



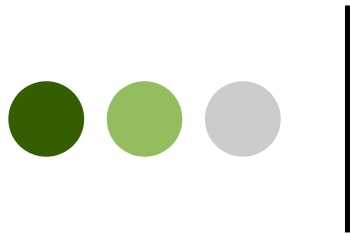
Développer une Application mobile *sous Android*

PJE M1 – Cedric Dumoulin

Présentation élaboré d'après le cours de :

Olivier Le Goaer

olivier.legoer@univ-pau.fr

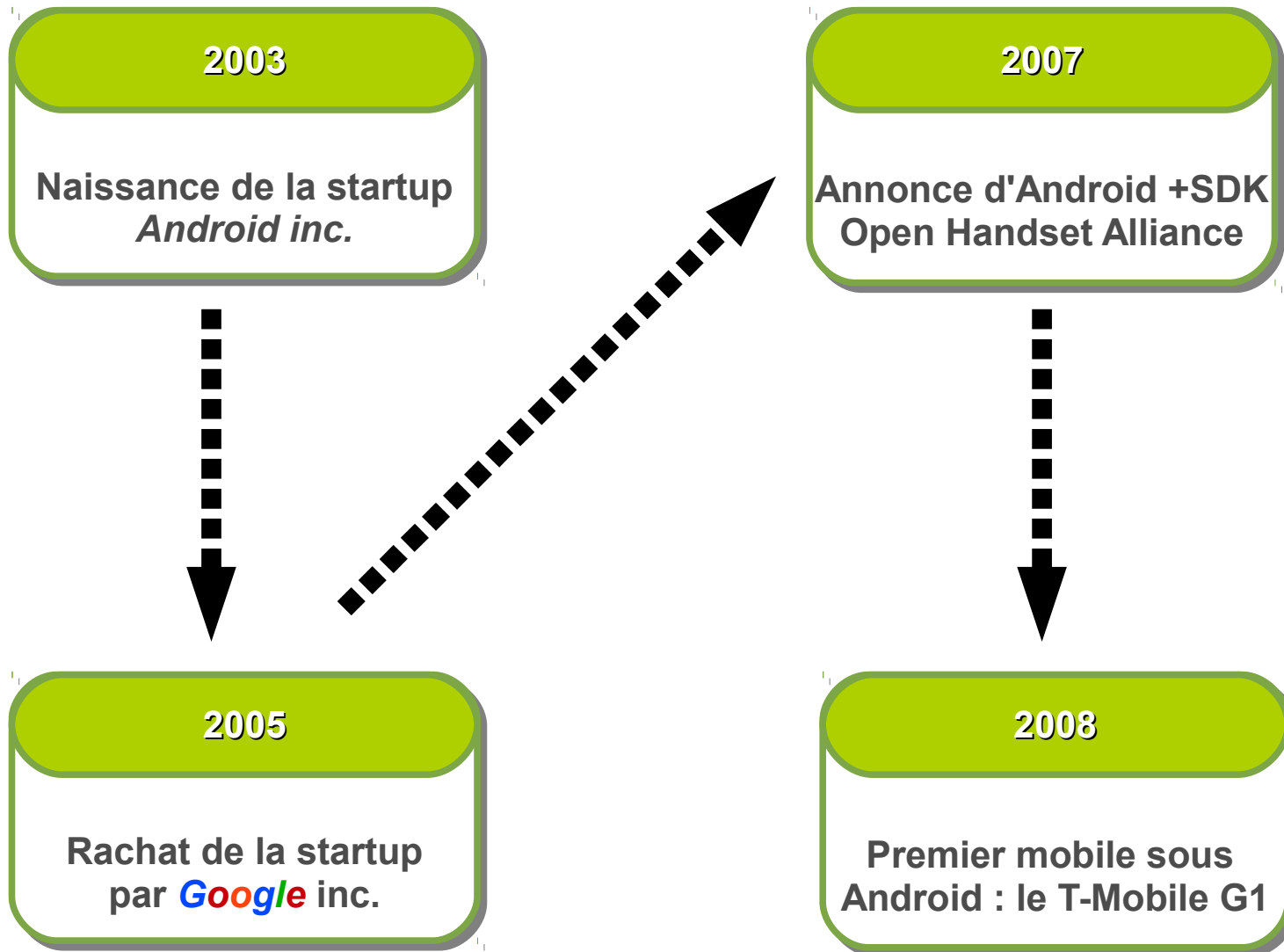


Remerciements

- Le cours et les transparents originaux sont de:
 - Olivier Le Goaer
 - olivier.legoer@univ-pau.fr
-
- Disponible sur le site developpez.com
 - <http://olegoer.developpez.com/cours/mobile/>
- Reproduit avec l'aimable autorisation de l'auteur.
- Copyright © 2013 Olivier LE GOAER. Aucune reproduction, même partielle, ne peut être faite de ce support et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur.



Bref historique



Voir annonce
par Sergey Brin
sur YouTube :
<http://goo.gl/LejV9>



●●● | Open Handset Alliance (OHA)

- Regroupement de + de 50 entreprises

- Fabricants de matériels
- Opérateurs mobile
- Développeurs d'applications
- ...



- Objectif :

- Développer des normes ouvertes pour les appareils de téléphonie mobile



Matériels Android

Tablette

APN

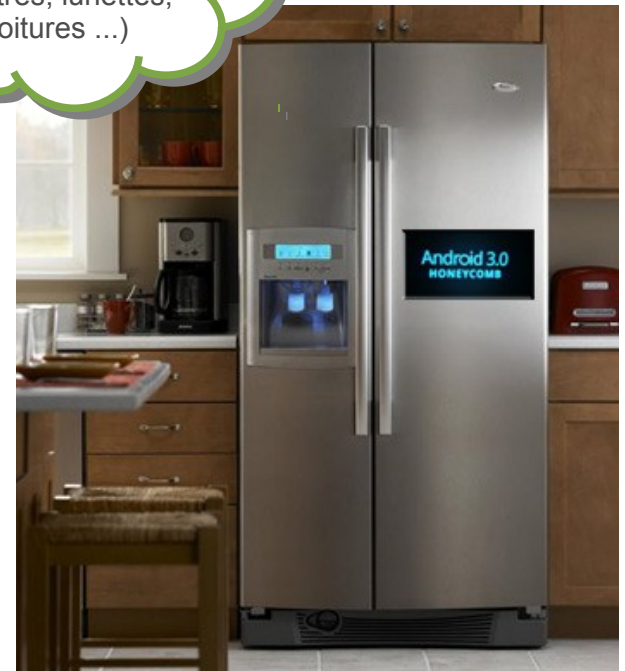
Smartphone



Embarqué
(électroménager,
montres, lunettes,
voitures ...)

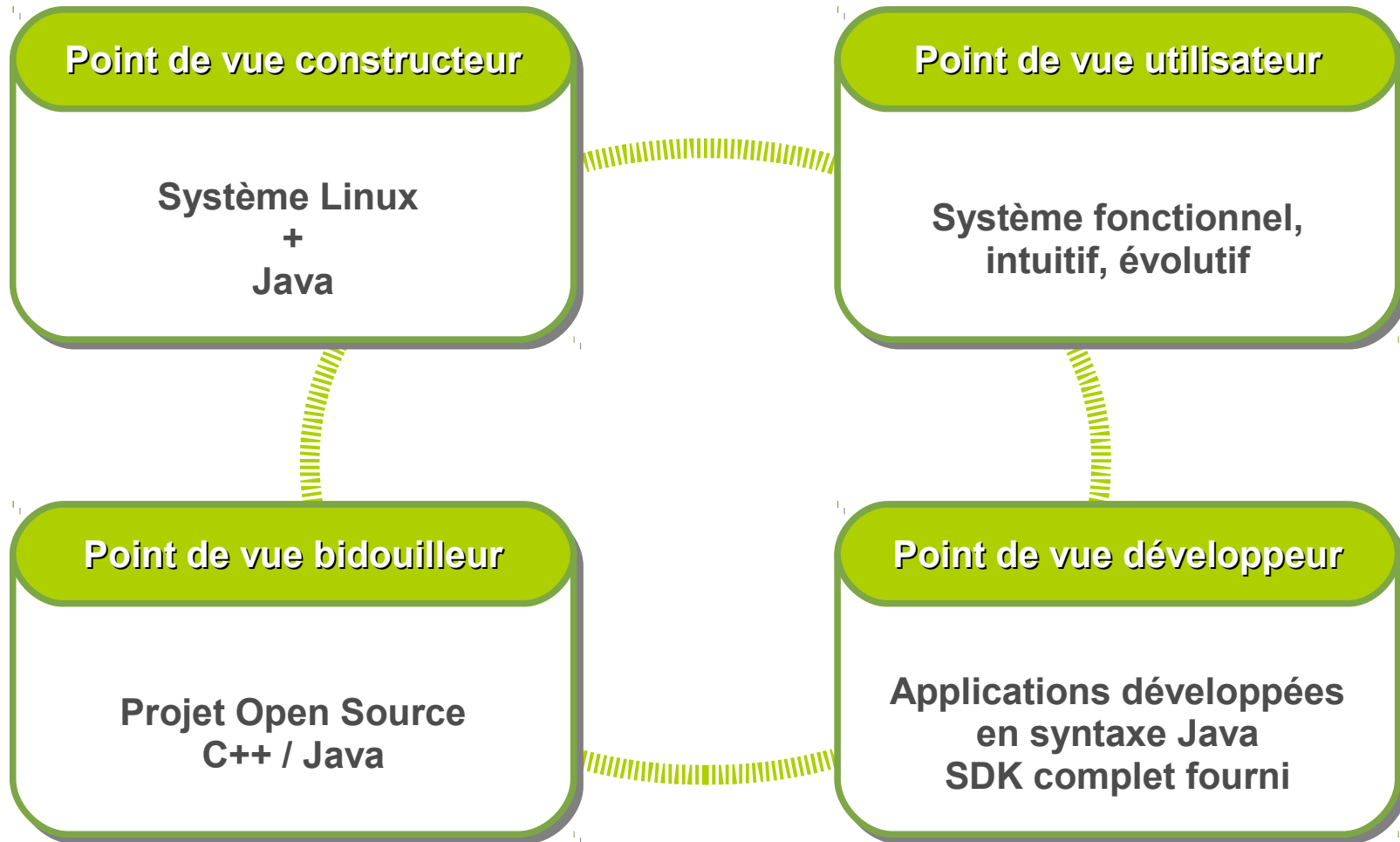
Netbook

Télévision



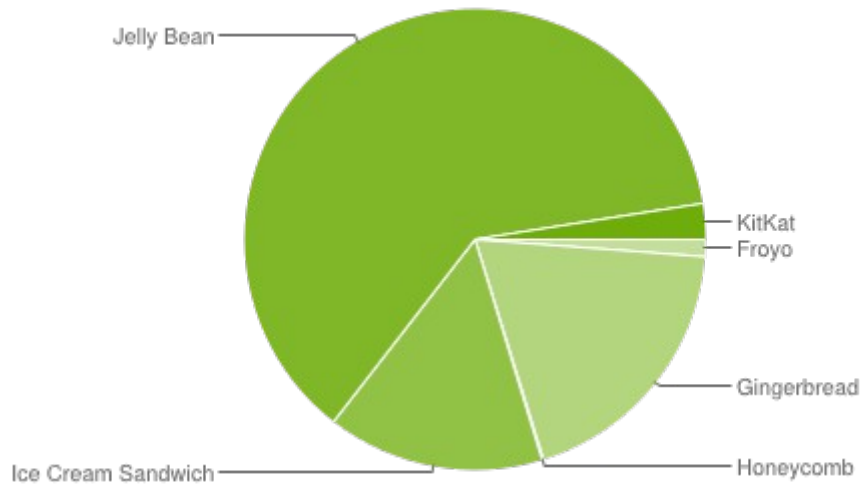


Points forts d'Android





Versions d'Android



March 3, 2014 – Google Inc

CodeName	Platform	API Level
<i>Cupcake</i>	Android 1.5	3
<i>Donut</i>	Android 1.6	4
<i>Eclair</i>	Android 2.1	7
<i>Froyo</i>	Android 2.2	8
<i>Gingerbread</i>	Android 2.3	9
<i>Honeycomb</i>	Android 3.0	11
<i>Ice Cream Sandwich</i>	Android 4.0	14
<i>Jelly Bean</i>	Android 4.1	16
<i>KitKat</i>	Android 4.4	19
Lollipop	Android 5.0	21
Marshmallow	Android 6.0	?
?	?	?



