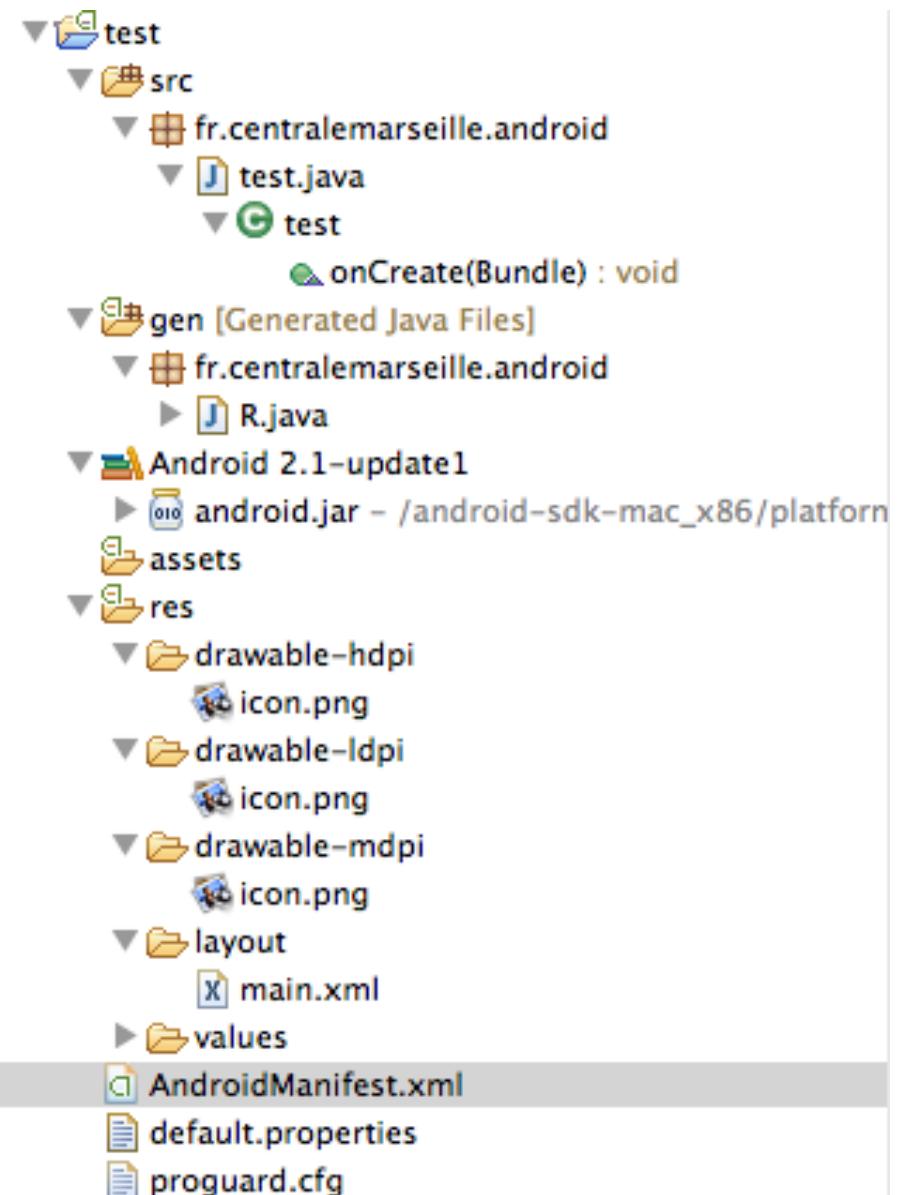
Cancel Edit AVD

Création d'un « Device Virtuel »

- Créer un nouveau projet android



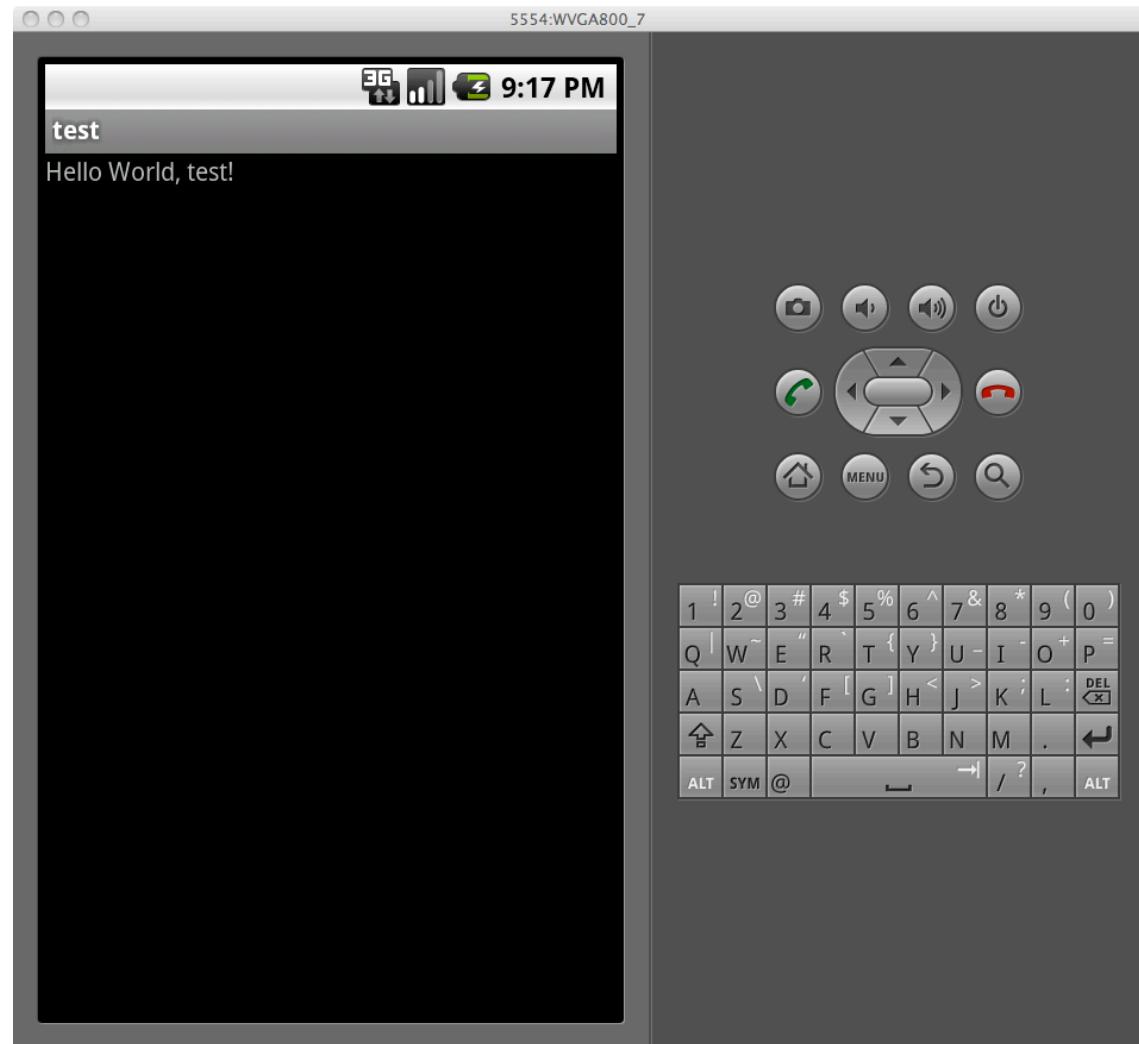
- Organisation des dossiers:
 - Src: sources
 - Gen: code généré
 - Res: ressources
 - Drawable: images
 - Layout: layout
 - Values: constantes
 - Manifest





Exécution de l'application :

- Sur téléphone
- Sur émulateur





- ADB: Android Debug Bridge:
 - Communication avec l'émulateur
 - Communication avec des devices Android
 - Intégration avec l'IDE
 - Utilisable en ligne de commande
 - Ex: adb devices, adb kill-server, adb start-server, etc...



Les applications

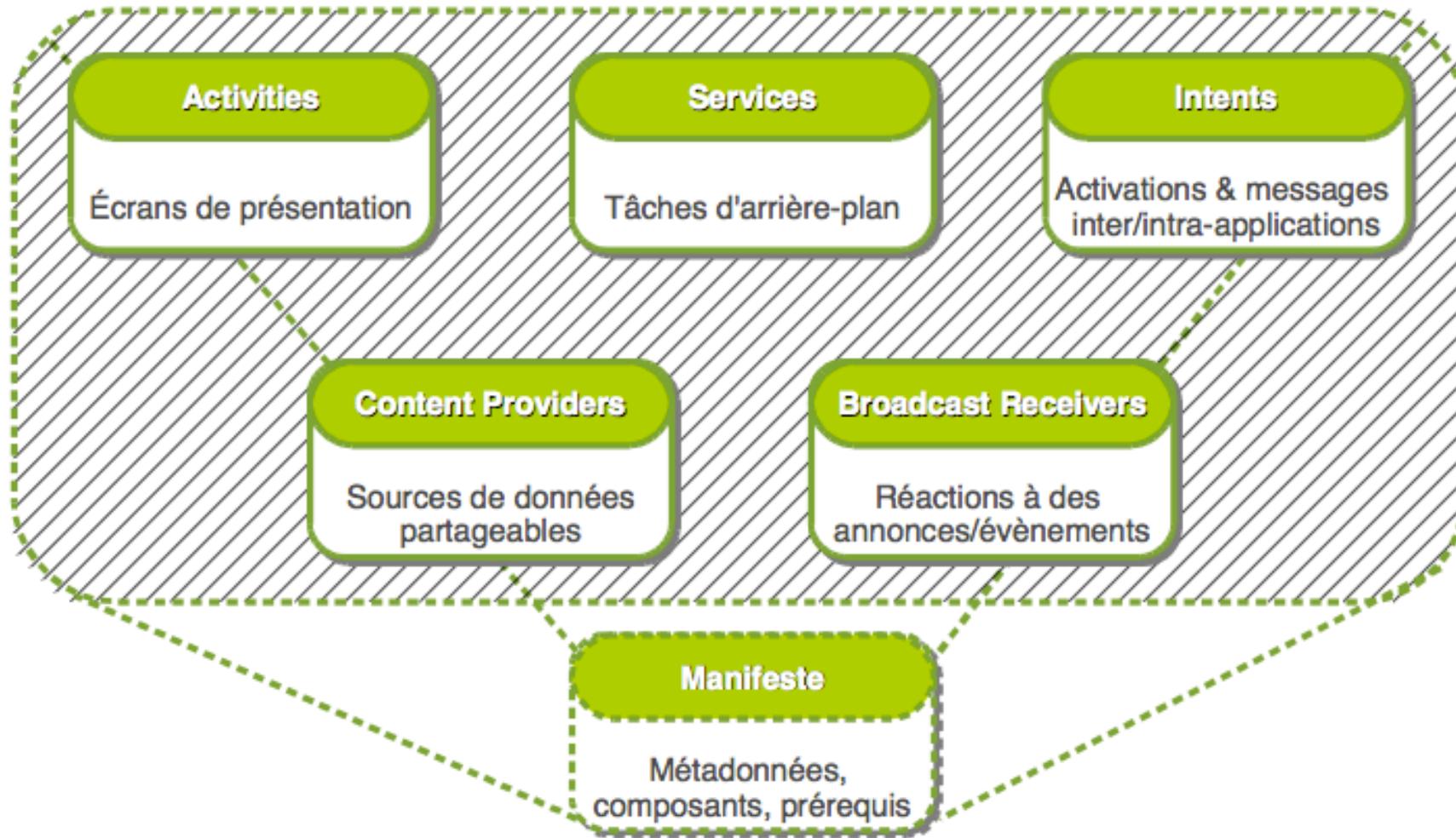


- 2 parties :
 - Les activités : des fenêtres interactives
 - Les services : tâches de fond.
- Les applications tournent dans leurs “SandBoxes”
- Communications entre applications : Les “intent”
 - Intent = intention : formule une demande
- Plusieurs composants peuvent répondre à un “intent”.



- Dernière couche sur Android
- Plusieurs sont intégrées dans le système :
 - Ecran "Home"
 - Gestion des Emails
 - Gestion des SMS/MMS
 - Gestion de la téléphonie
 - Google Maps...
 - Application supplémentaires installables
 - Toutes les applications sont écrites via le même SDK !

Les applications :: Éléments Fondamentaux

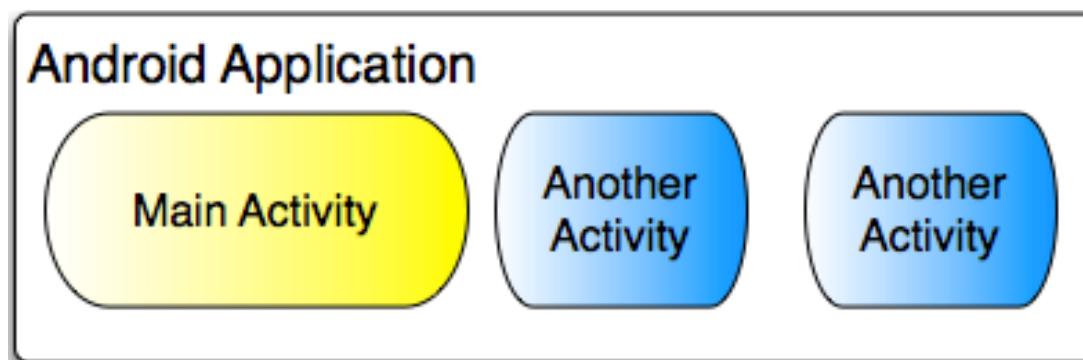




- Une activité ("Activity") = une IHM pour une action utilisateur précise, par ex:
 - Liste d'éléments parmi lesquels l'utilisateur peut choisir
 - Affichage d'une image avec un titre
 - Affichage d'un calendrier pour choisir une date
- Exemple d'une application de SMS :
 - Une activité pour choisir un contact
 - Une autre pour écrire le message
 - Une autre pour afficher un historique d'échanges.
- Chaque activité est indépendante des autres
- Une activité doit hériter de la classe :
`android.app.Activity`



- Une application est donc un ensemble d'activités
- On doit définir quelle est la première activité à exécuter lors du lancement de l'application
- Pour naviguer dans l'application chaque activité doit elle-même lancer l'activité suivante.



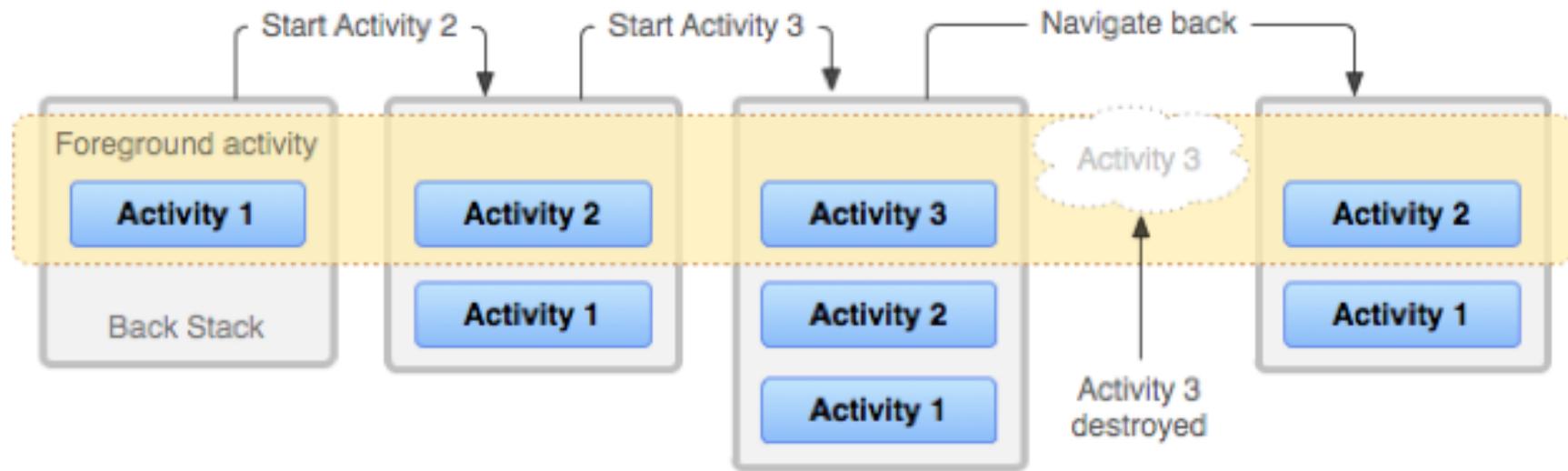


- Chaque activité est assignée à une fenêtre
 - Plein écran
 - Fenêtre flottante
- Une activité peut aussi posséder des sous fenêtres
 - Pop-up ...



- Pour pouvoir être lancée, toute activité doit être préalablement déclarée dans le “Manifest”
 - Une activité est désignée comme activité initiale de l'application
 - Ceci est indiqué dans le fichier manifest
 - Lancer une activité
 - Méthode startActivity(...)
 - Lancer une activité en vue d'obtenir un résultat en retour
 - Méthode startActivityForResult(...)

- Les activités sont empilées / dépilerées
 - Empilée quand une activité démarre
 - Dépilerée (i.e. détruite) quand on presse le bouton 'BACK'



- Une pression sur le bouton 'HOME' ne dépile pas l'activité.
 - Elle passe simplement en arrière plan



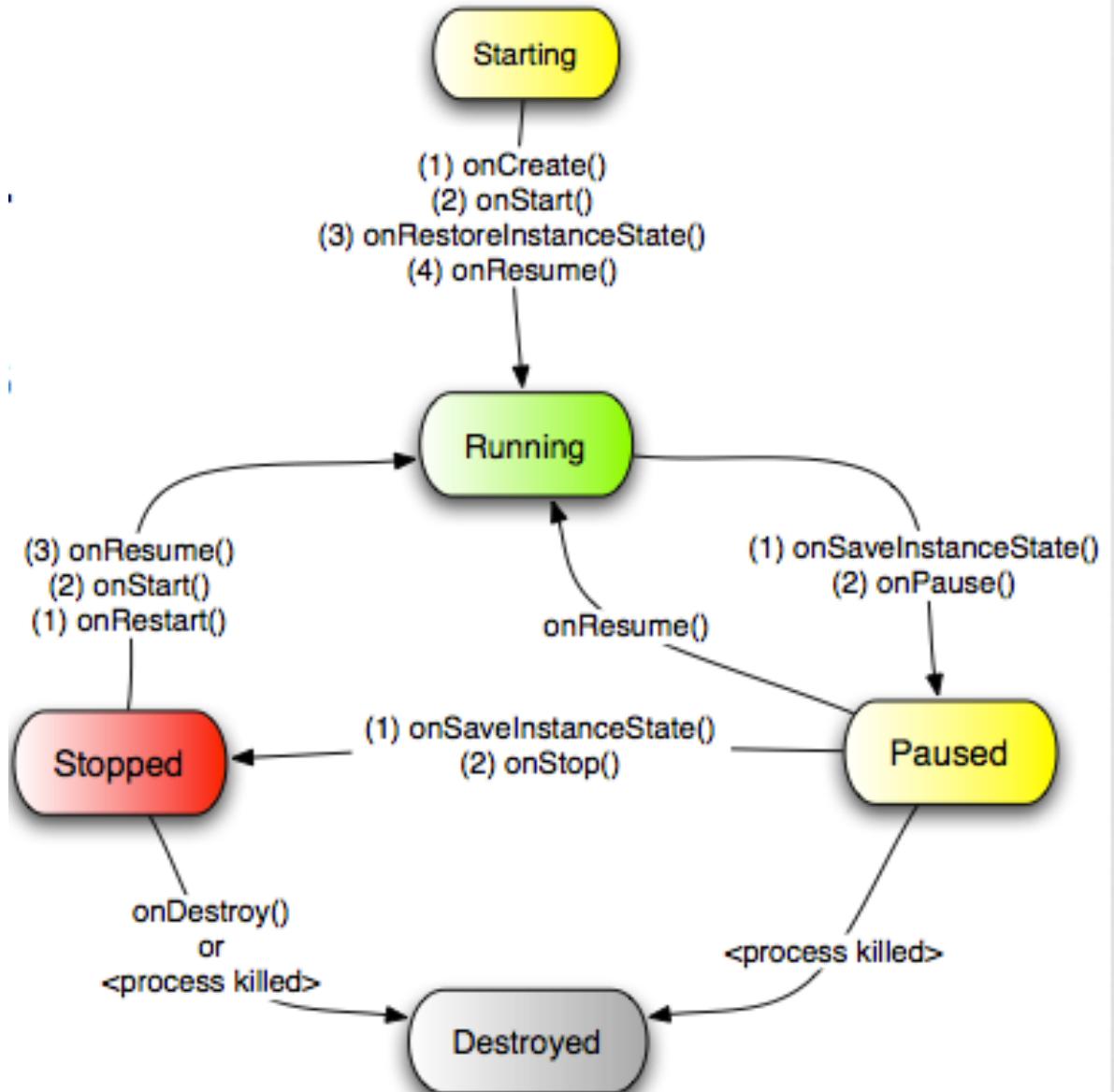
- Une activité possède trois états :
 - Active (running) : Quand l’activité est au premier plan et reçoit les actions utilisateur.
 - Paused : Quand elle est toujours visible mais n’a pas le focus (autre activité transparente par dessus ou activité ne prenant pas tout l’écran)
 - Toujours vivante
 - Mais peut être tuée en cas de ressources très limitées
 - Stopped : Quand elle n’est plus visible
 - Toujours vivante
 - Mais sera tuée dès que des ressources seront nécessaires.



- Le système tue les activités en état “stopped” (ou “paused”) de deux manières :
 - En appelant la méthode `finish()`
 - En tuant le processus tout simplement
- Quand l’activité sera à nouveau demandée :
 - Doit être complètement reconstruite
 - Doit Potentiellement recharger son dernier état

Les applications :: Cycle de Vie

- Une activité est notifiée de ses changements d'état par l'appel à ses méthodes :
 - void `onCreate(Bundle savedInstanceState)`
 - void `onStart()`
 - void `onRestart()`
 - void `onResume()`
 - void `onPause()`
 - void `onStop()`
 - void `onDestroy()`



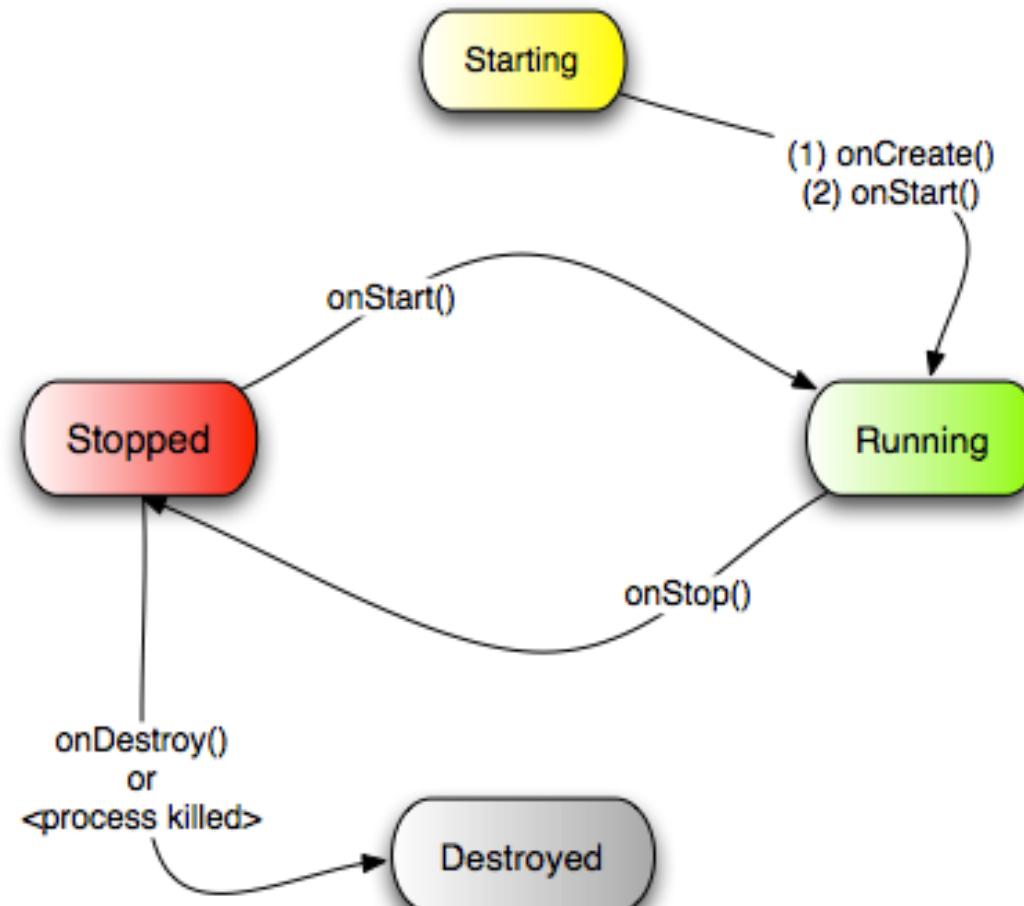


- Afin de sauvegarder le contexte, le système appelle "onSaveInstanceState()" avant de rendre l'application potentiellement tuable (paused...)
- Cet appel fournit un bundle "clé/valeurs" pour que le développeur puisse sauvegarder l'état
- Au prochain appel de "onCreate()" ce bundle sera fourni
- Il est également fourni via un appel à "onRestoreInstanceState()"
- L'appel à la sauvegarde n'est fait qu'en cas de risque de terminaison de l'activité par le système et non si cela vient d'une action utilisateur (back)



- Un service ne possède pas d'interface,
 - Peuvent être démarrés et arrêté,
 - Tourne en arrière plan en continue (ou presque)
- Exemple :
- Lecture de musique
 - Collecte de données
 - Suivi GPS
 - Vérification de mise à jour
 - ...
- Attention, les services s'exécutent sur le Thread principal de l'application: UI Thread

- Cycle de vie





- Étendent la classe "Service"
- Surchagent les méthodes onBind, onCreate, onStartCommand, onDestroy
- Les services doivent être déclarés dans le Manifest
- Pour lancer un service:
 - Intent intent = new Intent(this, UpdaterService.class);
startService(intent);
- Pour arrêter un service:
 - Intent intent = new Intent(this, UpdaterService.class);
stopService(intent);