1.5
Cupcake



1.6
Donut



2.0/2.1
Eclair



2.2
Froyo



2.3
Gingerbread



3.1/3.2
Honeycomb



4.0
IceCreamSandwich



4.1
Jelly Bean



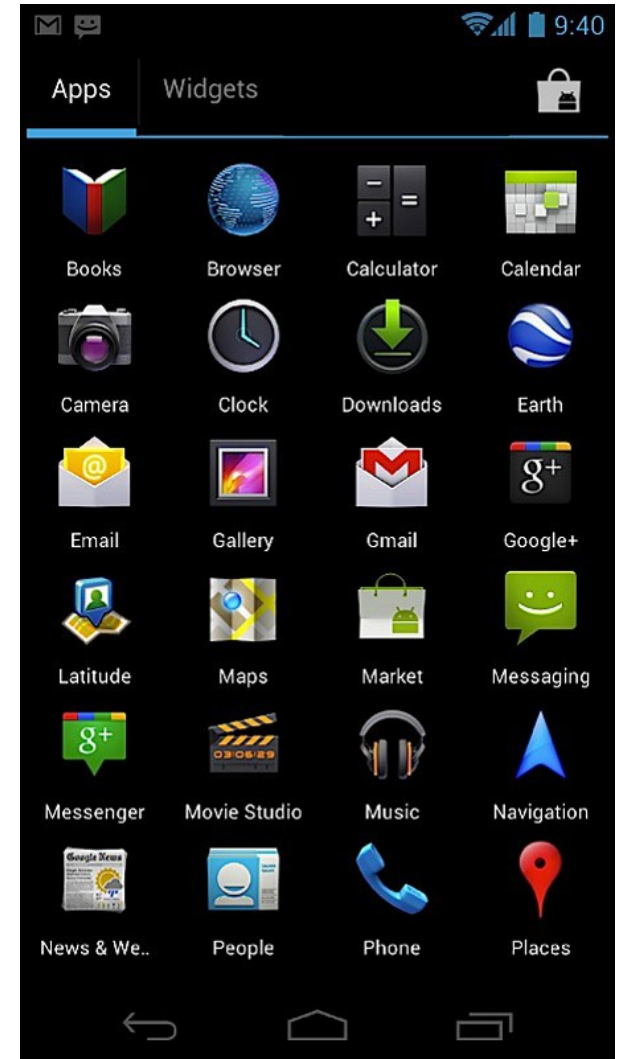
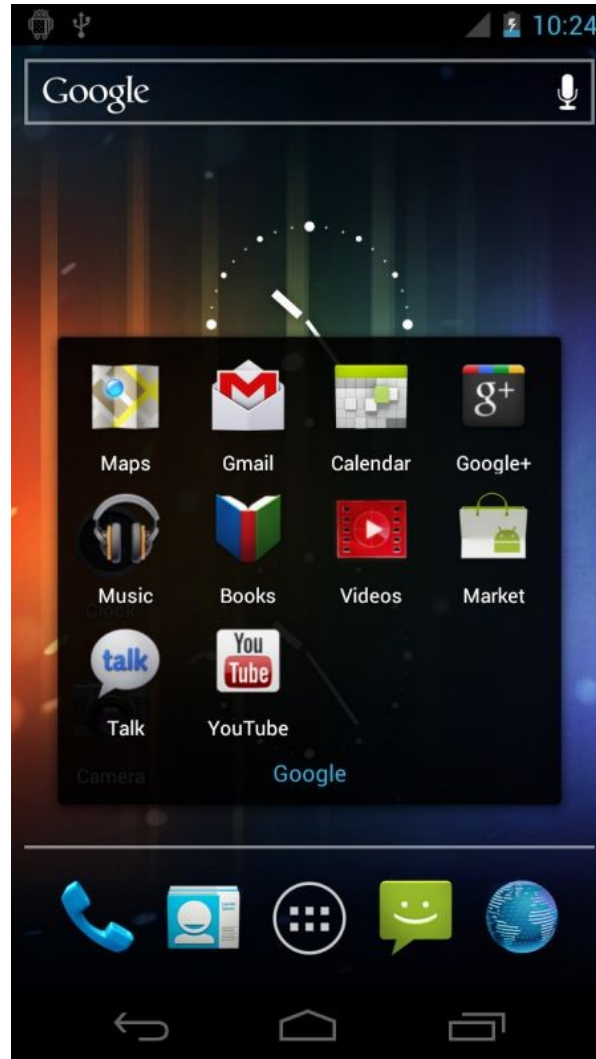
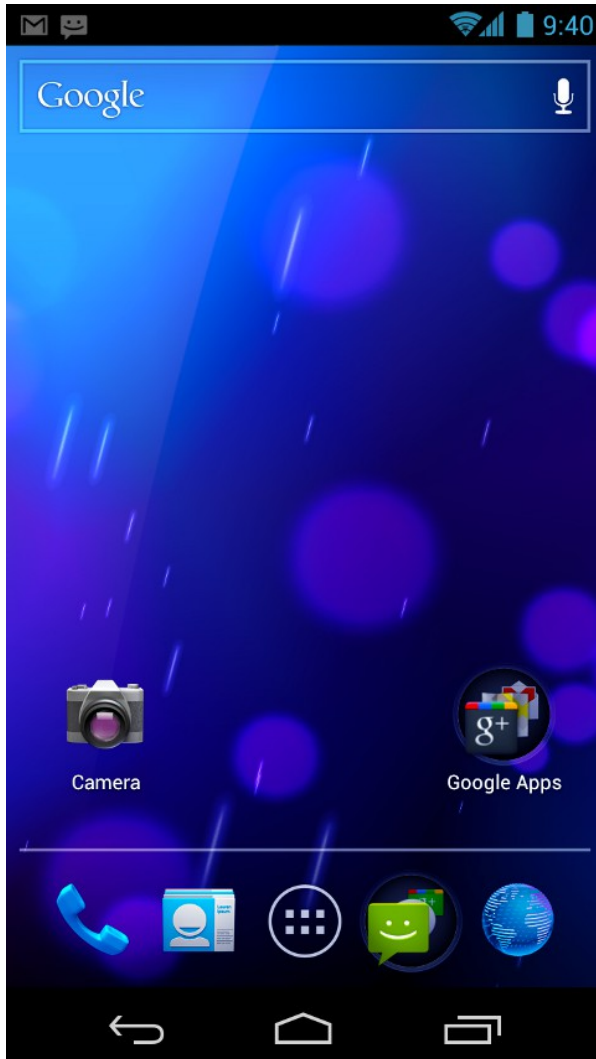
4.4



4.5

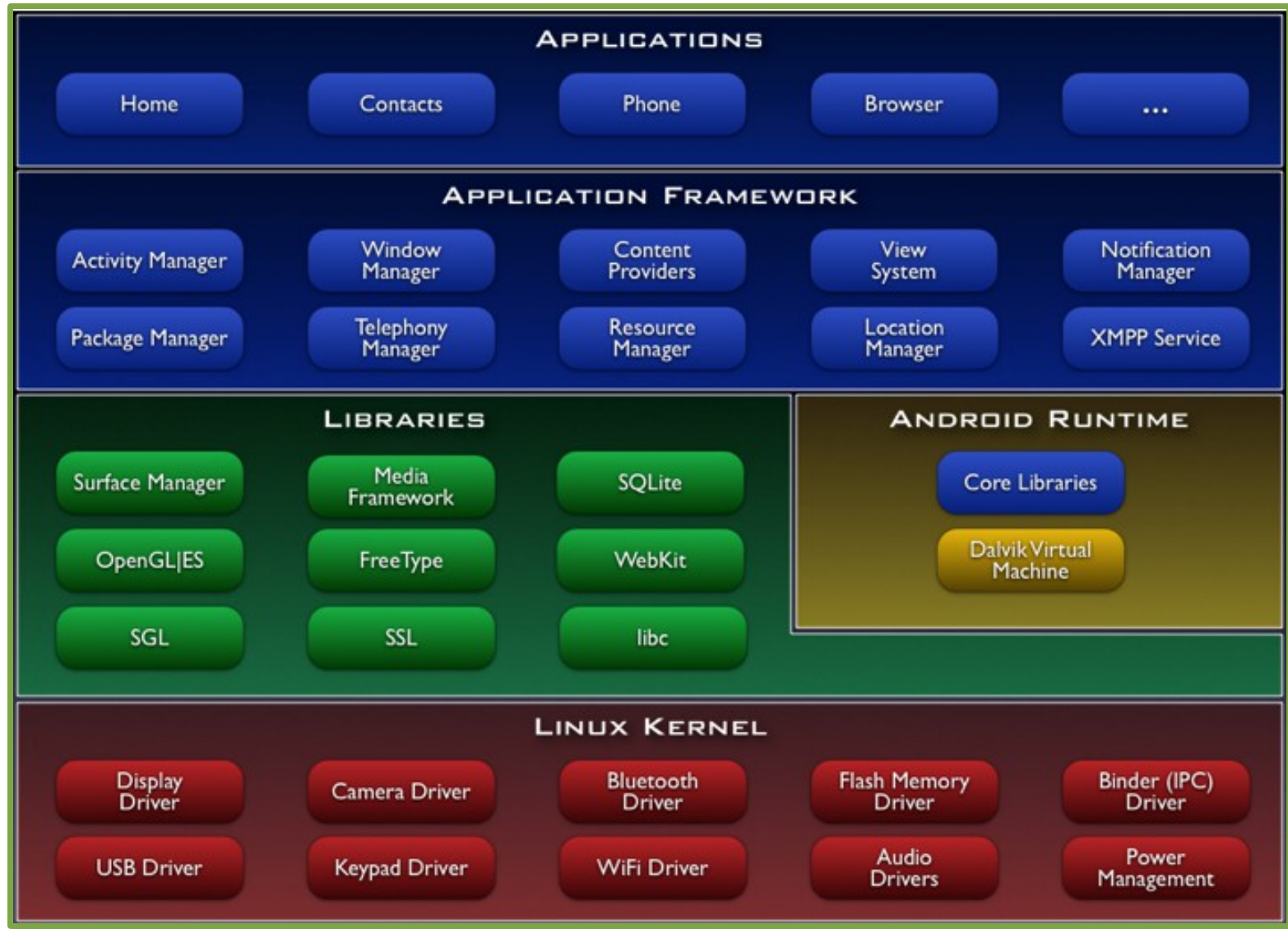


Android en images...