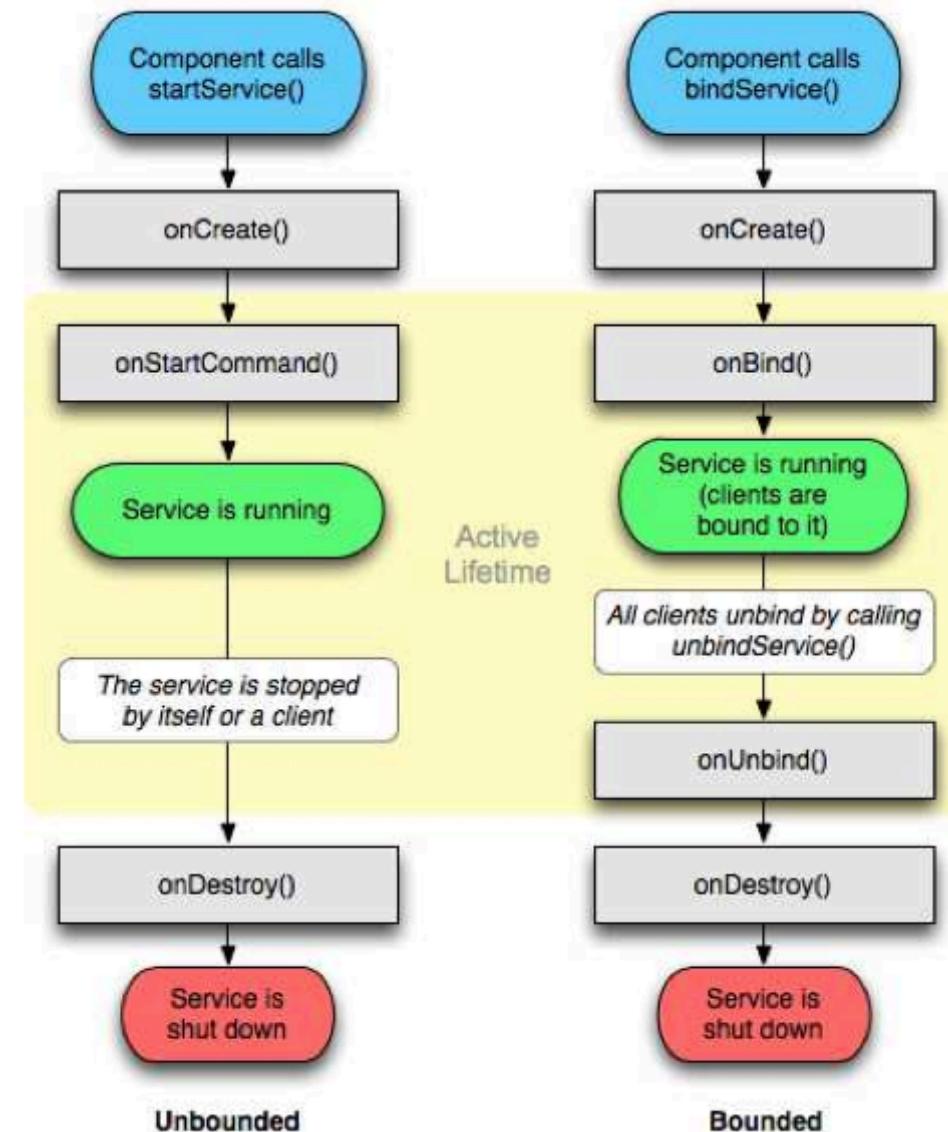


- Mode *Unbounded*
 - Un composant démarre et arrête un traitement en tâche de fond comme il le souhaite
- Opérations
 - startService(...)
 - stopService(...)
- Mode *Bounded*
 - Des composants (appelés "clients") établissent une connexion permanente afin d'interagir avec un service par le biais d'une interface
 - Opérations
 - bindService(...)
 - unbindService(...)



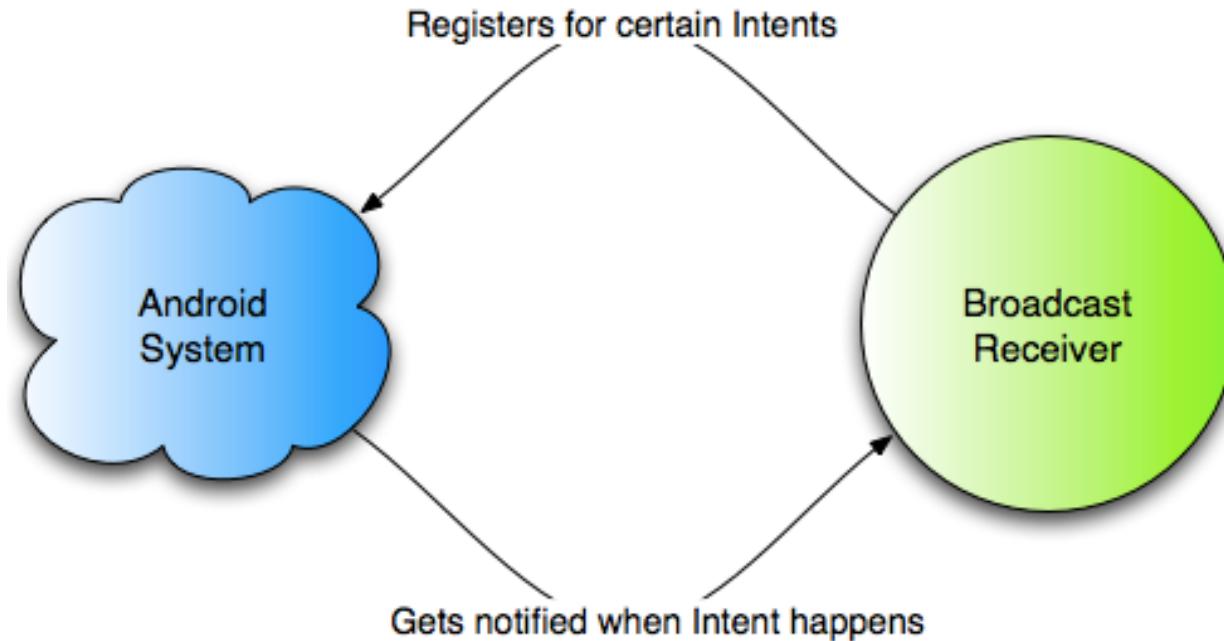
5 méthodes de callback :

- `onCreate()`
- `onStartCommand()`
- `onDestroy()`
- `onBind()`
- `onUnbind()`





Les applications :: Broadcast Receiver



- Les broadcast receiver sont :
 - Des éléments inactifs qui attendent un évènement
 - Il y a différents évènements système :
 - Batterie faible
 - Changement de langue du système
 - L'utilisateur a pris une photo
 - ...
- Il est possible de définir ses propres évènements
- Ils héritent de la classe `android.content.BroadcastReceiver`

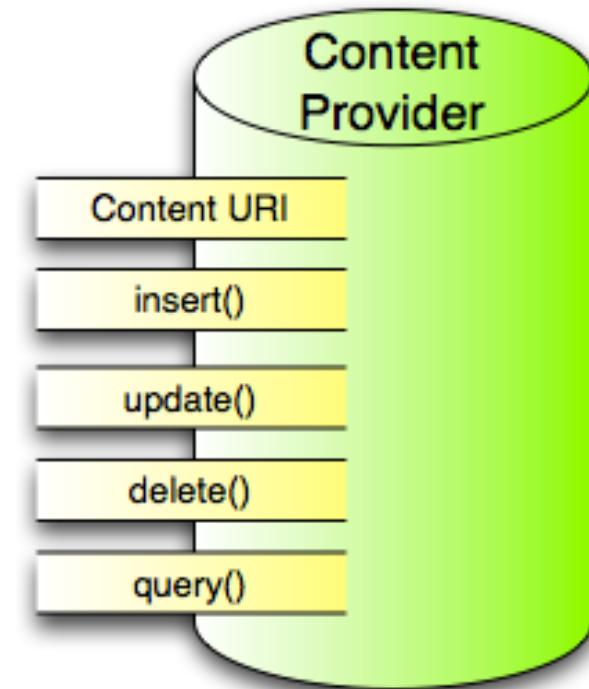


- Une application peut contenir plusieurs receivers : un par événement important
- Les receivers n'ont évidemment pas d'interface. Ils peuvent lancer des activités en cas de besoin
- Un receiver s'abonne/désabonne via le fichier manifest ou programmatiquement
- Ils peuvent également utiliser le NotificationManager pour signaler quelque chose à l'utilisateur (préférable)
 - Icône, vibration, alerte sonore, clignotement diode...



Les applications :: Content provider

- Les content provider permettent de rendre une partie des données d'un application accessible aux autres applications
 - Seul moyen pour un partage de données entre applications
- Les données sont donc exposées par une classe héritant de android.content.ContentProvider
 - Méthodes query(), Insert(), Update(), delete()...
- Les données sont accessibles grâce à l'URI dont le schème dédié est 'content'.
 - Exemple: *content://contacts/people/125*

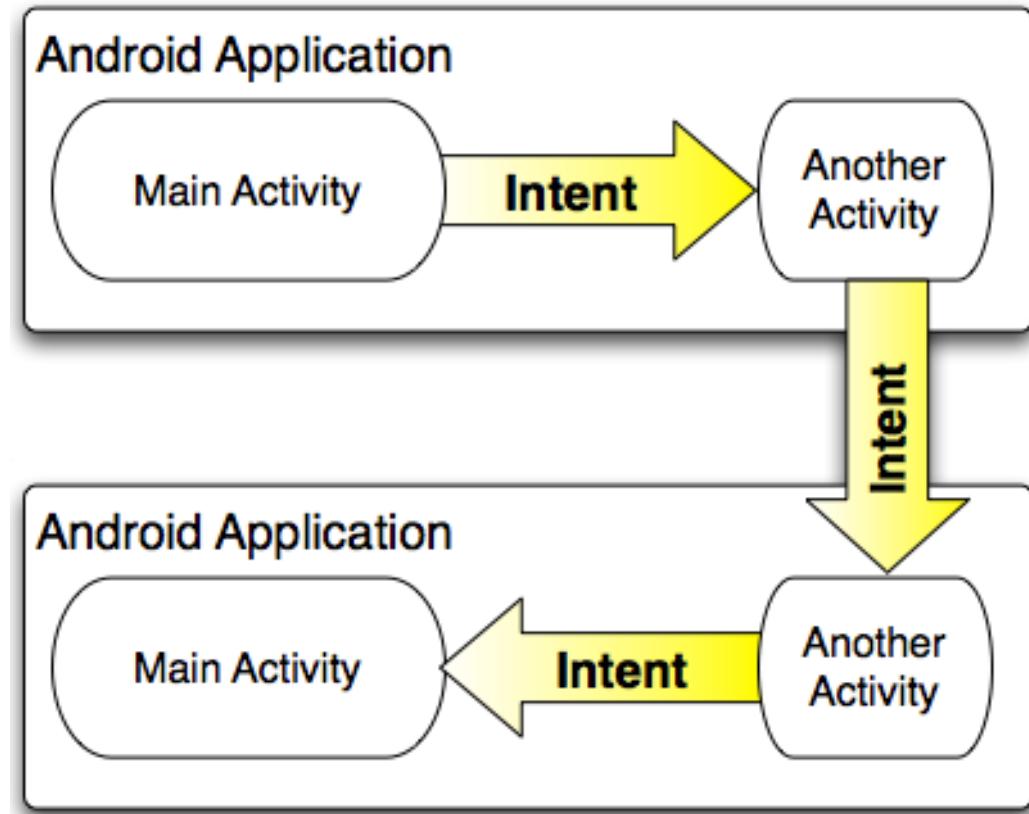




- Les autres applications n'accèdent pas directement à la classe de ContentProvider
- Utilisation d'un ContentResolver qui va rediriger les requêtes vers le provider voulu
- Si l'on tente d'accéder à une ressource d'une application n'étant pas en cours d'exécution le système Android se charge de la lancer avant.

Les applications :: Intent

- Les content providers sont activés par une requête d'un content resolver
- Mais les 3 autres systèmes (Activity, Service, BroadCast Receiver) sont activés par des messages asynchrone appelés "Intent" ou Intention
- Un intent dérive de android.content.Intent
- Un intent possède une action et un contenu particulier





- Pour les activités et les services il nomme l'action désirée et précise l'URI des données sur lesquelles agir.
 - Afficher / image
 - Editer / texte
 - ...
- Pour les broadcast receivers il se contente de nommer l'action à annoncer
 - Batterie faible



- Les Intents et les activités :
 - Lancement en passant un Intent en paramètre à une des méthodes suivantes :
 - Context.startActivity()
 - Activity.startActivityForResult()
 - L’activity peut accéder à celui ci avec :
 - getIntent()
 - Si le système doit envoyer des nouveaux intents :
 - Appel de onNewIntent() sur l’activité
 - En cas de résultat attendu
 - Appel de onActivityResult() sur l’activité appelante



- Les Intents et les services :
 - Lancement en passant un Intent en paramètre à la méthode suivante :
 - Context.startService()
 - Le système appellera ensuite la méthode onStart() en précisant cet Intent en paramètre
 - Connexion en passant un Intent en paramètre à la méthode suivante :
 - Context.bindService()
 - Le Système appellera ensuite la méthode onBind() en précisant cet Intent en paramètre



- Les Intents et les Broadcast receiver :
 - Une application voulant envoyer un évènement va utiliser une des méthodes suivantes :
 - Context.sendBroadcast()
 - Context.sendOrderedBroadcast()
 - Context.sendStickyBroadcast()
 - Le système va alors appeler la méthode onReceive() sur tous les broadcast receivers intéressés en passant en paramètre l'Intent.



- La catégorie
 - Une chaîne de caractère précisant quel type de composant peut gérer l'intent.
 - Plusieurs catégories peuvent être précisées.
 - Exemples :
 - CATEGORY BROWSABLE : Le contenu peut être Affiché dans le navigateur
 - CATEGORY HOME : L'activité est de type Home
 - CATEGORY LAUNCHER : par ex l'activité principale d'une application appartient à cette catégorie, pour indiquer qu'elle apparaît dans le menu du launcher
 - CATEGORY PREFERENCE : l'activité est un panneau de préférences



- Quelques exemples :
 - ACTION_VIEW content://contacts/people/1 – Affiche les information sur le contact 1
 - ACTION_CALL content://contacts/people/1 – Affiche le mode d'appel rempli avec les informations du contact 1
 - ACTION_VIEW tel:123 – Affiche le mode d'appel rempli avec "123". (ACTION VIEW s'adapte donc au contenu)
 - ACTION_CALL tel:123 – Idem
 - ACTION_EDIT content://contacts/people/1 – Permet de modifier les informations du contact 1
 - ACTION_VIEW content://contacts/people/ – Affiche la liste des contacts



- Intent Filter : Récepteur d'intention -> ils informent le système à quel « intent » les composants peuvent réagir.
- Un composant peut avoir plusieurs filtres
- Chaque composant peut définir ses filtres d'intention au niveau du manifest
 - ACTION : Quelles actions sont supportées ?
 - DATA : Pour des données de quelle nature?
 - CATEGORY : Dans quelles circonstances ?
- Permettra au bus de savoir si le message d'activation (i.e. l'intent) doit être délivré au composant ou non



- Exemple de filtre:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    http://schemas.android.com/apk/res/android>  
    <application>  
        <activity android:name=".Now" android:label="Now">  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>  
    </application>  
</manifest>
```

L'élément intent-filter sous l'élément activity annonce que :

- Cette activité est l'activité principale de cette application.
- Elle appartient à la catégorie LAUNCHER, ce qui signifie qu'elle aura une icône dans le menu principal d'Android.



Les applications :: Manifest

- Fichier XML
- Précise l'architecture de l'application
- Chaque application doit avoir un fichier AndroidManifest.xml à la racine du projet

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.marakana"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".HelloAndroid"
            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>
    <uses-sdk android:minSdkVersion="5" />
</manifest>
```





- Contenu :
 - Précise le nom du package java utilisant l'application. Cela sert d'identifiant unique !
 - Il décrit les composants de l'application:
 - Liste des activités, services, broadcast receivers
 - Précise les classes qui les implémentent
 - Précise leurs capacités (à quels intents ils réagissent)
 - Ceci permet au système de savoir comment lancer chaque partie de l'application afin de satisfaire au principe de réutilisabilité.



- Contenu (suite) :
 - Définit les permissions de l'application:
 - Droit de passer des appels
 - Droit d'accéder à Internet
 - Droit d'accéder au GPS ...
 - Précise la version d'Android minimum nécessaire
 - Déclare les librairies utilisées
 - Déclare des outils d'Instrumentation (uniquement pour le développement)



- Conventions :
 - Seuls deux éléments sont obligatoires
 - < manifest > : contient le package, la version... Englobe tout le fichier
 - < application > : décrit l'application et contiendra la liste de ses composants.
 - Les données sont passées en tant qu'attribut et non en tant que contenu
 - Tous les attributs commencent par "android:" (sauf quelques un dans < manifest >)



- Les ressources
 - Au lieu de contenir les données en tant que tel le fichier manifest peut faire appel à des ressources
 - < activity android : icon = "@drawable/smallPic" ... >
 - Ces ressources sont définies dans le répertoire « res » de l’application.



- Permissions
 - Une application ne peut utiliser certaines fonctionnalités à moins que cela ne soit précisé dans le fichier Manifest
 - Il faut donc préciser les permissions nécessaires grâce à :
 - < uses permission >
 - Il existe des permission standard :
 - android.permission.CALL EMERGENCY NUMBERS
 - android.permission.READ OWNER DATA
 - android.permission.SET WALLPAPER
 - android.permission.DEVICE POWER
 - Il est possible de définir ses propres permissions

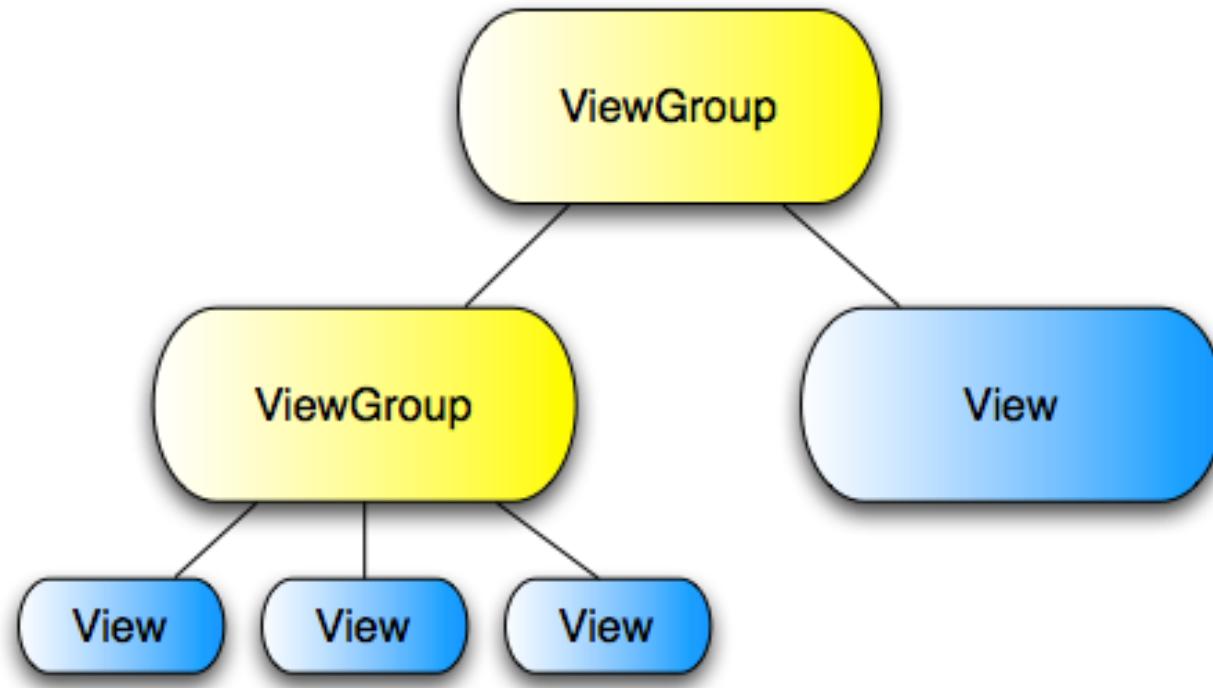


Interface graphique utilisateur (GUI)



- Une API Java riche
 - Des layouts et des widgets (appelés “Vues”)
- Programmation déclarative XML
 - Sépare la vue du code métier
- Fonctionnalité de personnalisation
 - Hériter et redéfinir un widget de base
 - Combiner des widgets existants
- Rendu 2D/3D
 - OpenGL, Renderscript

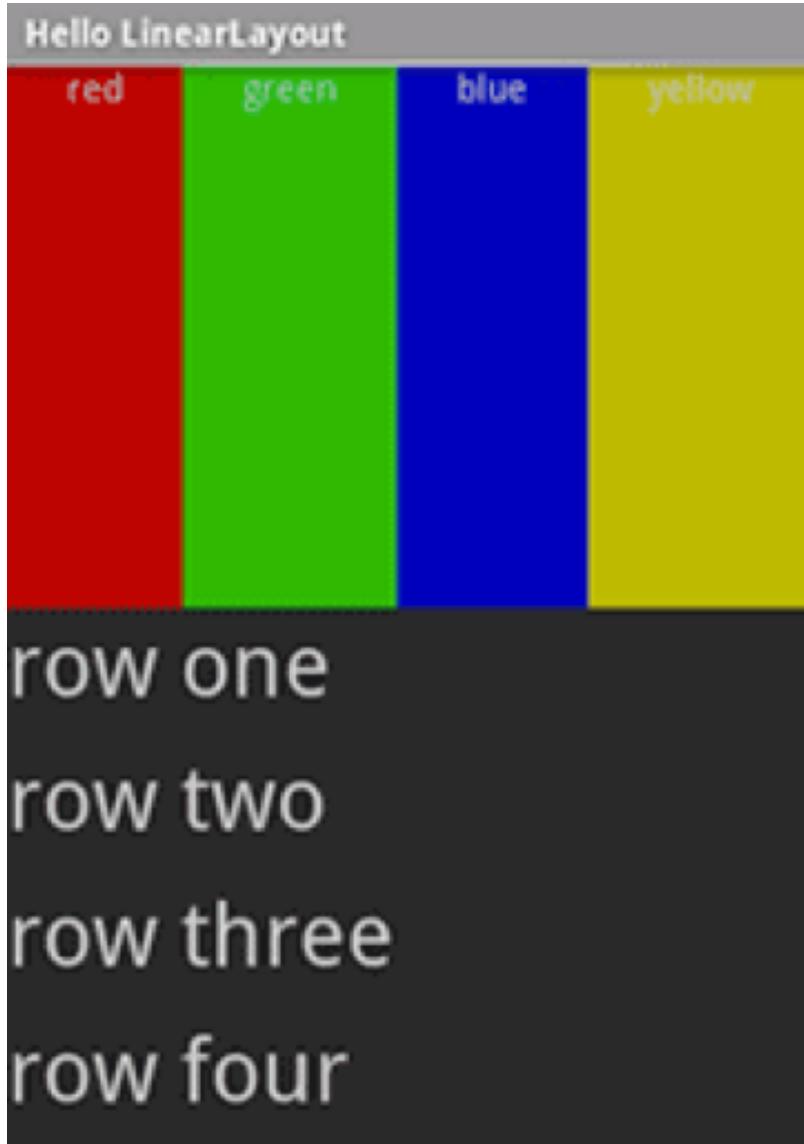
- Les vues et les layouts



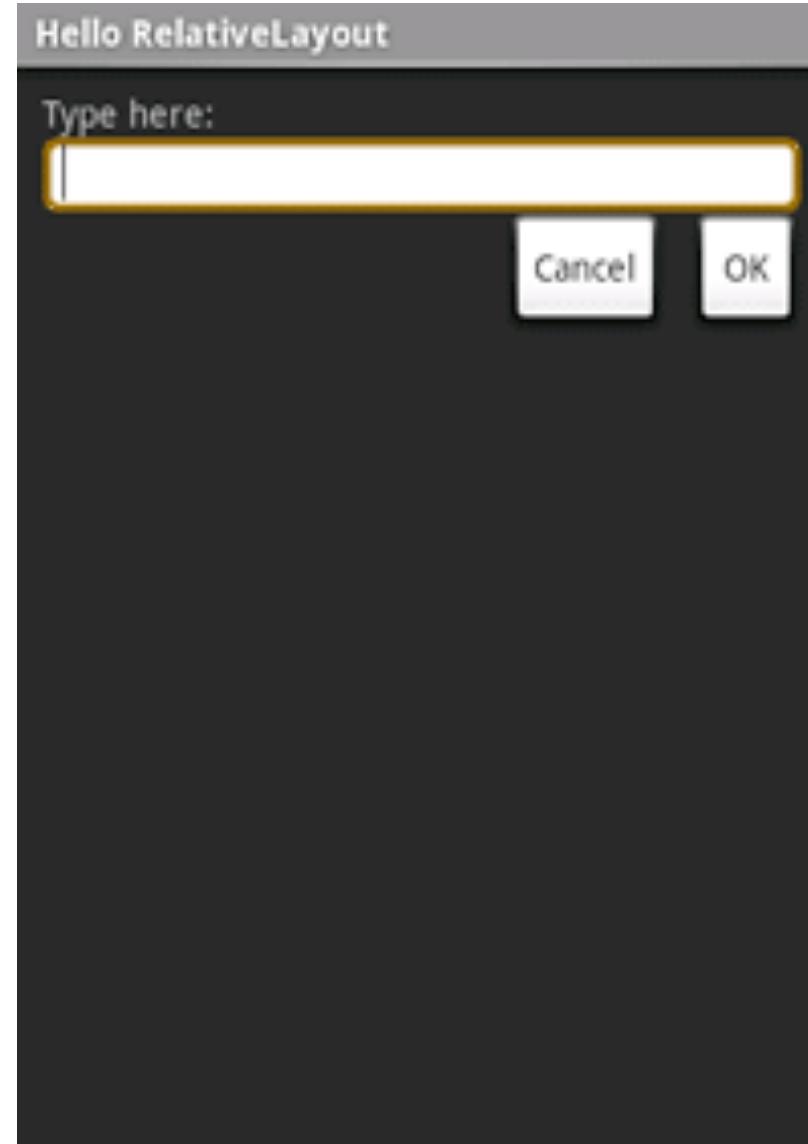


- Sous Android, la notion de mise en page est reliée à la notion de « Layout ». Cette dernière représente l’agencement des différents éléments graphiques dans votre interface.
- Voici quelques exemples de layouts :
 - LinearLayout
 - FrameLayout
 - RelativeLayout
 - TableLayout
 - GridView
 - TabHost

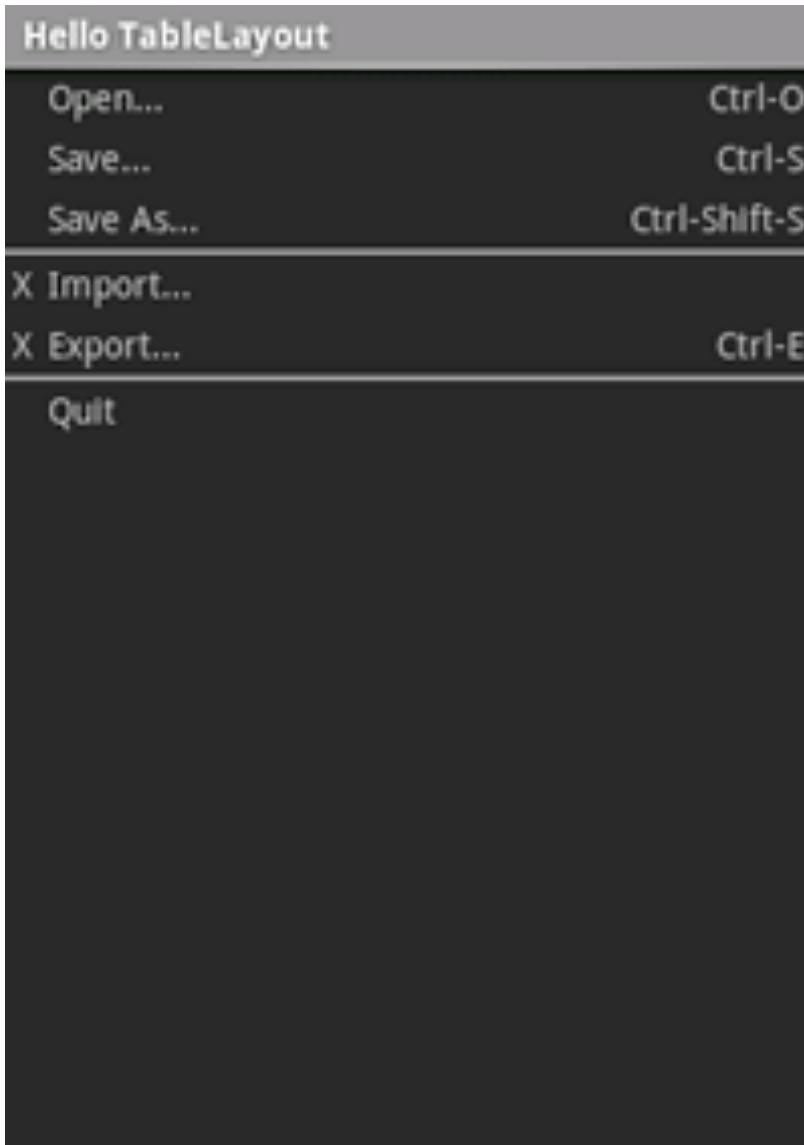
- LinearLayout



- RelativeLayout



- TableLayout



- FrameLayout

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

- AbsoluteLayout

- Pour placer les éléments par positions fixes



- Orientation
 - Sens de placement des vues dans un conteneur
 - android:orientation = vertical | horizontal
- Taille
 - Surface prise par la vue
 - android:layout_width/android:layout_height = <??>px
| fill_parent | wrap_content
- Gravité
 - Alignement d'une vue dans son conteneur
 - android:layout_gravity = left | center_horizontal | top
| bottom | right | ...