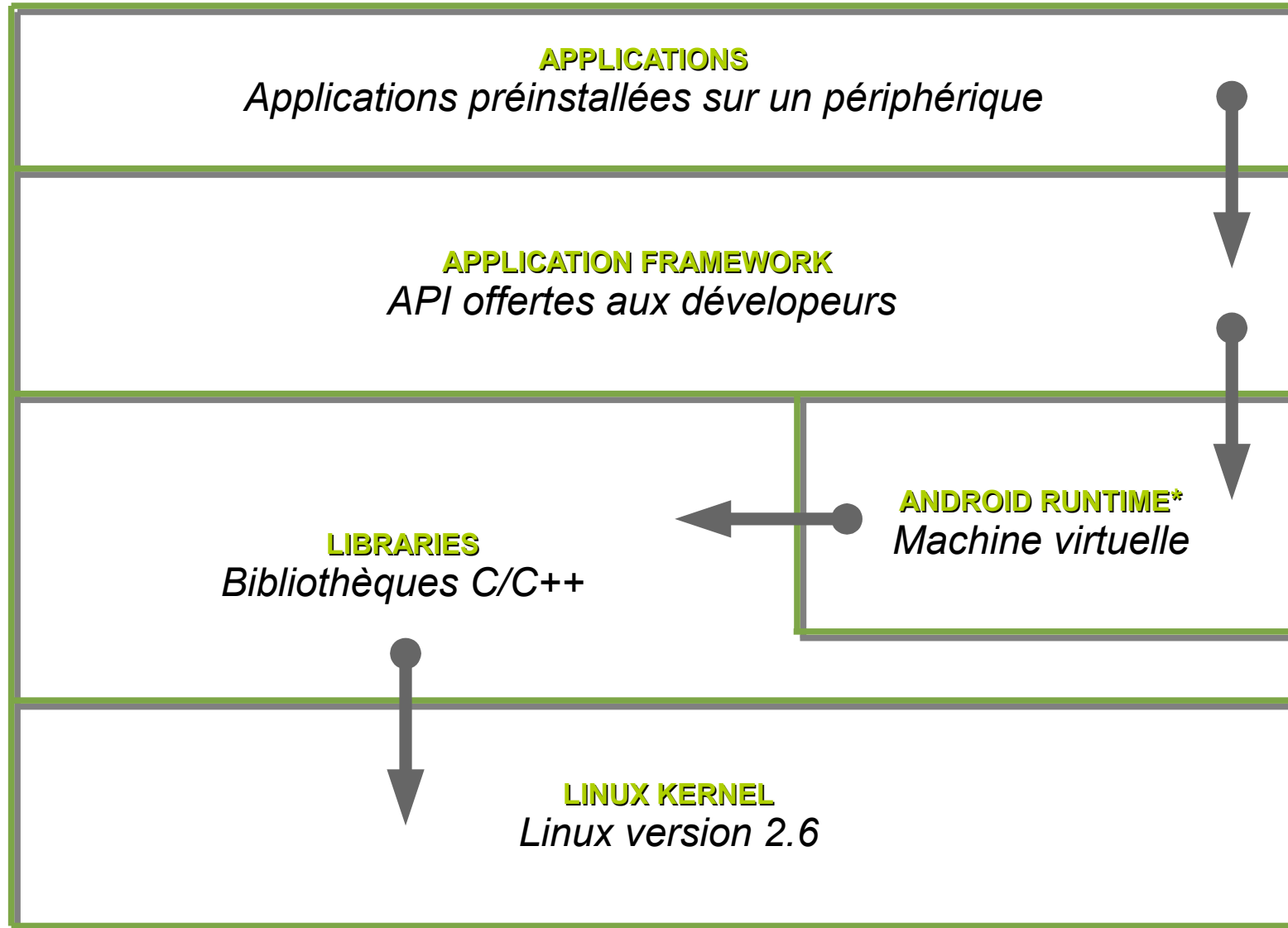


Architecture logicielle





Couches logicielles





Machine virtuelle "Dalvik"

- Offre l'avantage de toute machine virtuelle
 - Couche d'abstraction entre le développeur d'applications et des implémentations matérielles particulières
- La VM Dalvik n'est pas une VM Java
 - Tient compte des contraintes de CPU et mémoire
 - Exécute des fichiers `.dex` (*Dalvik Executable*) optimisés
- La VM crée une instance Dalvik pour chaque application (i.e. processus lourds)
 - Les applications sont totalement indépendantes ("*sandbox*")
 - Espaces protégés (mémoire, disque)
 - Évite un plantage généralisé !

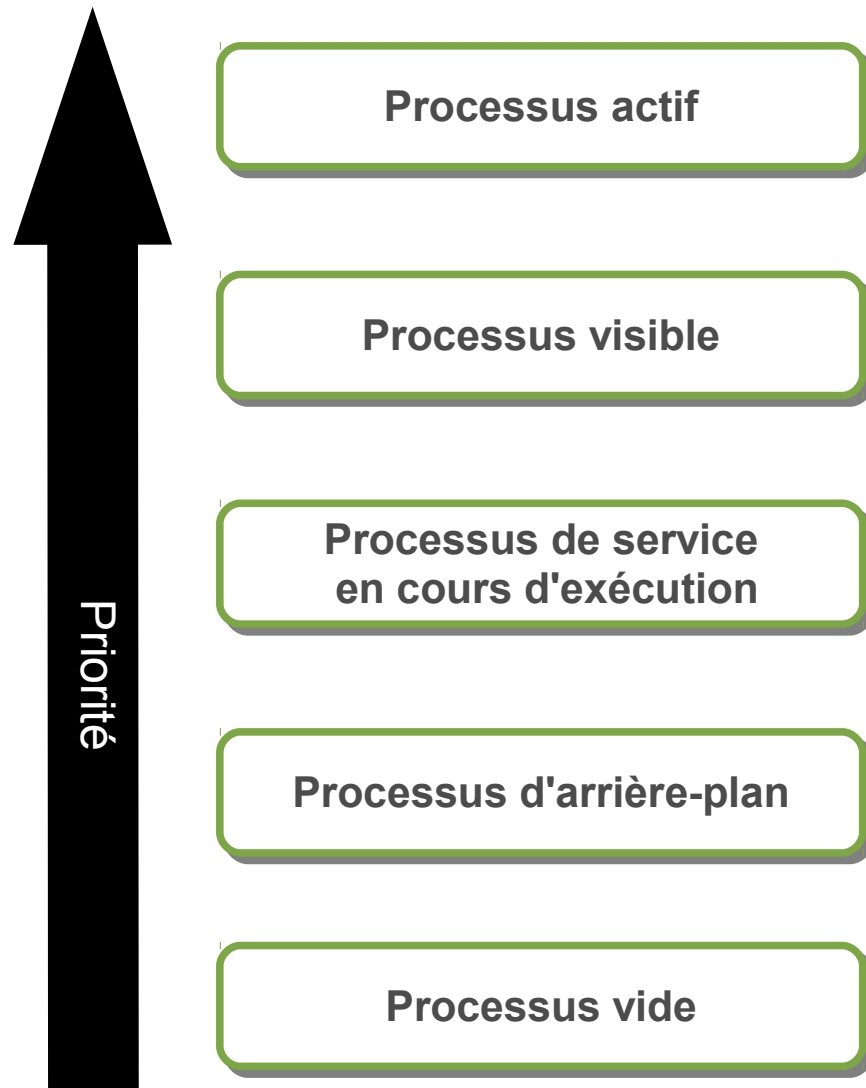


De Dalvik à ART...

- 1^{ère} génération : VM Dalvik
 - Machine à registres (versus à piles comme la JVM)
 - Compilation *Just-In-Time* (JIT)
 - Application partiellement compilée
 - Surcoûts liés à la compilation du bytecode à l'exécution
- 2^e génération : VM ART (*Android RunTime*)
 - Compilation *Ahead-Of-Time* (AOT)
 - Application précompilée entièrement lors de sa première installation
 - Execution 2 fois plus rapide qu'avec Dalvik
 - Temps d'installation rallongé
 - Taille des APK augmentée



Priorités des processus



- Android gère ses ressources de manière agressive
 - Pour garantir une haute réactivité
 - Élimination de processus sans avertissement !
 - Politique de priorité basée sur une hiérarchisation des processus

●●● | Le petit univers Java

- Concepts de la POO Java

- Paquetage, classe, annotation, héritage, interface...

- Idioms

- Classes anonymes, classes internes, POJO...

- Bibliothèques (API)

- J2SE (subset) : `java.io.*`, `java.lang.*`...
- Android : `android.view.*`, `android.telephony.*`...
- Google : `com.google.android.maps.*`...

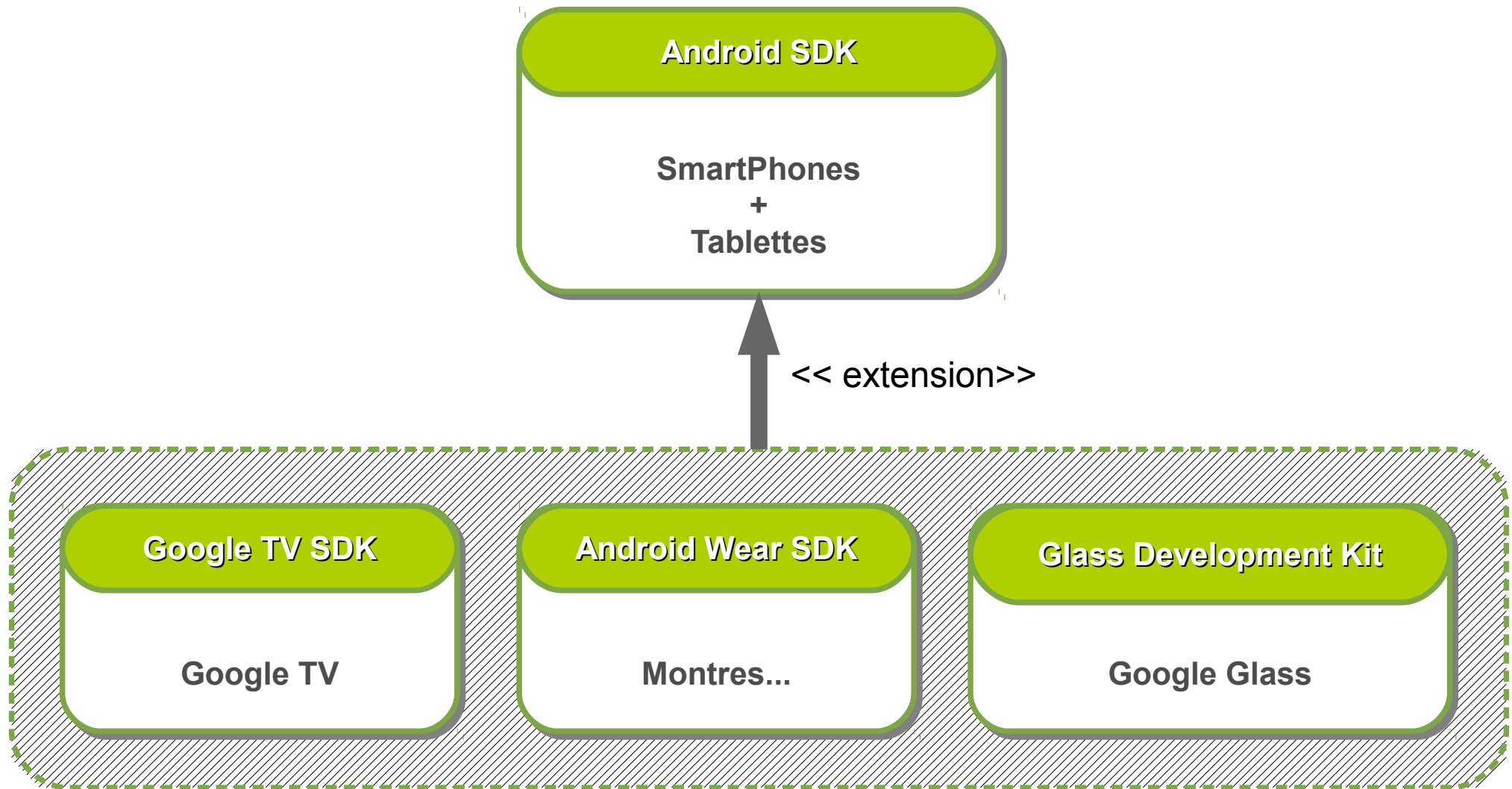


- Design patterns

- Singleton, Builder, Factory, Observer (Listener), DAO...