- **Poids**
 - Taux d'espace libre affectés à chaque widgets
 - android:layout_weight = ? (0 par défaut)
- **Espacement (intra)**
 - Espacement entre un contenu et les bords de sa vue
 - android:padding? = top | left | right | bottom
- **Espacement (inter)**
 - Espacement entre les vues
 - android:layout_margin? = ??px



- Utiliser des ressources depuis le code
 - La plupart des éléments de l'API sont prévus pour accepter des ressources Android en paramètre (int ressource)
 - `obj.setColor(R.color.rose_bonbon);`
`obj.setColor(android.R.color.darker_gray);`
- Référencer des ressources depuis d'autres ressources
 - `attribute="@+[packageName]:resourcetype/resourcident"`
 - `<EditText android:textColor="@color/rose_bonbon"/>`
 - `<EditText android:textColor="@android:color/darker_gray"/>`



- Il est nécessaire d'instancier les vues (i.e. obtenir des objets) pour pouvoir les manipuler
- Récupérer les widgets depuis le code
 - Grâce aux identifiants affectés à chaque vue
 - `Button myButton = (Button)
 findViewById(R.id.my_button);`
- Récupérer toute une vue depuis le code
 - Déserialiser (inflate) un fichier XML décrivant un layout ou un menu
 - `View my_view = LayoutInflater.inflate(R.layout.main,
 null);`
 - `MenuItemInflater.inflate(R.menu.control, my_menu);`



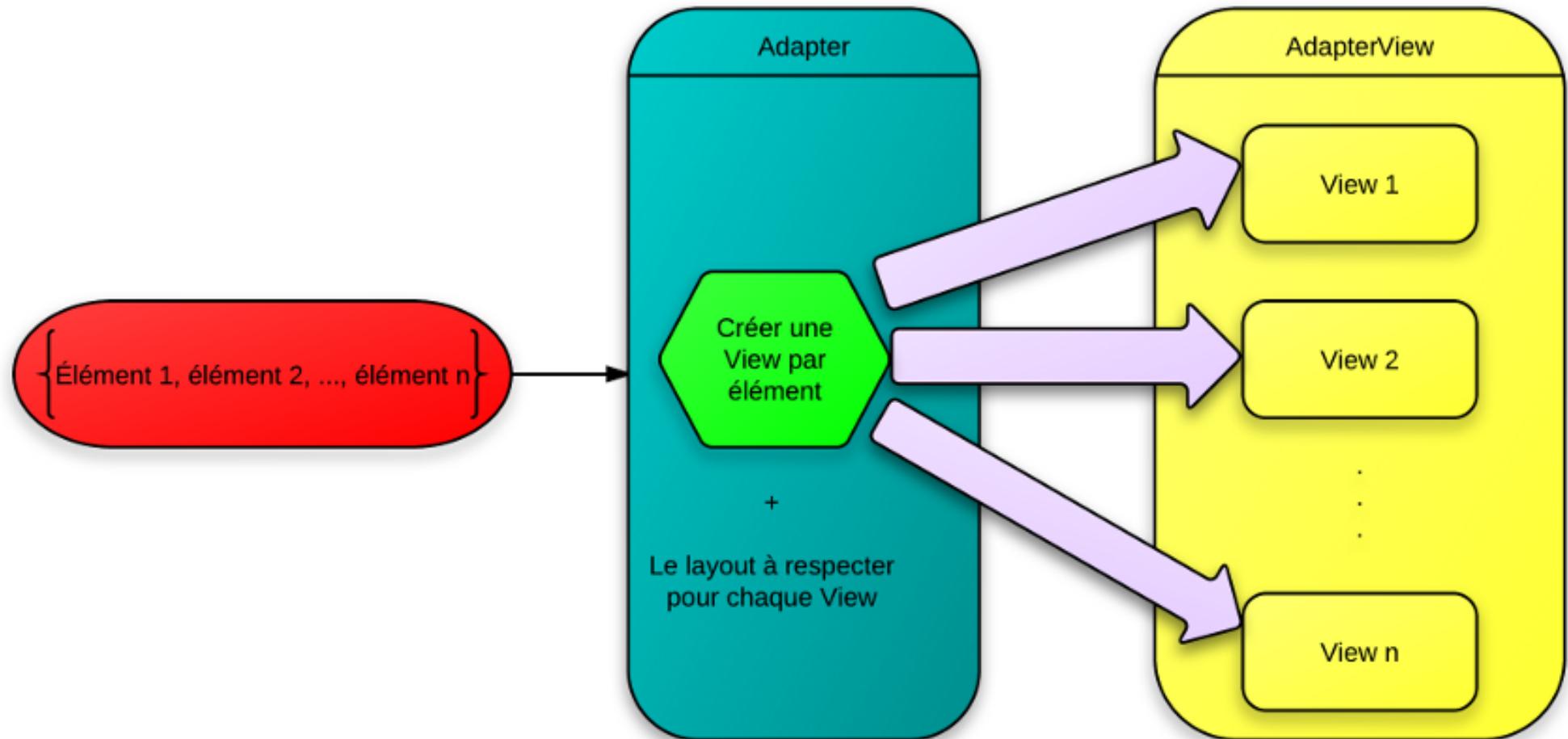
- Qu'est ce qu'un Listener?
 - Un listener se traduit par un « écouteur ».
 - Écoute une source jusqu'à ce qu'elle soit utilisée.
 - Pour qu'une action soit réalisée, il faut qu'un émetteur envoie un signal dans le canal d'écoute pour que le Listener se déclenche.
- Principe des écouteurs
 - Fournir une implémentation respectant un contrat (interface) afin de réagir à des types d'évènements particuliers :
 - Gestion du KeyPad (tap, trackball)
 - OnClickListener, OnLongClickListener
 - onClick(View), onLongClick(View)
 - OnKeyListener
 - onKeyUp(KeyEvent), onKeyDown(KeyEvent)
 - Gestion du TouchScreen (pression, gesture)
 - OnTouchListener
 - onTouchEvent(MotionEvent)



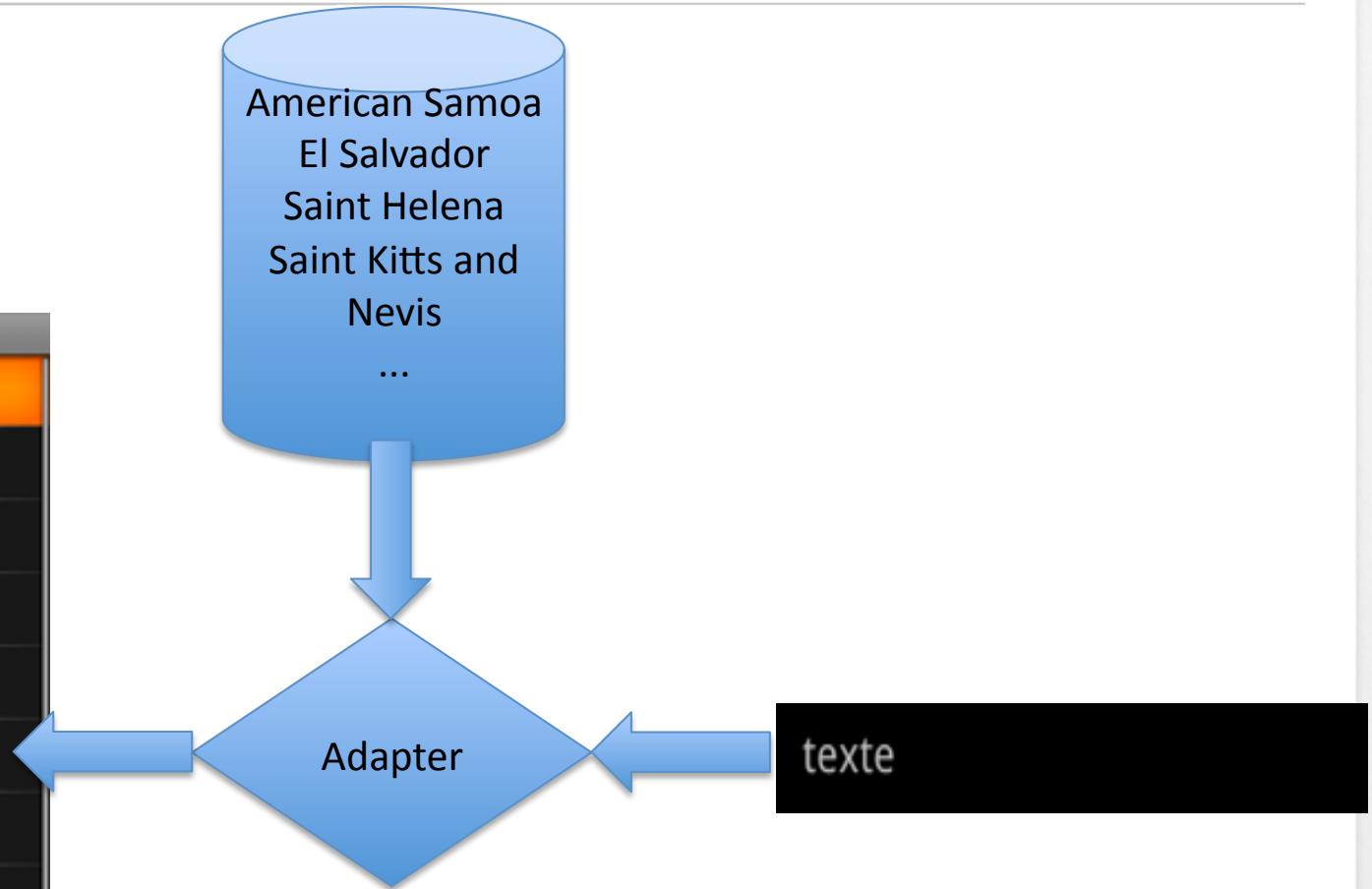
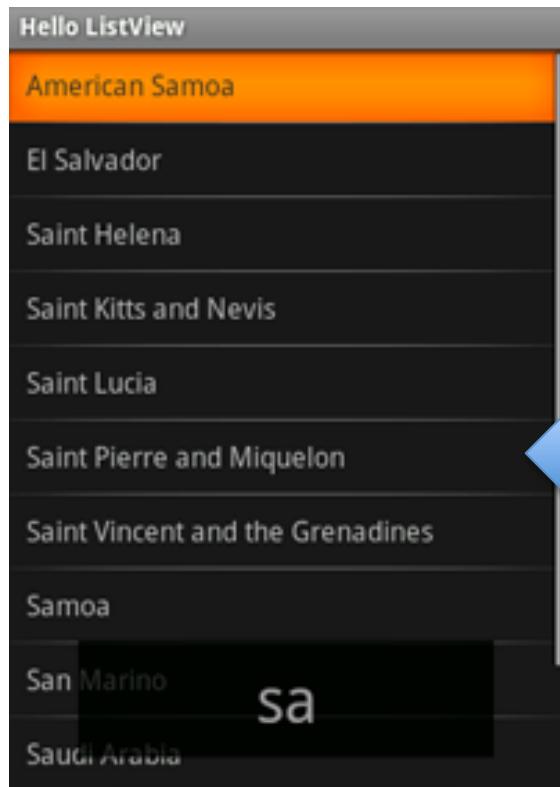
- Les adaptateurs sont des classes qui lient des données aux vues de l'UI
 - Les vues concernées étendent android.widget.AdapterView
- Responsables de la création des vues « enfants »
- Classes d'adaptateurs
 - Héritent de android.widget.BaseAdapter
 - Adapters natifs: SimpleAdapter, ArrayAdapter<?> (sert à récupérer des données stockées dans une collection)
 - Exploite par défaut la valeur de la méthode `toString()` des objets de la liste
 - CursorAdapter : sert à récupérer des données stockées dans une base de données relationnelle (SQLite)
 - Vous pouvez étendre ces classes de base pour gérer finement vos items (conseillé)



GUI :: Les adaptateurs



GUI :: Les adaptateurs





ArrayAdapter

- Utilise les types génériques pour lier une View Adapter à un tableau d'objet de la classe spécifiée
- Utilise la valeur `toString` du tableau pour remplir la `textView`
- Exemple d'appel :

```
ArrayAdapter<String> aa = new ArrayAdapter<String>(this, R.layout.list_item,  
COUNTRIES);  
myListView.setAdapter(aa);
```



SimpleAdapter

- Adapter simple pour lier des données à des vues

```
SimpleAdapter mSchedule = new SimpleAdapter (this.getBaseContext(), listItem,  
    R.layout.layoutitem, new String[] {"img", "title", "description"}, new int[] {R.id.img,  
    R.id.title, R.id.description});
```

- Contexte de la vue
- tableau d'objets
- Layout pour chaque item
- Les clefs de la hashmap (par exemple)
- Les ID des ressources correspondantes dans le layout



Custom Adapter

- Il est possible de définir ses propres adapters
- Étendre la classe BaseAdapter
- Surcharger les méthodes:
 - getView: génère la vue pour un objet
 - getCount: renvoie le nombre d'objets
 - getItem: renvoie l'item à la position
 - getItemId: retourne la position de l'item



Problème sur les jeux de données importants:

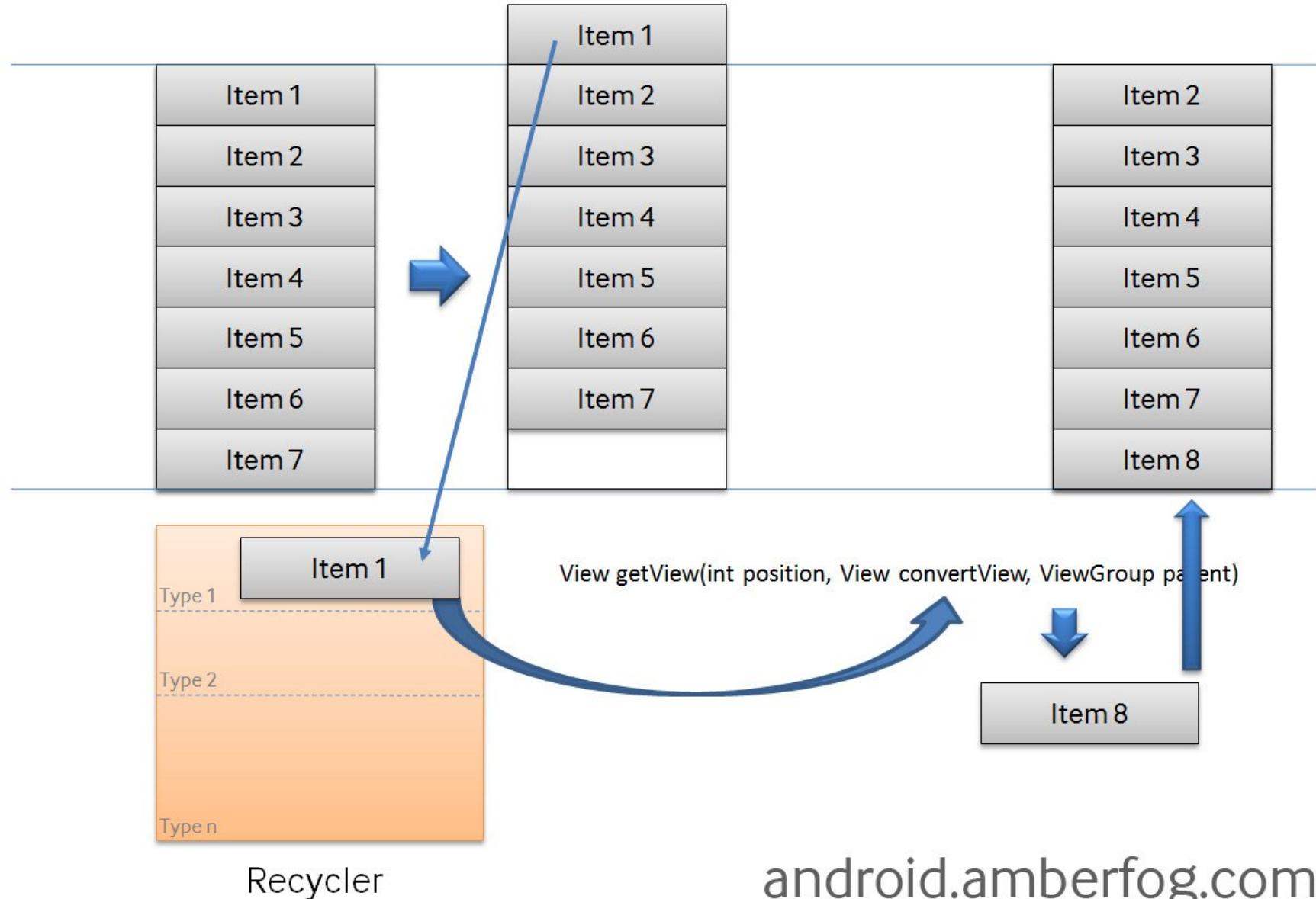
- Mémoire
- Performance

Solution

- « populate on demand »
- Recycler les vues pour réduire la création d'objets



GUI :: Les adaptateurs : Optimisations





GetView :: méthode non optimisée

```
1 public View getView(int position, View convertView, ViewGroup parent) {  
2     View item = mInflater.inflate(R.layout.list_item_icon_text, null);  
  
3     ((TextView) item.findViewById(R.id.text)).setText(DATA[position]);  
4     ((ImageView) item.findViewById(R.id.icon)).setImageBitmap(  
5         (position & 1) == 1 ? mIcon1 : mIcon2);  
  
6     return item;  
7 }
```

GetView :: La bonne méthode

```
1 public View getView(int position, View convertView, ViewGroup parent) {  
2     if (convertView == null) {  
3         convertView = mInflater.inflate(R.layout.item, parent, false);  
4     }  
  
5     ((TextView) convertView.findViewById(R.id.text)).setText(DATA[position]);  
6     ((ImageView) convertView.findViewById(R.id.icon)).setImageBitmap(  
7         (position & 1) == 1 ? mIcon1 : mIcon2);  
  
8     return convertView;  
9 }
```



GetView :: La plus rapide

```
1 public View getView(int position, View convertView, ViewGroup parent) {  
2     ViewHolder holder;  
3  
4     if (convertView == null) {  
5         convertView = mInflater.inflate(R.layout.list_item_icon_text,  
6                                         parent, false);  
7         holder = new ViewHolder();  
8         holder.text = (TextView) convertView.findViewById(R.id.text);  
9         holder.icon = (ImageView) convertView.findViewById(R.id.icon);  
10  
11         convertView.setTag(holder);  
12     } else {  
13         holder = (ViewHolder) convertView.getTag();  
14     }  
15  
16     holder.text.setText(DATA[position]);  
17     holder.icon.setImageBitmap((position & 1) == 1 ? mIcon1 : mIcon2);  
18  
19     return convertView;  
20 }
```



Header & Footer

- `ListView.addHeaderView()`
- `ListView.addFooterView()`
- Ces méthodes doivent être appelées avant `setAdapter()`

List selector

- « highlight » l'item sélectionné
- Non visible en « touch mode » (ex: un téléphone avec un trackball)
 - Il n'y a aucune sélection dans ce mode
- Visible derrière les items de la liste
 - `android:drawSelectorOnTop="true"`



GUI :: Les adaptateurs : Customiser la liste

```
1 <selector>
2
3     <item android:state_window_focused="false"
4         android:drawable="@color/transparent" />
5
6     <item android:state_focused="true" android:state_enabled="false"
7         android:state_pressed="true"
8         android:drawable="@drawable/list_selector_background_disabled" />
9
10    <item android:state_focused="true" android:state_enabled="false"
11        android:drawable="@drawable/list_selector_background_disabled" />
12
13    <item android:state_focused="true" android:state_pressed="true"
14        android:drawable="@drawable/list_selector_background_transition" />
15    <item android:state_focused="false" android:state_pressed="true"
16        android:drawable="@drawable/list_selector_background_transition" />
17
18    <item android:state_focused="true"
19        android:drawable="@drawable/list_selector_background_focus" />
20
21 </selector>
```

Il suffit ensuite de définir un background dans le fichier xml décrivant une cellule :
android:background="@drawable/listview_selector"



- **android:transcriptMode**
 - Comportement de la liste quand le contenu change
 - “disabled” -> ne scrolle pas
 - “normal” -> scrolle vers le bas si le dernier élément est visible
 - “alwaysScroll” -> scrolle toujours vers le bas
- **android:stackFromBottom**
 - Empile les éléments dans l’ordre inverse
 - Commence avec le dernier item de l’adapter
 - Utile pour les tchats
 - Messaging, Talk, IRC, etc.



- Les webservices de type REST (Representational State Transfer) permettent d'exposer des fonctionnalités comme un ensemble de ressources (URI) accessibles par la syntaxe et la sémantique du protocole HTTP.
- Format des données renvoyées:
 - XML
 - Parsing de ressources au format XML
 - Voir API javax.xml.parsers
 - Approche hiérarchique (DOM) : org.w3c.dom
 - Approche évènementielle (SAX) : org.xml.sax
 - JSON
 - JSON est un format léger d'échange de données
 - Il est facile à écrire et lire, facilement analysable
 - C'est un format totalement indépendant de tout langage. Ce qui en fait un langage d'échange de données idéal.
 - Il se base sur deux structures :
 - Une collection de couple "Nom / Valeur"
 - Une liste de valeurs ordonnées



- Il existe plusieurs librairies JSON
 - Sojo
 - JSON.org
 - FlexJSON
 - Jackson
 - Gson
 - Et bien d'autres....



- Les tâches consommatoires de ressources doivent s'exécuter dans un Thread séparé.
- Implémentation « Typique » : les AsyncTasks
 - Étends la class AsyncTask <Url, void, JSONObject >
 - **onPreExecute** : appelée avant le traitement
 - **doInBackground** : traitement en tâche de fond
 - **onPostExecute** : appelé après le traitement
 - **onProgressUpdate** : appelé pendant le traitement pour afficher la progression par exemple.
 - Exemple d'appel:
`new LoadItemsTask().execute(new URL(...));`



Client / Serveur :: Accès Internet

Java

```
// private class LoadItemsTask extends AsyncTask<URL, Void, JSONObject> {

    protected JSONObject doInBackground(URL... params) {
        HttpURLConnection conn = (HttpURLConnection) params[0].openConnection();
        InputStream input = conn.getInputStream();
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        copy(input, baos);
        JSONObject jsonRoot = new JSONObject(baos.toString());
        return jsonRoot;
    }
}
```

Java

```
// private class LoadItemsTask extends AsyncTask<URL, Void, JSONObject> {

    protected void onPostExecute(JSONObject jsonRoot) {
        List<Items> items = parseJson(jsonRoot);
        appendItemsToList(item);
        notifyDataSetChanged();
    }
}
```



Client / Serveur :: Accès Internet

Java

```
// private class LoadImageTask extends AsyncTask<URL, Void, Bitmap> {  
  
    public LoadImageTask(ImageView imageView) {  
        mImageView = imageView;  
    }  
  
    protected Bitmap doInBackground(URL... params) {  
        HttpURLConnection conn = (HttpURLConnection) params[0].openConnection();  
        InputStream input = conn.getInputStream();  
        return BitmapFactory.decodeStream(input);  
    }  
}
```

Java

```
// private class LoadImageTask extends AsyncTask<URL, Void, Bitmap> {  
  
    protected void onPostExecute(Bitmap result) {  
        mImageView.setImageBitmap(result);  
    }  
}
```

```
new LoadImageTask(holder.imageView).execute(  
    new URL(BASE_URL + item.imageUrl));
```



- Plusieurs problèmes
 - AsyncTask et les rotations: lors d'une rotation, l'Activité est détruite, puis recréée => du coup les données sont rechargées.
 - AsyncTask et cycle de vie: AsyncTask continue de s'exécuter même si l'Activité ou l'application ont été détruites -> il faut tuer explicitement la tâche: `AsyncTask.cancel()`.
 - Stopper une tâche: La tâche n'est pas tuée immédiatement, elle passe dans un état « cancelled », mais du coup il faut gérer cet état pendant le traitement de longues opérations.
 - Limitations sur le nombre de tâches lancées, un trop grand nombre de tâches lancées, fait « crasher » une application
 - Les AsyncTasks s'exécutent en série et non en parallèle



- **Librairie Volley**

- Bibliothèque permettant de construire facilement des applications réseaux très performantes sur Android.
- Volley fonctionne avec un mécanisme de thread pool configurable
- Il supporte la priorisation des requêtes
- Les résultats des requêtes sont automatiquement mis en cache disque et mémoire
- On peut très facilement annuler des requêtes en cours, afin d'éviter qu'elles ne soient exécutées pour rien
- Supporte différents formats: images, json, texte brut,...

```
git clone https://android.googlesource.com/platform/frameworks/volley
```



- Implémenter Volley
 - Intégrer les sources directement dans le projet ou en liant la librairie.
 - Dans la main Activity, il faut déclarer un objet RequestQueue.

```
1 | private RequestQueue mRequestQueue;  
2 | .  
3 | .  
4 | .  
5 | mRequestQueue = Volley.newRequestQueue(this);
```

- Crédit d'un l'objet JsonObjectRequest en lui passant divers paramètres comme : méthode http, listeners en cas de succès, d'erreur



Client / Serveur :: Accès Internet

```
01 JsonObjectRequest jr = new
 02   JsonObjectRequest(Request.Method.GET,url,null,new
 03     Response.Listener<JSONObject>() {
 04       @Override
 05         public void onResponse(JSONObject response) {
 06           Log.i(TAG,response.toString());
 07           parseJSON(response);
 08           va.notifyDataSetChanged();
 09           pd.dismiss();
 10     }
 11   },new Response.ErrorListener() {
 12     @Override
 13       public void onErrorResponse(VolleyError error) {
 14         Log.i(TAG,error.getMessage());
 15       }
 16   });
 17 }
```

- Ajout de l'objet jsonObjectRequest à la RequestQueue:
mRequestQueue.add(jr);



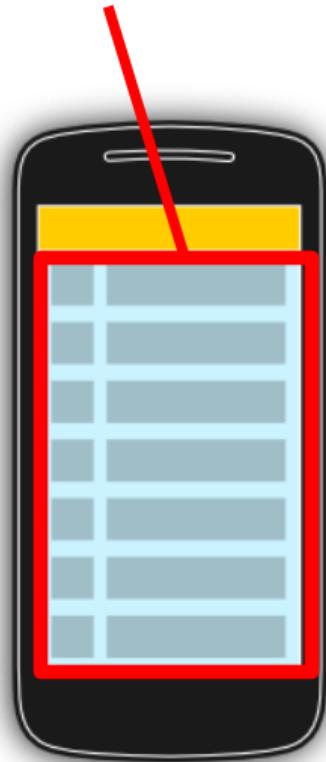
- Le « Toast » (le plus simple): permet d'afficher de petits messages rapides à l'utilisateur. Moins intrusif qu'une popup, mais réservé à des petites notifications.
- « Dialog »: *AlertDialog, ProgressDialog, DatePickerDialog ou TimePickerDialog*.
- « Personnaliser un Dialog »: Utilisation de la classe `LayoutInflater` qui va nous permettre de générer une « view » à partir d'un fichier xml et de l'affecter au « Dialog » grâce à la méthode `setView()`.
- Les « FragmentDialog »: utilisation recommandée par Google.