Kits de développement



● ● ● | Outils du développeur android developers

- Android Studio

- Assistant à la création de projets
- Créateur d'interface graphique (WYSIWYG)
- Vues et perspectives dédiées



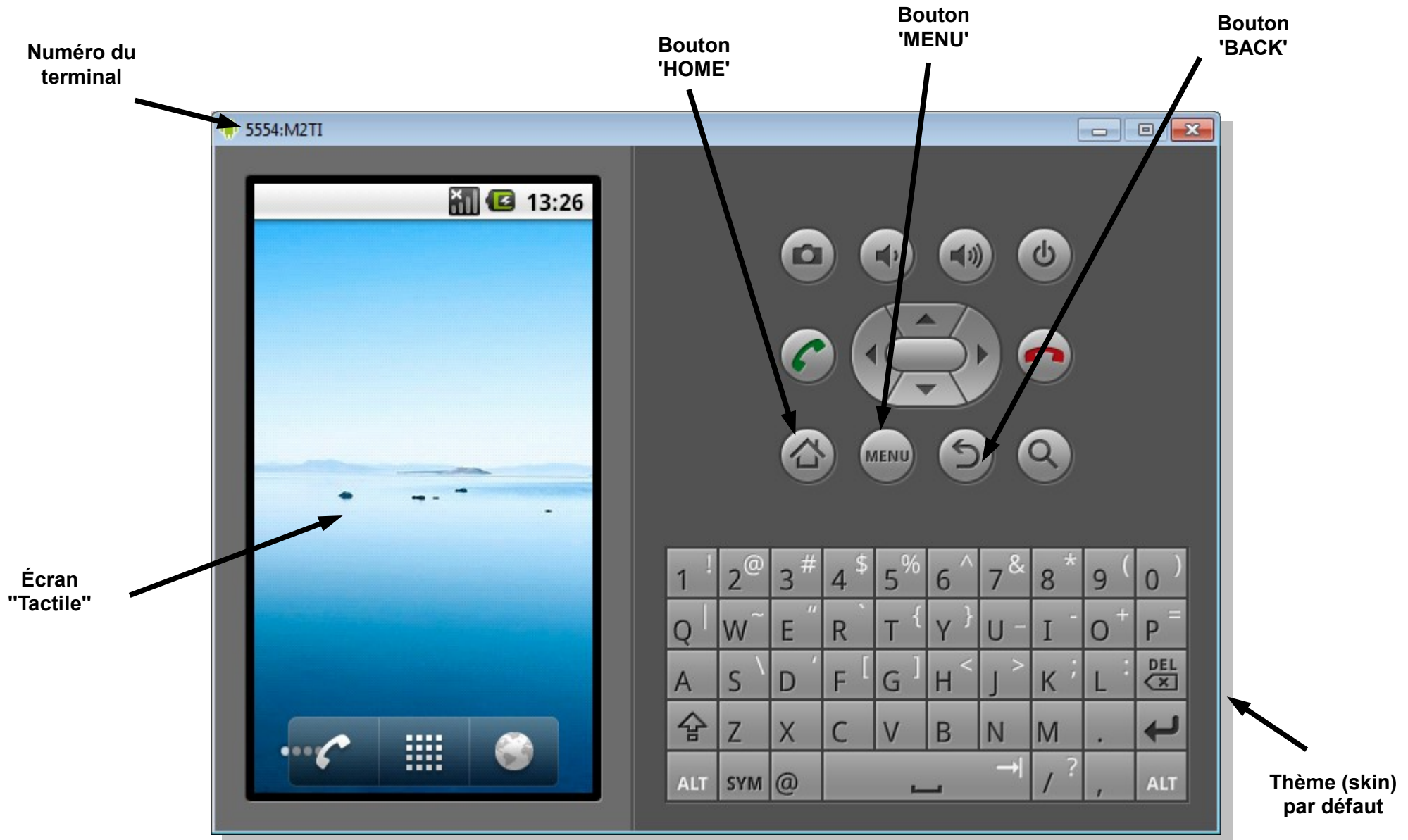
Alternative :
Eclipse ADT

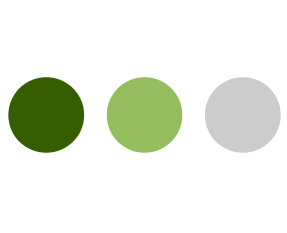
- Android SDK

- API
- Émulateurs
- Débogueur, compilateur, empaqueteur, signature, analyseur statique de code (Lint), obfuscateur (ProGuard)



Émulateur : aperçu





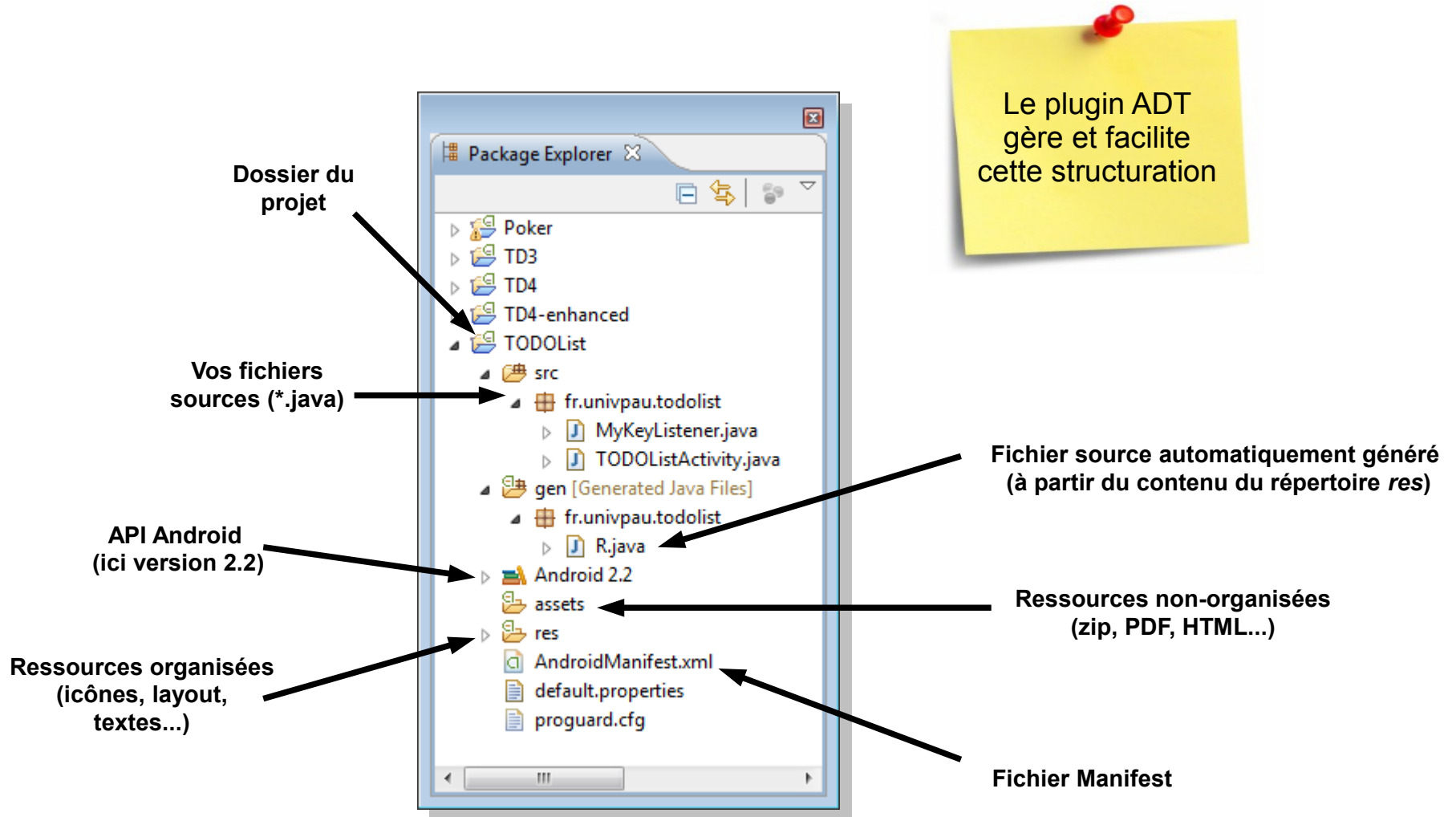
Émulateur : limitations

Alternatives :
• AndroVM
• GenyMotion

- Lent au démarrage et à l'usage
 - Emulation (i.e. niveau d'indirection) *versus* simulation
 - Noyau Linux + extensions android
- Fonctionnalités non-disponibles
 - Appareil photo (Camera↔Webcam supporté)
 - Vibreur
 - Appels téléphoniques réels
 - Capteurs en général
 - Connexions USB
 - Évolution de la charge de la batterie



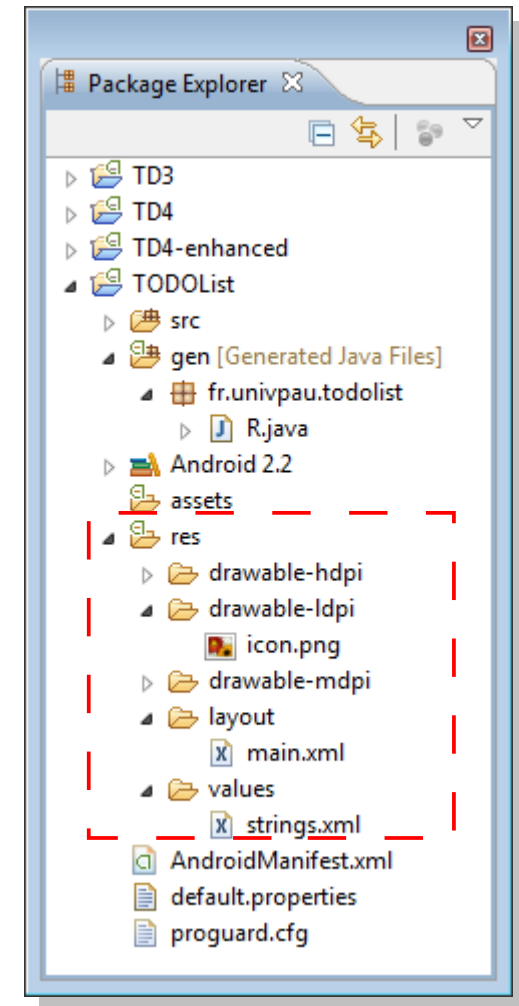
Arborescence projet





Externaliser les ressources

- Android prévoit *de facto* l'externalisation des ressources
 - Facilité de maintenance, de mise à jour et de gestion
- Créer des ressources
 - Simples valeurs, chaînes de caractères, couleurs, menu, layout, styles, images, etc.
 - Stockées dans les sous-répertoires de /res/ du projet
 - un seul niveau de profondeur est autorisé !





Compilation et déploiement

Sources Java



Bytecode Java



Bytecode Dalvik
(optimisé)



Ressources + Manifest



Application
empaquetée
+ signée

Entièrement
automatisé
avec le
plugin ADT
sous Eclipse !!



/data/app



Environnement contraint

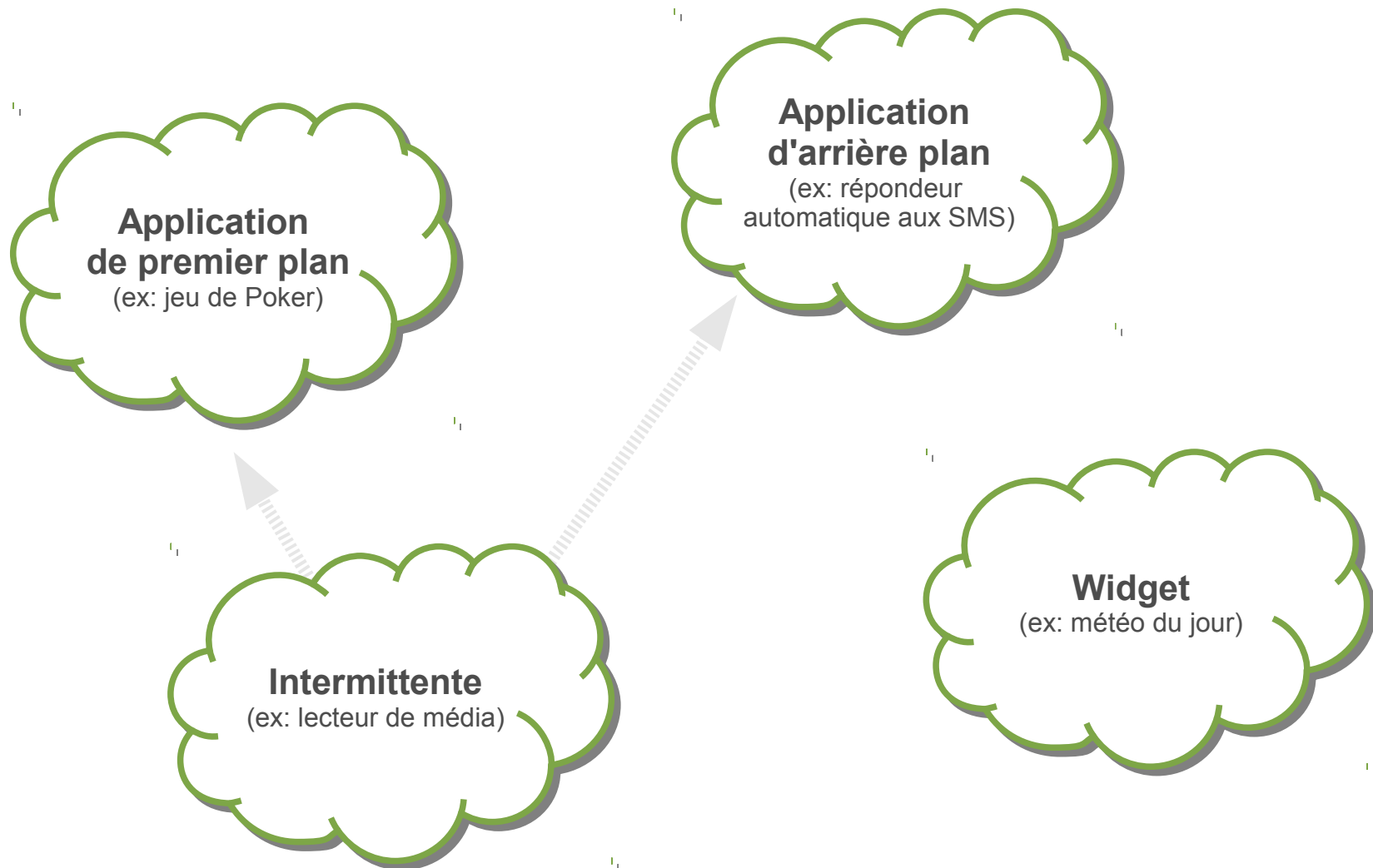
- Pour vos développements, gardez à l'esprit que les appareils mobiles ont :
 - Une puissance processeur plus faible
 - Une RAM limitée
 - Des capacités de stockage permanent limitées
 - De petits écrans avec de faibles résolutions
 - Des coûts élevés de transfert de données
 - Des taux de transfert plus lents avec une latence élevée
 - Des connexions réseau moins fiables
 - Des batteries à autonomie limitée