- Nouveauté depuis Android 3.0
 - Prévus pour les écrans plus larges, comme les tablettes
- Principe de base
 - Fragmentation de l'espace d'affichage en différentes zones, chargeables indépendamment, et réutilisables
 - Même idée que les "Frames" en HTML
- Un fragment est une classe qui étend
 - android.app.Fragment
- Les fragments sont ensuite attachés/détachés à une activité hôte



ListFragment

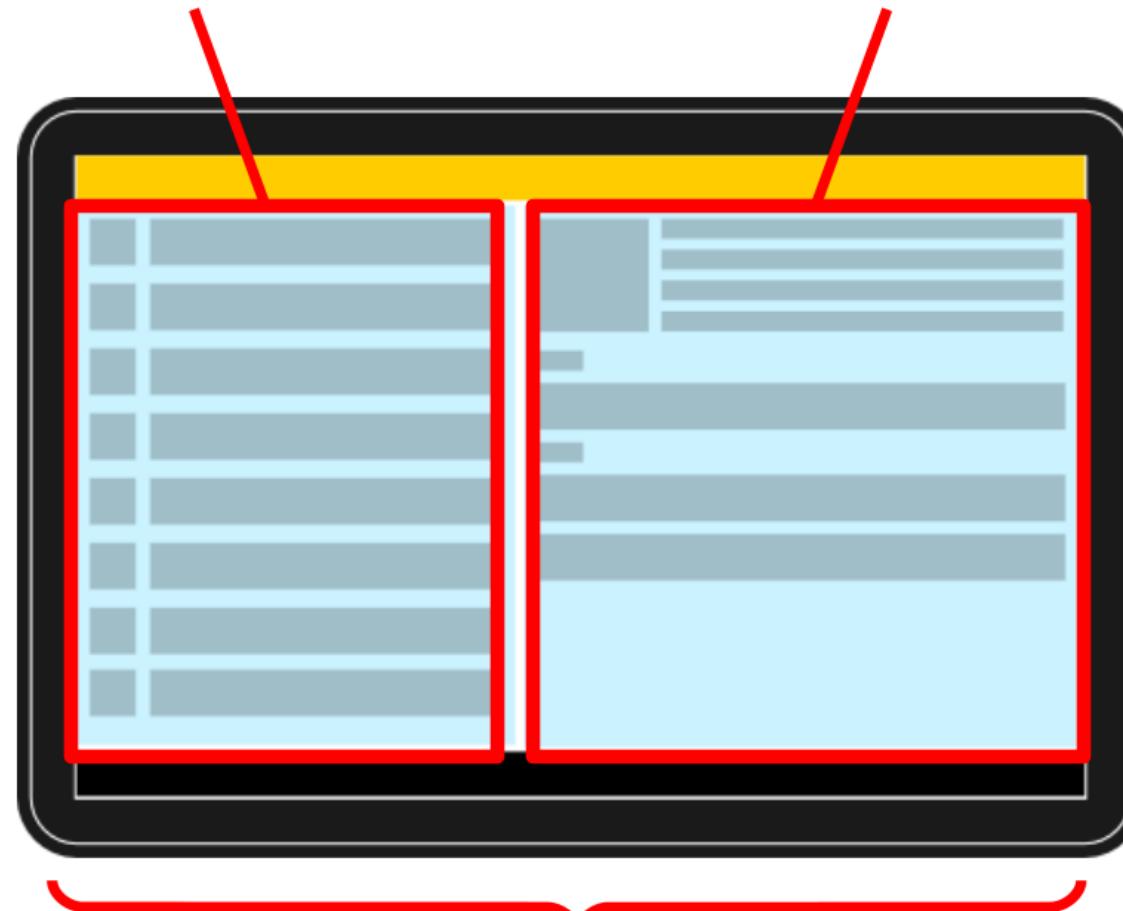


DetailsFragment



MainActivity*

ListFragment DetailsFragment



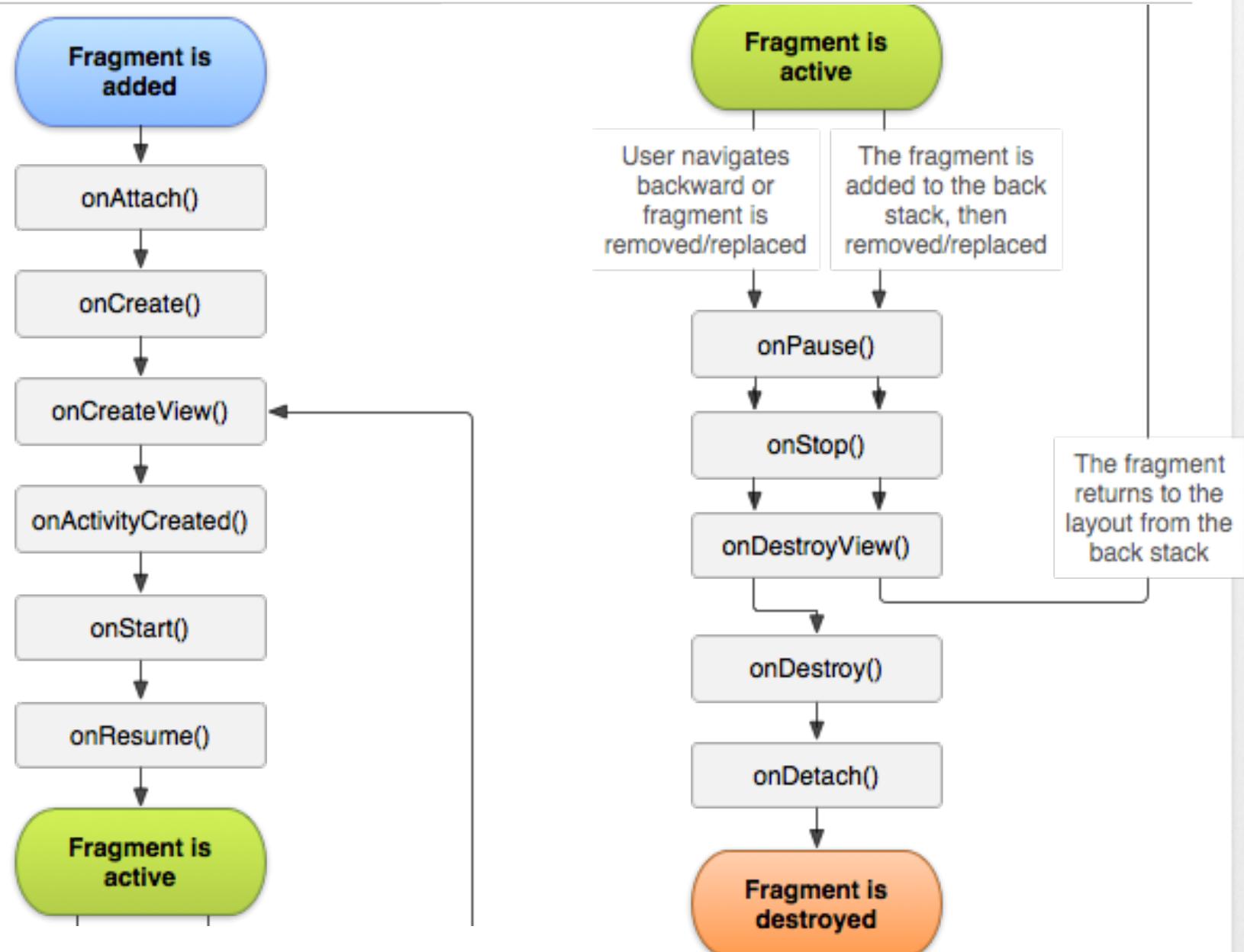
MainActivity



- Avantages:
 - Composants faciles à réutiliser
 - Possible de concevoir des applications « single panel » pour les téléphones et « multi-panel » pour les tablettes.
- Cycle de vie
 - Un fragment a son propre cycle de vie mais il est toujours connecté au cycle de vie de l'activité.

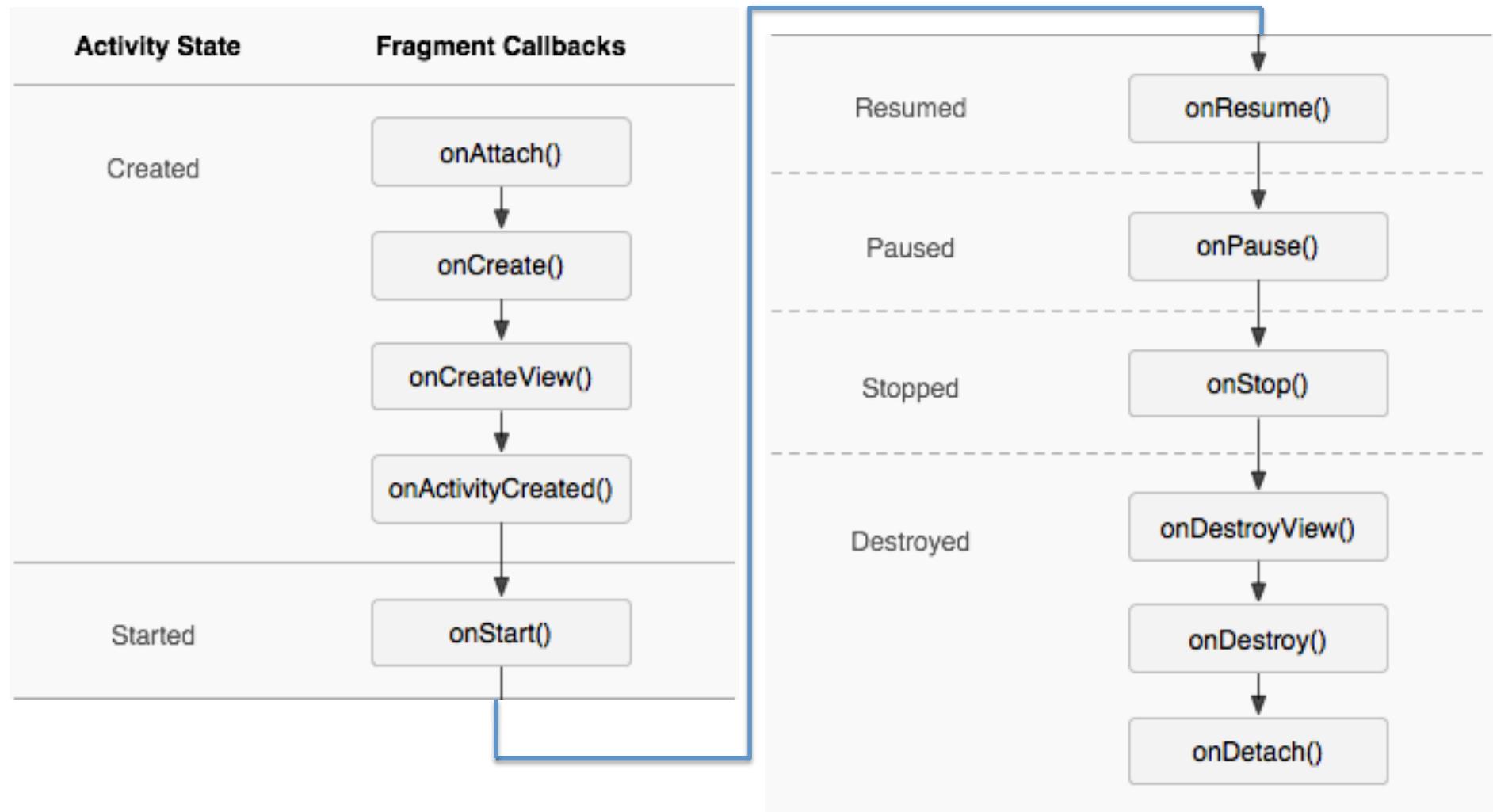


GUI :: Fragments





GUI :: Fragments





- Généralement un fragment implémentera les méthodes suivantes:
 - `onCreate(Bundle)` pour initialiser les différents composants
 - `onCreateView(LayoutInflater, ViewGroup, Bundle)` appelé lorsque l'interface est créée la première fois. Cette méthode permet de renvoyer la vue associée au fragment à l'activité
 - `onPause()` pour conserver les modifications utilisateurs pour qu'il puisse les retrouver lorsque le fragment sera à nouveau actif.
- Hérite de la classe `Fragment`, `ListFragment`,



Ajouter un fragment à une « Activité »:

- Déclarer le fragment dans le layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
        android:id="@+id/viewer"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```