Construction d'une application Android

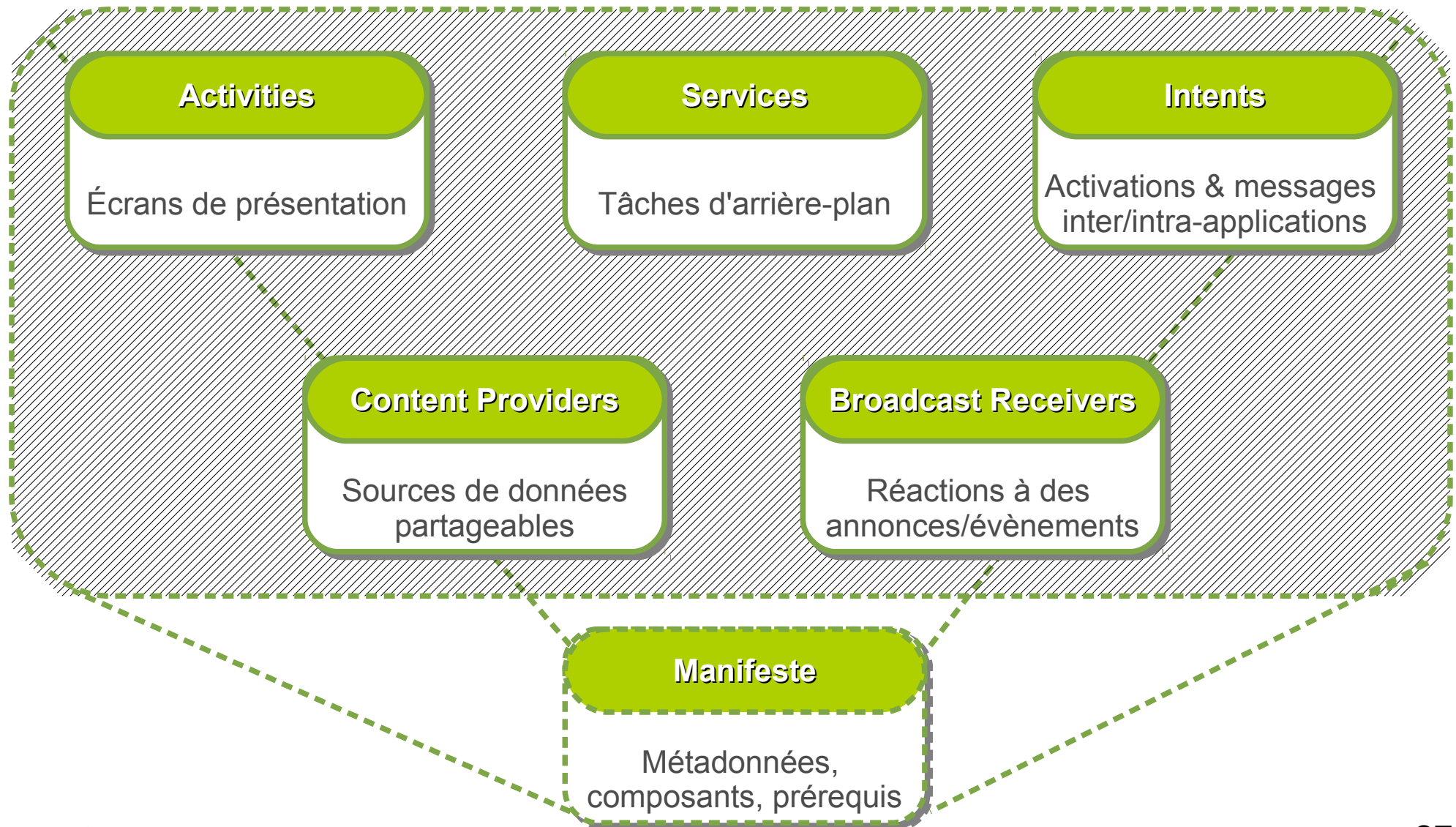


Types d'applications Android





Les différents composants





Le manifeste

- Chaque projet contient à sa racine un fichier `AndroidManifest.xml` qui :
 - Nomme le paquetage Java de l'application. Ce dernier sert d'identificateur unique de l'application.
 - Déclare les composants applicatifs (activities, services, broadcast receivers, content providers) de l'application et leurs filtres si nécessaire (à quels intents ils réagissent)
 - Déclare les permissions que l'application doit avoir pour fonctionner (droit de passer des appels, droit d'accéder à Internet, droit d'accéder au GPS...)
 - Déclare le niveau minimum de compatibilité du SDK pour que l'application fonctionne



AndroidManifest.xml

Pensez à bien déclarer **tous** les composants applicatifs de votre application !

Permissions requises

Nœud de l'application

Nœud d'une activité de l'application

Nœud d'un service de l'application

Nœud d'un broadcast receiver de l'application

Nœud d'un content provider de l'application

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="fr.univpau.bankster">

  <uses-permission />
  <uses-sdk />
  <supports-screens />

  <application>

    <activity>
      <intent-filter>
        <action />
        <category />
        <data />
      </intent-filter>
    </activity>

    <service>
      <intent-filter> . . . </intent-filter>
    </service>

    <receiver>
      <intent-filter> . . . </intent-filter>
    </receiver>

    <provider>
      <grant-uri-permission />
    </provider>

  </application>
</manifest>
```

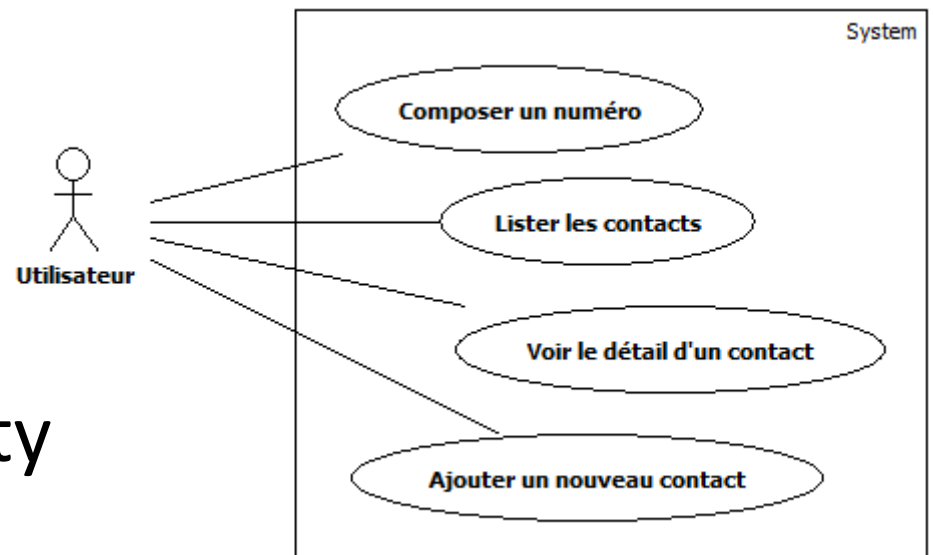


Notion de contexte

- Le contexte modélise les informations globales sur l'environnement de l'application
- Possède les méthodes importantes
 - getResources, getPackageName, getSystemService...
 - startActivity, startService, sendBroadcast, getContentResolver...
 - openFileInput, openOrCreateDatabase, getSharedPreferences...
- Accès au contexte
 - Depuis une Activity ou un Service : this (car héritage)
 - Depuis un BroadcastReceiver : en argument de onReceive()
 - Depuis un ContentProvider : this.getContext()

● ● ● | **Activité**

- Une activité \approx un écran graphique
 - Incarne souvent un cas d'utilisation (use case UML)
- Une application est formée de n activités
- Exemple : application de téléphonie
 - 1) Numéroteur
 - 2) Annuaire des contacts
 - 3) Fiche d'un contact
 - 4) Ajout d'un contact
- Étend `android.app.Activity`

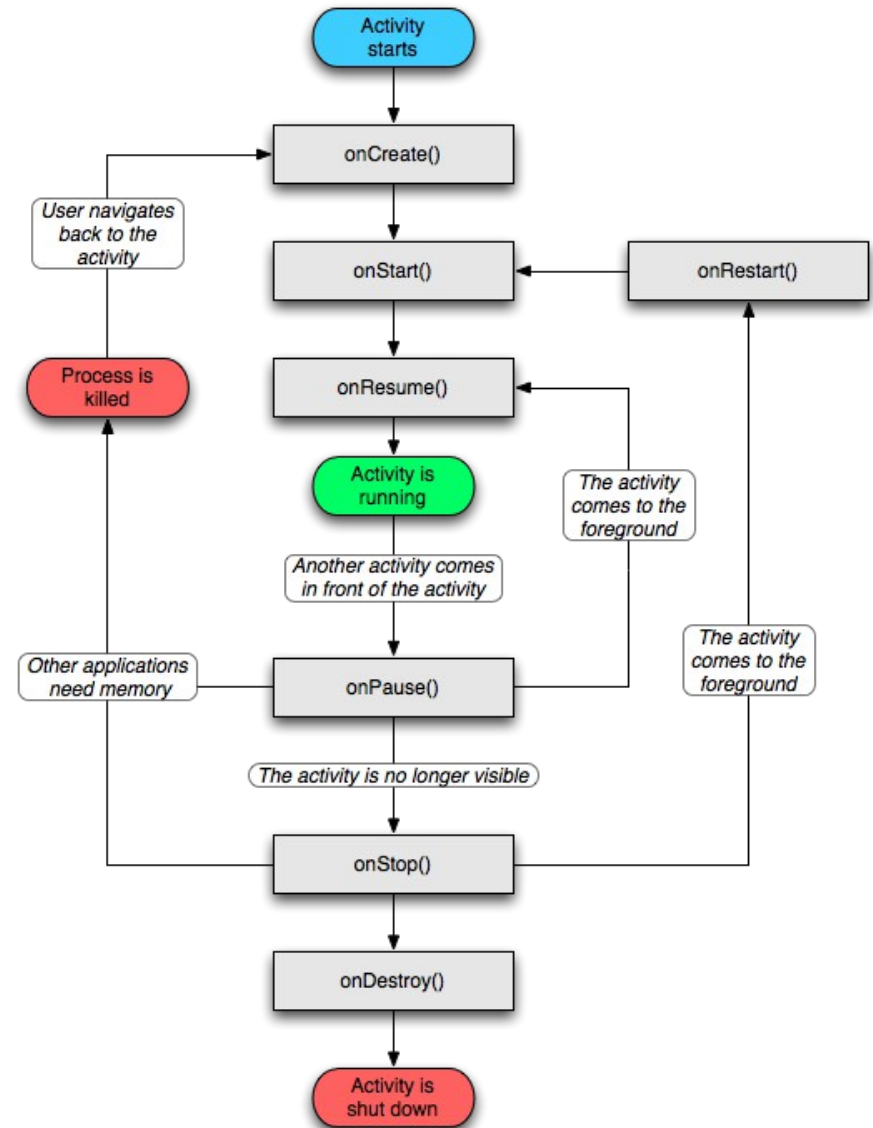




Cycle de vie d'une activité

● 7 méthodes de callback

- void onCreate(...)
- void onStart()
- void onRestart()
- void onResume()
- void onPause()
- void onStop()
- void onDestroy()





Activité : code source