- Ajouter un Fragment à un ViewGroup dans le code
Utilisation de l'API **FragmentTransaction**. On récupère l'instance de FragmentTransaction depuis l'Activity

```
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

Ensute on peut utiliser la méthode add en spécifiant le container et le fragment à insérer.

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```



- Un thème correspond à une "feuille de styles"
 - Ensemble de styles à appliquer à une activité
 - Thèmes par défaut : Theme.Holo.Dark, ...
- Un style est une ressource Android
 - System-defined : android.R.style.Widget_Button, ...
 - User-defined : R.style.Joli, R.style.Joli2, ...
 - Héritage de style possible en xml (parent="@style/Joli")
- Chaque vue est stylisable
 - Propriétés : taille, padding, background, textColor, ...
 - Un seul style applicable à la fois (android:style="@style/Joli")



- L'apparence des items est définie par défaut par des layouts système
 - android.R.layout.simple_spinner_item
- spécifie un texte aligné à gauche et un bouton radio à droite, ainsi qu'un texte noir sur fond blanc.
 - android.R.layout.simple_list_item_1
- Spécifie un texte aligné à gauche, ainsi qu'un texte blanc sur fond transparent.
 - ...
- Vous pouvez évidemment définir vos propres layouts pour créer des items plus complexes
 - fr.centrale.R.layout.mon_bel_item



- Différentes formes
 - LED
 - Son
 - Vibreur
 - Barre de notification (icône)
- Utilisation facilitée par le notification manager
 - `NotificationManager nm = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE
);`
 - `nm.notify(NUMBER, new Notification(...))`



- Utiliser les images 9 patchs
 - Images divisées en neuf zones (dont certaines étirables)
 - Outil Draw 9-patch du répertoire /tool du SDK Android
- Draw9patch.exe produit des fichiers *.9.png





- Prévoir différentes variantes d'une même chaîne
 - /res/values-fr/strings.xml On manipule une seule ressource,
 - /res/values-en/strings.xml
 - /res/values-it/strings.xmlsans se soucier de la langue (R.strings.hello)
- Le choix sera fait automatiquement en fonction de la configuration du terminal (ex: LOCALE=FR_fr)
- S'applique également aux images car elles peuvent afficher du texte !
 - /res/drawable-fr/splashscreen.png
 - /res/drawable-en/splashscreen.png



Shared Preferences ::

- Mécanisme simple et léger
 - Sauvegarde de paires clé/valeur simple
 - SharedPreferences pref =
getPreferences(Activity.MODE_PRIVATE)
- Sauvegarder des préférences
 - Récupère un éditeur de préférences :
Editor ed = pref.edit()
 - Stocke les paires :
ed.putString("teacher", "Olivier Le Goaer");
ed.putBoolean("isBrilliant", true);
 - Valide les modifications :
ed.commit();
- Retrouvez des préférences
 - String t = pref.getString("teacher","unknown");



Publier une application



- Toute application doit être signée
 - Signature digitale avec un certificat dont la clé privée est conservée par le(s) développeur(s)
 - Signature avec la clé de débugage par défaut
- Trois étapes :
 1. Obtenir une clé privée (stockée dans un "keystore")
Utilitaire keytool.exe du JDK
 2. Signer l'APK avec la clé privée
 - Procédure : Utilitaire jarsigner.exe du JDK entièrement automatisée
 3. Optimiser l'APK qui vient d'être signé



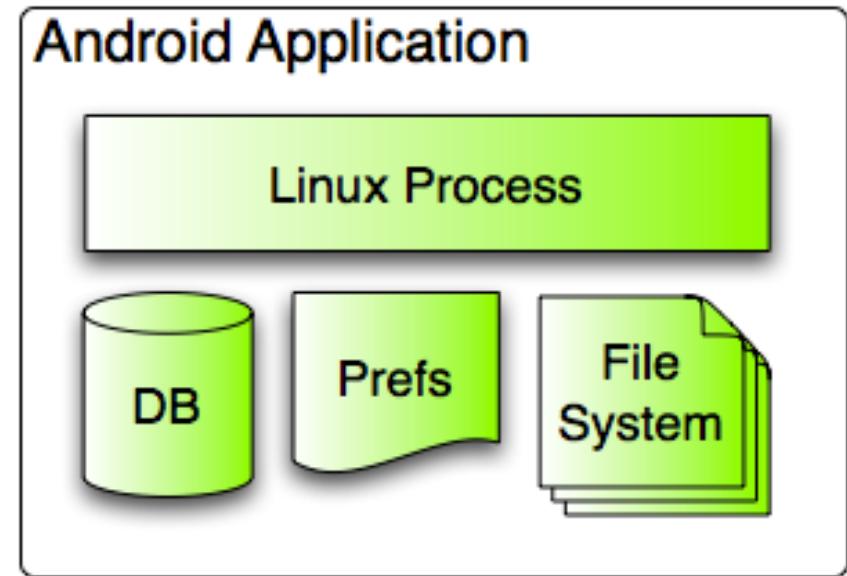
- 3 solutions s'offrent à vous
 - Choisir Google Play Store (Android Market)
 - Site officiel : <https://play.google.com>
 - 25\$ de frais de dossier pour l'accès au store
 - 70% du prix de vente va aux développeurs (30% à Google)
 - Les autres revenus générés sont reversés via Google Checkout
 - Autopublier sur votre propre site Web
 - Exemple : <http://www.univ-pau.fr/~olegoaer/bankster.apk>
 - Type MIME pour le téléchargement : application/vnd.android.package-archive
 - Choisir un magasin alternatif
 - YAAM (<http://yaam.mobi/>)
 - Bazaar (<http://www.bazaarandroid.com/>)
 - AndroLib (<http://fr.androlib.com/>)



Android et la sécurité

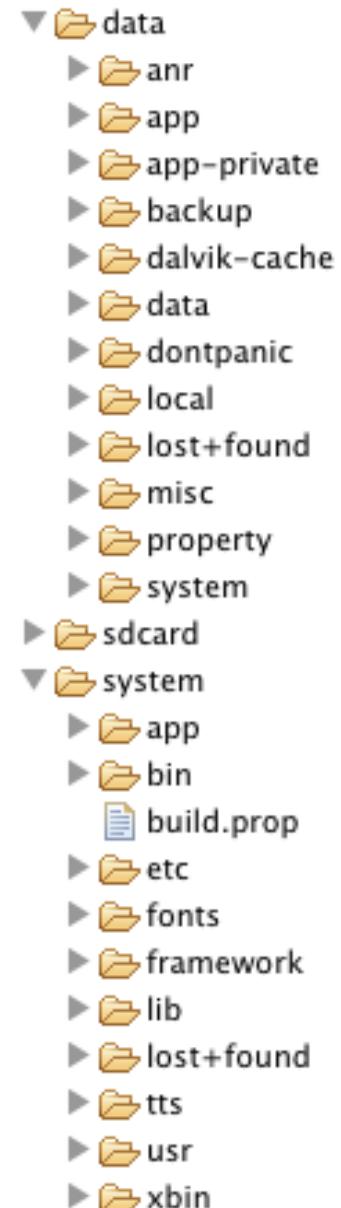


- Chaque application Android s'exécute dans son propre processus Linux
- Chaque application dispose de sa “sandbox”, avec ses propres préférences, sa base de données
- Les autres applications ne peuvent accéder à aucune de ces données à moins qu'elles n'est été partagée explicitement.





- Le “file system” dispose de 3 points de montage. Un pour le système, un pour les applications, et un pour le reste.
- Chaque application dispose de sa “sandbox”. Personne d'autre ne peut y accéder. La sandbox se trouve dans / data/data/<package name>
- La sdcard est toujours accessible. Tout le monde peut y accéder et c'est un bon endroit pour y stocker de gros fichiers comme des vidéos ou de la musique.

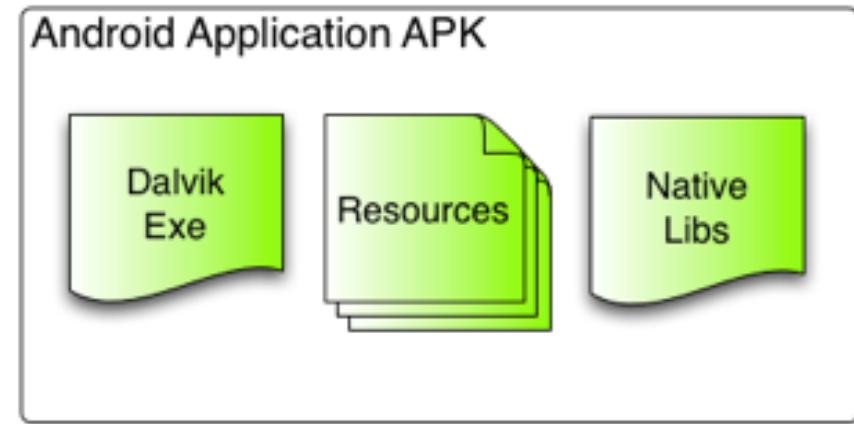




- La “Sandbox”
 - Une application est une « île » à elle seule
 - Elle contient un certain nombre d’activités, services, receivers et providers.
 - Elle dispose de son propre système de fichiers, base de données, librairies natives.
 - Aucune application ne peut accéder à n’importe laquelle de ses données, sans demander au préalable la permission.

Généralités :: Application

- Dalvik Executable + Resources = APK
- Doit être signé (mais la clef de debuggage est suffisante pour le développement)

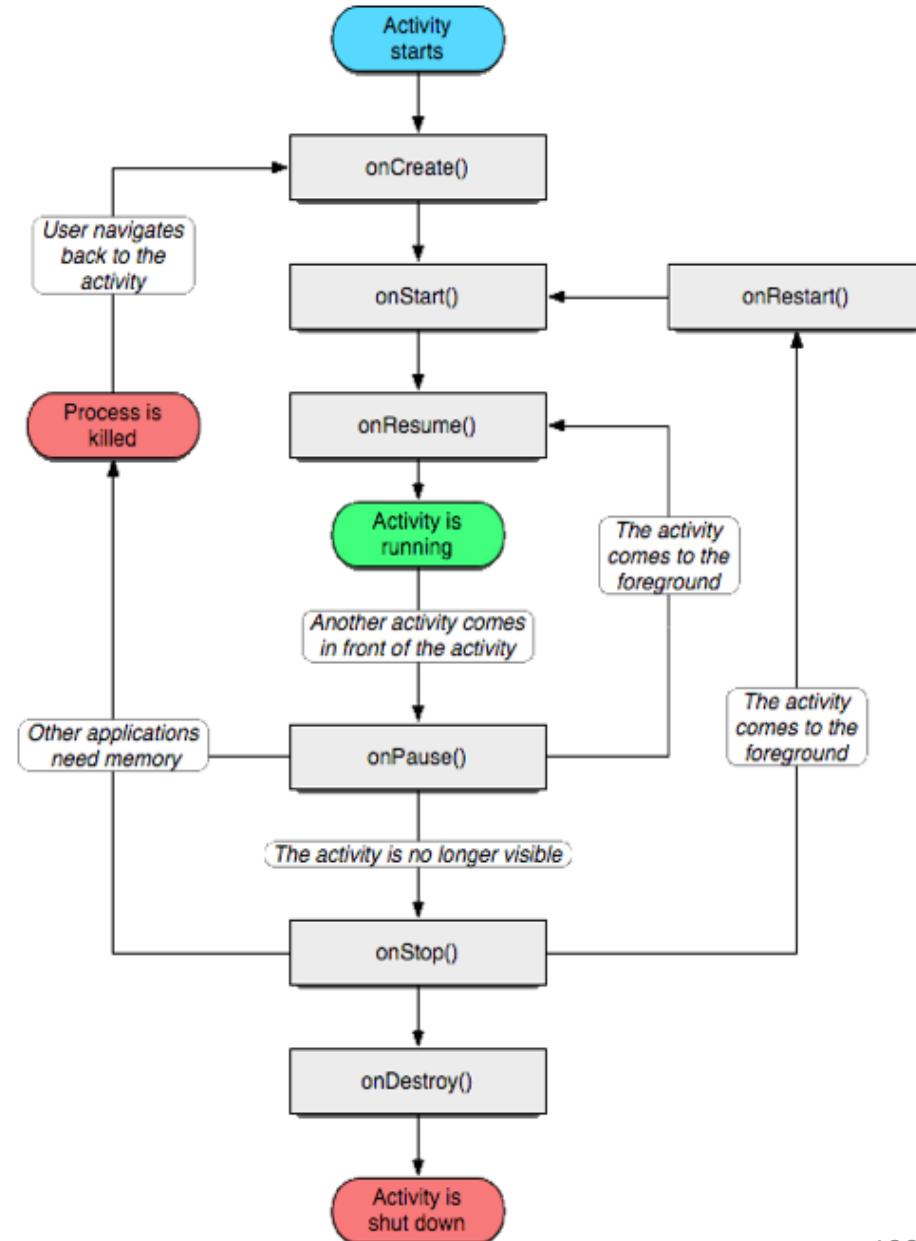




- Lorsqu'une activité passe en arrière-plan
 - L'état de tous les widgets (munis d'un id) sont automatiquement sauvegardé / restauré via un bundle
- Lorsqu'une activité doit être recréée
 - Deux cas de re-création complète suite à une destruction
 - 1. L'activité change de configuration (orientation, langage...)
 - 2. L'activité passe en arrière plan mais est tuée par le système (réquisition !)
 - Sauvegarde/restauration manuelle via un bundle
 - `onSaveInstanceState()` et `onRestoreInstanceState()`
 - Pour qu'un objet survive à un changement de configuration
 - `onRetainNonConfigurationInstance()` et `getLastNonConfigurationInstance()`

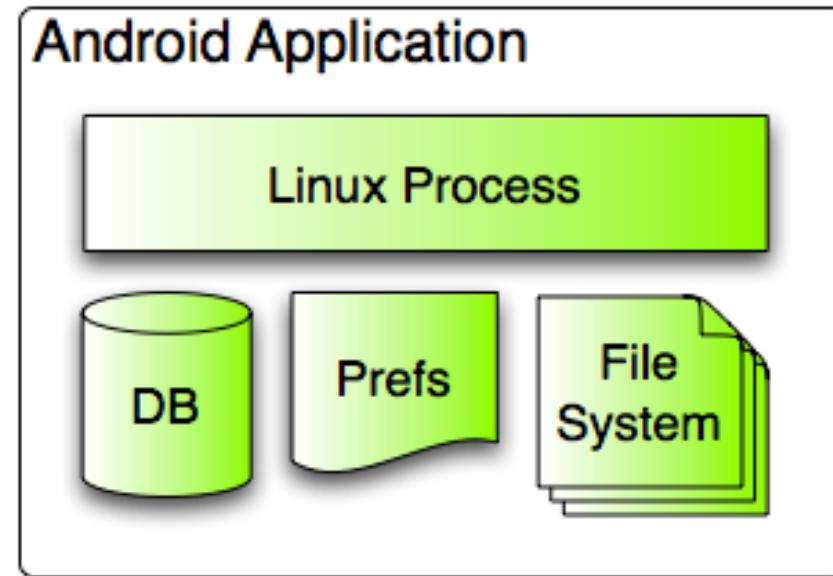
Les applications :: Cycle de Vie

- Une activité est notifiée de ses changements d'état par l'appel à ses méthodes :
 - void onCreate(Bundle savedInstanceState)
 - void onStart()
 - void onRestart()
 - void onResume()
 - void onPause()
 - void onStop()
 - void onDestroy()



Android :: Sécurité

- Les applications ne peuvent rien faire de potentiellement dangereux: impacter les autres applications, l'utilisateur ou l'OS
 - Elles ne peuvent lire / écraser les données privée de l'utilisateur
 - Elles ne peuvent lire les données d'autres applications
 - Elles ne peuvent accéder directement à Internet
 - Elles ne peuvent conserver le mobile actif,
 - ...



- Chaque application dispose de sa propre “sandbox”
 - Pour partager des ressources les applications doivent demander des permissions, et c'est l'utilisateur lors de l'installation qui accepte ou non ces permissions.
- Linux gère la sécurité



- Plusieurs variantes d'une même image
- Plusieurs orientations possibles (portrait / Paysage)