```
package fr.univpau.bankster;

import android.app.Activity;

public class Home extends Activity {

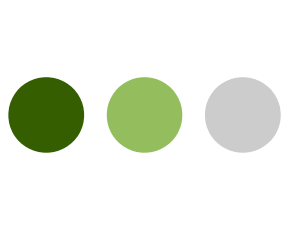
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* Allocation des ressources ici */
    }

    @Override
    public void onResume() {
        /* Préparation des vues ici */
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        /* Désallocation des ressources ici */
    }
}
```

Le bundle sert à mémoriser l'état de l'UI de l'activité lorsqu'elle passe en arrière plan

Evitez le «syndrome onCreate»
Ne pas mettre tout son code dans cette méthode !

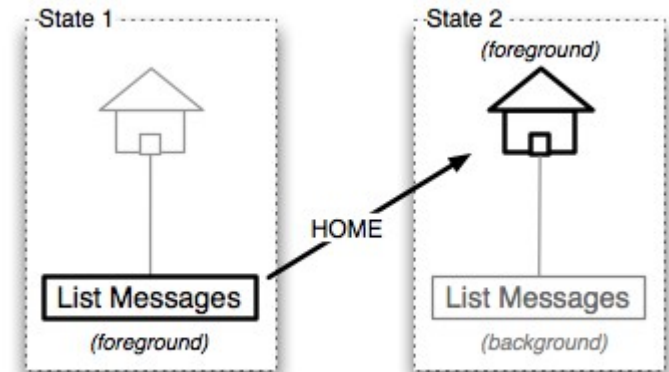
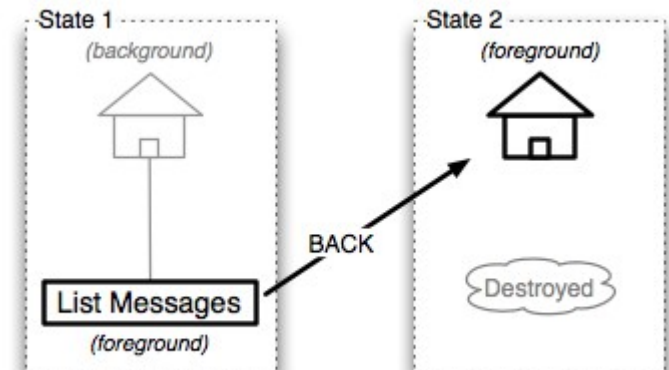


Lancer une activité

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

La pile des activités (BackStack)

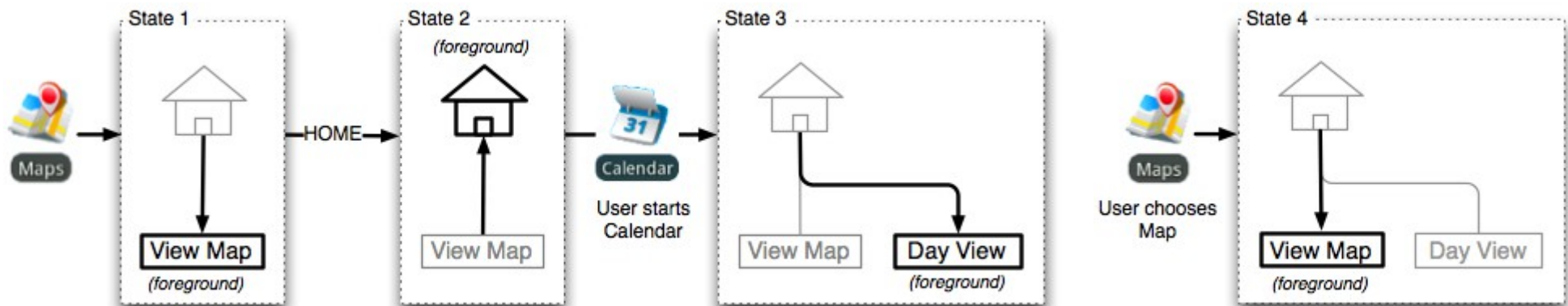
- Les activités sont empilées/dépilées
 - Empilée quand une activité démarre
 - Dépilée (i.e. détruite) quand on presse le bouton 'BACK'
- Une pression sur le bouton 'HOME' ne dépile pas l'activité.
 - Elle passe simplement en arrière plan





Multitâches

- Plusieurs piles d'activités peuvent coexister avec Android
 - L'utilisateur passe de l'une à l'autre



Reprend l'activité
située au
sommet de
la pile



Service

- Sert à effectuer des opérations ou des calculs en dehors de l'interaction utilisateur
 - Ne nécessite pas une interface graphique
- Deux types de services :
 - Local : service qui s'exécute dans le même processus que votre application
 - Distant (IPC) : service qui s'exécute dans des processus indépendants de votre application (nécessite une description AIDL)
- Un service est une classe qui étend
 - `android.app.Service`

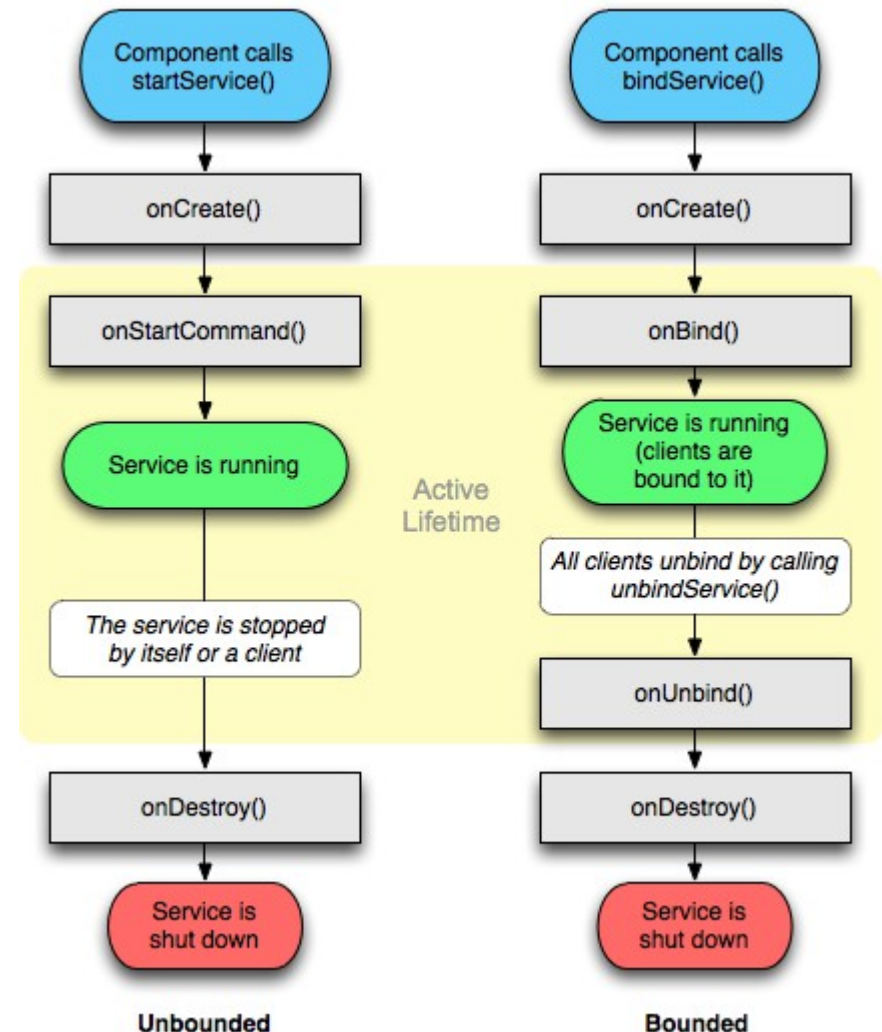


Cycle de vie d'un service

- 5 méthodes de callback

- onCreate()
- onStart() *SDK<2.0*
- onStartCommand()
- onDestroy()
- onBind()
- onUnbind()

- S'exécute dans le processus courant





Service : code source

```
package fr.univpau.bankster;

import android.app.Service;

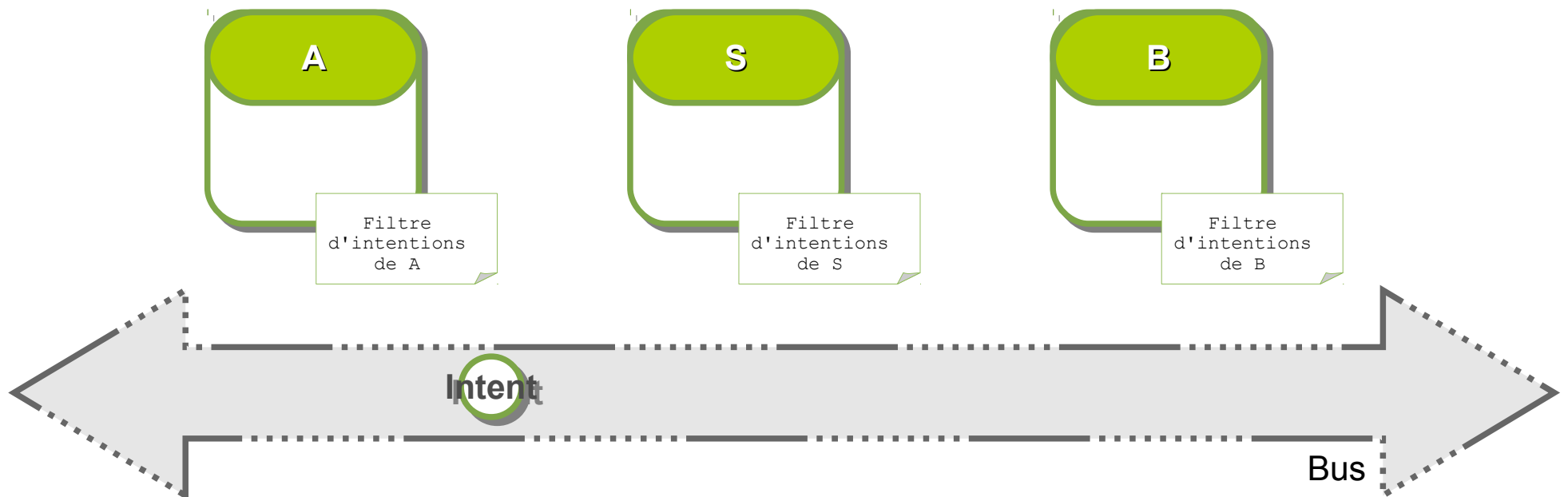
public class AccountCleaner extends Service {

    @Override
    public void onCreate() {
        /* Allocation des ressources ici */
    }
    @Override
    void onStartCommand(Intent intent,
                        int flags, int startId) {
        /* Votre code du service ici */
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        /* Désallocation des ressources ici */
    }
}
```



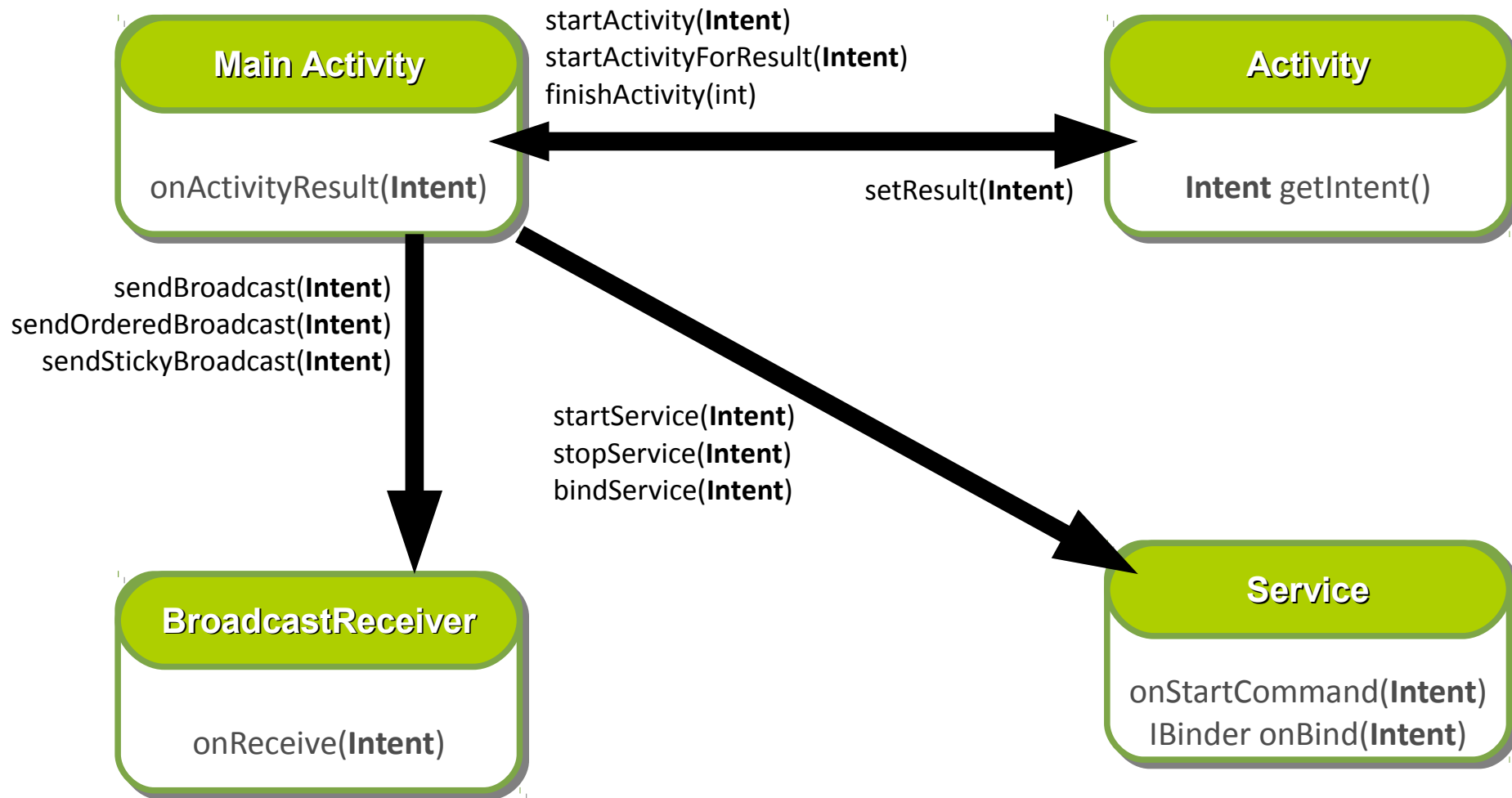
Intents : principes

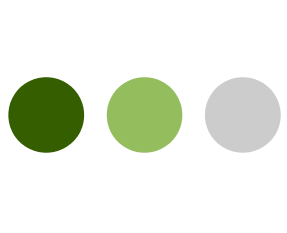
- Trois types de composants applicatifs sont activés via des intentions (*intents*)
 - Activity, Service et BroadcastReceiver
- Principe d'un bus à messages (*middleware*)





Intents : vue d'ensemble





Natures des intents

● Types

1. Direct

- Instance de `android.content.Intent`
- Le composant cible est activé par le composant source

2. Par procuration

- Instance de `android.content.PendingIntent`
- Le composant cible est activé par un élément tiers, de la part du composant source (toutes ses permissions lui sont cédées pour l'occasion)

● Désignations

1. Explicite

- Le composant cible est nommé
- « *Activer tel composant* »

2. Implicite

- Le composant cible n'est pas nommé
- « *Activer un composant capable de prendre en charge cette action, sur cette donnée* »
- Les filtres d'intention indiquent au bus si le message d'activation doit être délivré au composant



Intents : les extras

- Il est possible d'utiliser l'intent pour convoyer des données supplémentaires
 - Une à une, ou regroupées dans un bundle (android.os.Bundle)
- Stockage sur le principe d'une table de hash
 - Méthode putExtra surchargée
 - Les types de base (+array) sont gérés
 - Les types complexes (c-a-d les classes) doivent implémenter Parcelable, ou Serializable
- Récupération
 - Méthode propre à chaque type, de la forme getXXXExtra()



Intent : code source

```
package fr.univpau.bankster;

import android.app.Activity;
import android.content.Intent;

public class Home extends Activity {

    void goToNextScreen() {

        Intent i = new Intent();    /* Intent de type direct */
        i.putExtra("happy", false); /* Donnée additionnelle */

        if(je_connaiss_la_cible) {
            /* Désignation explicite (implémentation) */
            i.setClass(Home.this, NextActivity.class);
        } else {
            /* Désignation implicite (action + data) */
            i.setAction(Intent.ACTION_DIAL);
            i.setData(Uri.parse("tel:01-56-60-12-34"));
        }
        this.startActivity(i); /* Poussé sur le bus */
    }
}
```



Actions et URI courantes


Action	URI	Signification
ACTION_EDIT	content://contacts/people/125	Éditer la fiche du contact 125
ACTION_VIEW	geo:49.5000,123.5000	Ouvrir l'application de géolocalisation à la position donnée (latitude, longitude).
ACTION_CALL	tel:0156601234	Appeller le numéro
ACTION_VIEW	google.streetview:cbll=49.5000,123.5000	Ouvrir google street view à la localisation donnée
...

- Actions natives android

- Constantes de la classe android.content.Intent

- Format d'une URI

- `scheme://host:port/path`
- schemes usuels : http, mailto, tel, mms, geo, file...



Voir aussi les
Intents « libres »
(www.openintents.org)



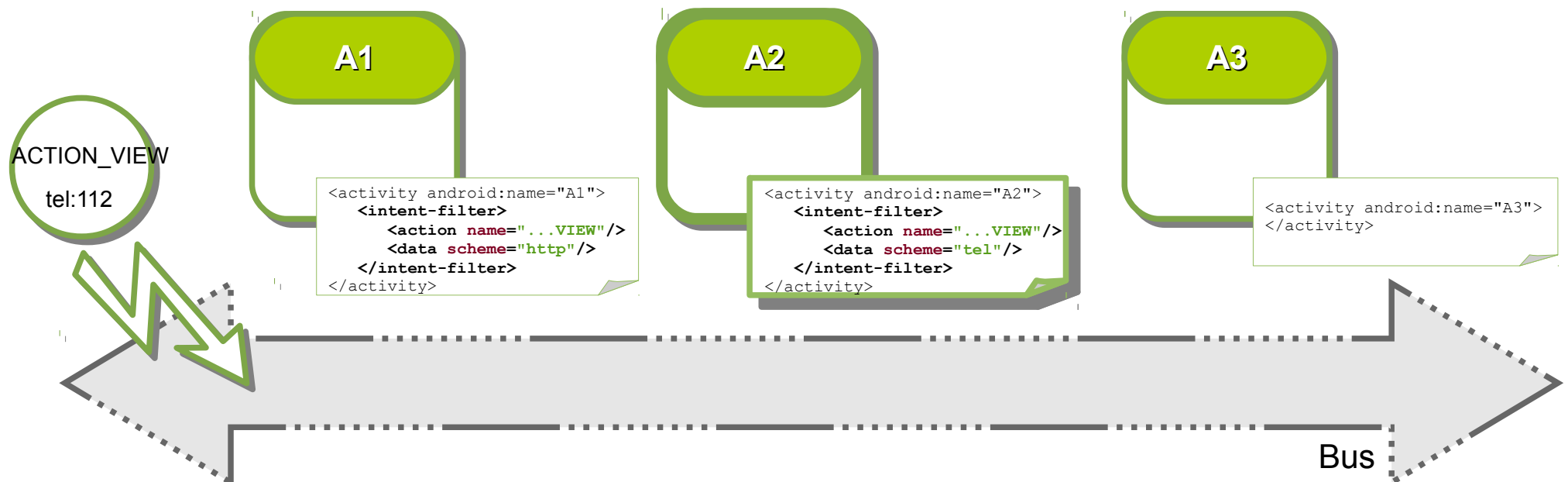
Filtres d'intention

- Chaque composant s'enregistre auprès du bus via ses filtres d'intention dans le manifeste
 - ACTION : Quelles actions sont supportées ?
 - DATA : Pour des données de quelle nature ?
 - CATEGORY : Dans quelles circonstances ?
- Permet au bus de savoir si le message d'activation (i.e. l'intent) doit être délivré au composant ou non
- L'absence de filtres implique une désignation explicite du composant



Résolution des intentions

- Tout repose sur le mode de désignation choisi
 - Explicite : le composant cible est exécuté (i.e. instancié)
 - Implicite : 0, 1 ou plusieurs composants sont éligibles
 - Dans le cas spécifiques des activités, comme une seule d'entre elles peut être exécutée à la fois, l'utilisateur sera invité à faire un choix





Interface graphique utilisateur (GUI)



Principe de R.java

- Contient les ID des **Resources**
- Chaque élément défini dans le répertoire */res* impacte le fichier R.java (à ne pas toucher)
 - Génération automatique de classes internes à la classe R, ainsi que des constantes de type entier sur 32 bits
- Chemin d'accès aux ressources via R.java
 - user-defined : `fr.univpau.foo.R.color.rose_bonbon`
 - system-defined : `android.R.color.darker_gray`
- Objet java représentant les ressources
 - Instance de la classe `android.content.res.Resources`
 - Ressources du projet en cours : `context.getResources()`



R.java

```
public final class R {  
    public static final class attr {  
    }  
    public static final class drawable {  
        public static final int icon=0x7f020000;  
    }  
    public static final class id {  
        public static final int editText1=0x7f050000;  
        public static final int listView1=0x7f050001;  
    }  
    public static final class layout {  
        public static final int main=0x7f030000;  
    }  
    public static final class string {  
        public static final int app_name=0x7f040001;  
        public static final int hello=0x7f040000;  
    }  
}
```

Le fichier R permet
de tirer parti d'une
vérification
syntaxique par
le **compilateur**

Le fichier R permet
de tirer parti d'une
complétion
automatique
de code dans
l'éditeur



Ressources organisées

- menu/

- Fichiers XML qui décrivent des menus de l'application

- raw/

- Fichiers bruts (ex: .mp3)

- values-*[qualifier]*/

- Fichiers XML pour différentes sortes de ressources (textes, styles, dimensions, ...)

- xml/

- Divers fichiers XML qui servent à configurer l'application (préférences, infos BDD, ...)

- anim/

- Fichiers XML qui sont compilés en objets d'animation

- color/

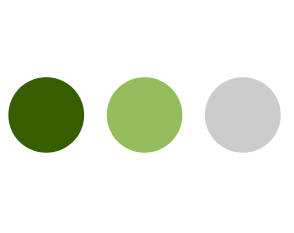
- Fichiers XML qui décrivent des couleurs

- drawable-*[qualifier]*/

- Fichiers bitmap (PNG, JPEG, or GIF), 9-Patch, ou fichiers XML qui décrivent des objets dessinables

- layout/

- Fichiers XML compilés en vues (écrans, fragments, ...)



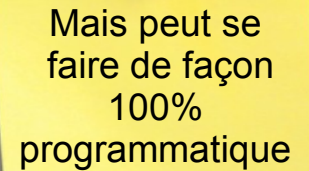
Exploiter les ressources

- 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/resourselident"`
 - `<EditText android:textColor="@color/rose_bonbon"/>`
 - `<EditText android:textColor="@android:color/darker_gray"/>`



UI : User Interface

- Une API Java riche
 - Des layouts et des widgets (appelés « Vues »)
- Programmation déclarative à la XML
 - Sépare la vue du code métier
- Fonctionnalité de personnalisation
 - Hériter et redéfinir un widget de base
 - Combiner des widgets existants
 - Dessin 100% personnalisé - View::onDraw(Canvas canvas)
- Rendu 2D/3D (non abordé dans ce cours)
 - OpenGL, Renderscript

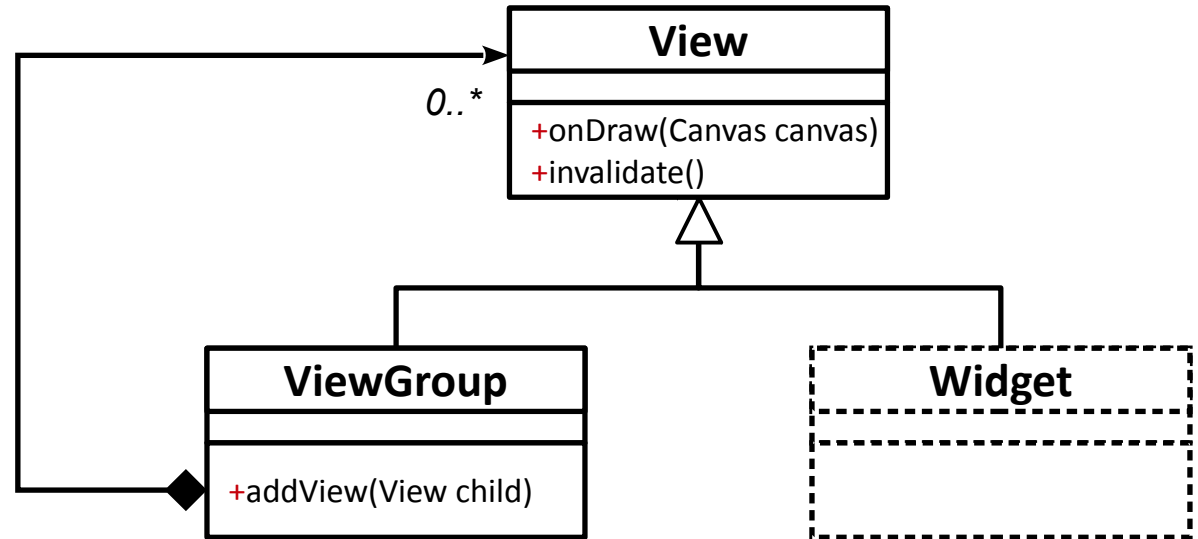


Mais peut se
faire de façon
100%
programmative



Les Vues Android

- Principe du design pattern *Composite* (i.e. un arbre)



- Nœud (ViewGroup)

- LinearLayout
- TableLayout
- RelativeLayout
- FrameLayout
- ScrollView

- Feuille (Widget)

- Button
- EditText
- TextView
- Spinner
- CheckBox



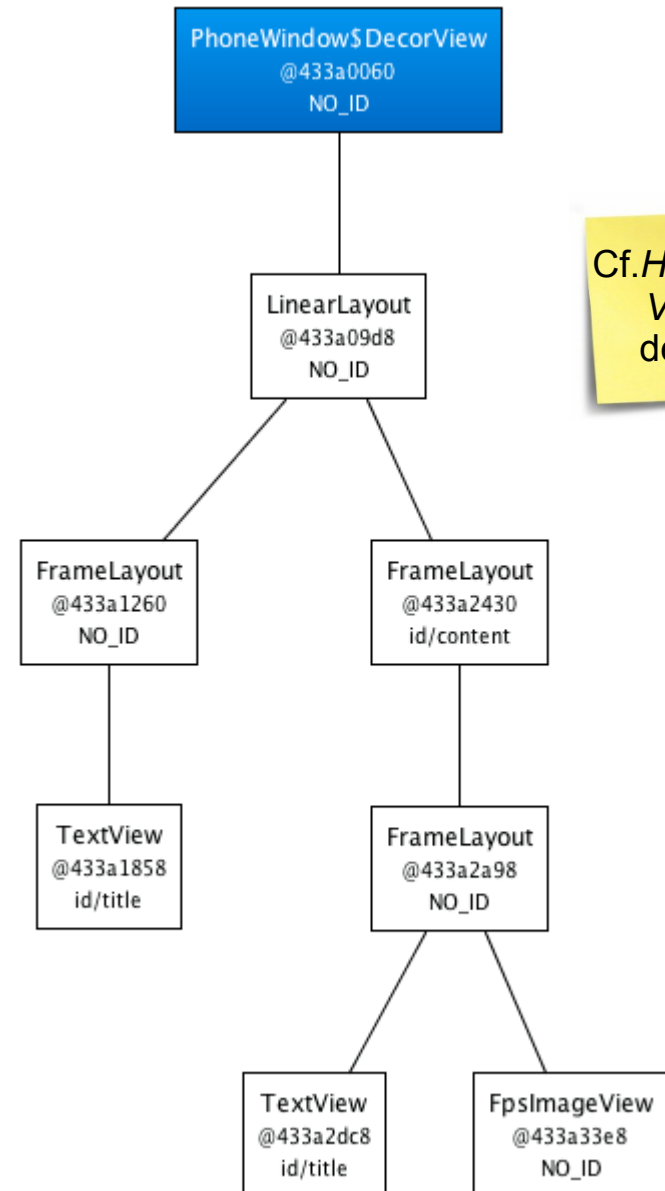
Arborescence de vues

- Affecter une arborescence à un écran

- Activity.setContentView() avec la racine de l'arbre

- Récupération d'une vue dans l'arbre par son Id

- Activity.findViewById()
 - recherche à partir de la racine définie pour l'écran (cf. ci-dessus)
- View.findViewById()
 - recherche à partir du nœud courant



Cf. *Hierarchy Viewer* de ADT



Quelques ViewGroup...



Linear Layout



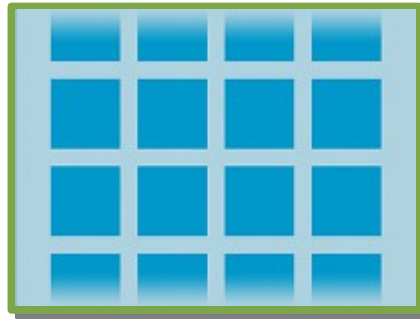
Relative Layout



Web View



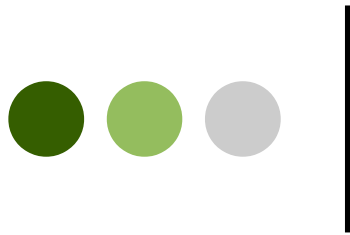
List View



Grid View



Tab Host



Propriétés de placement

- 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 | ...`



Propriétés de placement

- Poids

- Taux d'espace libre affectés à chaque widgets
- `android:layout_weight = ?` (0 par défaut)

- Espacement (intra)

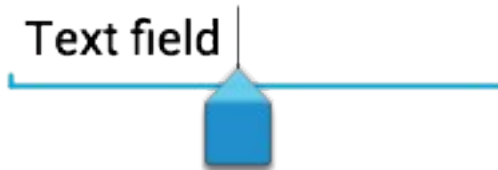
- Espacement entre un contenu et les bords de sa vue
- `android:padding? = top | left | right | bottom`

- Espacement (inter)

- Espacement entre les vues
- `android:layout_margin? = ??px`



Quelques Widgets...



jay@gmail.com

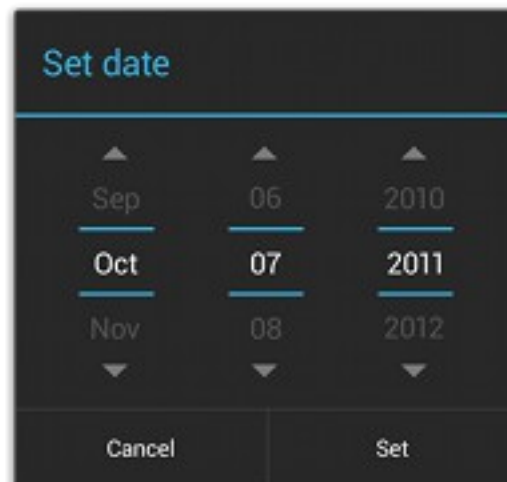
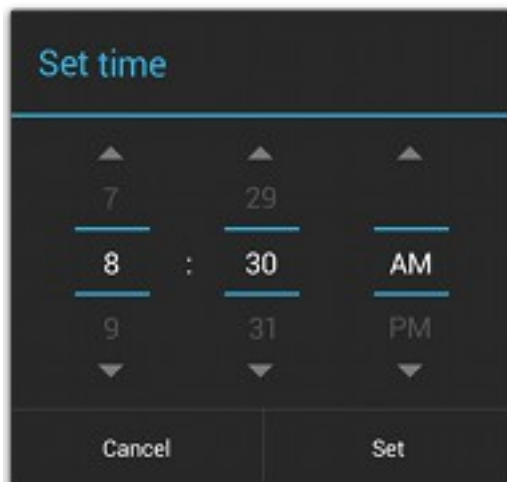
Home

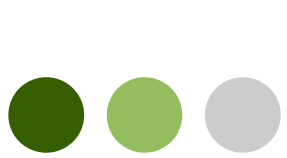
Home

Work

Other

Custom





Sauvegarder/restaurer l'état des vues

- 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 recréation 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 unique objet (le chanceux !) survive à un changement de configuration
 - onRetainNonConfigurationInstance() et getLastNonConfigurationInstance()



Fragments

- Nouveauté depuis Android 3.0
 - Introduite pour gérer les écrans plus larges (ex: tablettes)
- Principe de base
 - Fragmentation de l'espace d'affichage en différentes zones, chargeables indépendamment
- Un fragment est une classe qui étend
 - `android.app.Fragment`
- Les fragments sont ensuite attachés/détachés à une activité hôte
 - N'héritent pas du contexte. Le récupère de leur activité hôte.



- Possède son propre cycle de vie (callbacks)
- Mais synchronisé avec le cycle de l'activité hôte
- Possède un BackStack interne (gérée par l'hôte)

- Gérer la mise en page

- Utiliser le FragmentManager
- Ajouter, supprimer et remplacer des fragments dynamiquement



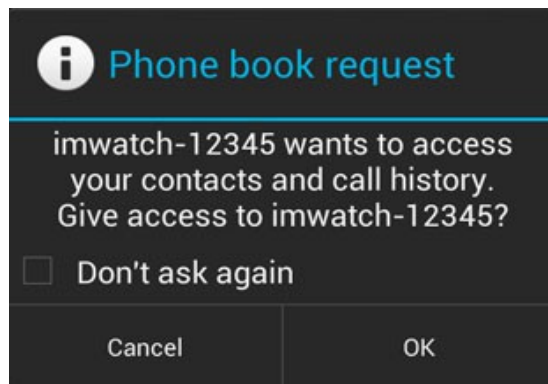
Activités dédiées

- Afin de simplifier les choses, il existe des classes prédéfinies à étendre directement
 - ListActivity : si votre activité présente une liste d'items
 - ActionBarActivity : si votre activité contient une barre d'actions
 - TabActivity : si votre activité présente des onglets
 - PreferenceActivity : si votre activité présente un panneau de préférences
 - FragmentActivity : si votre activité contient des fragments
 - MapActivity : si votre activité présente une carte google maps (introduite par l'API Google Maps)

Dialogues et Toasts

- Dialogues

- Confirmation

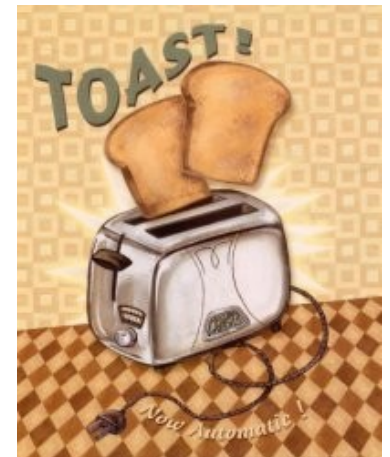
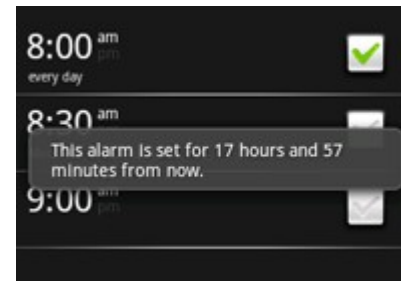


- Progression



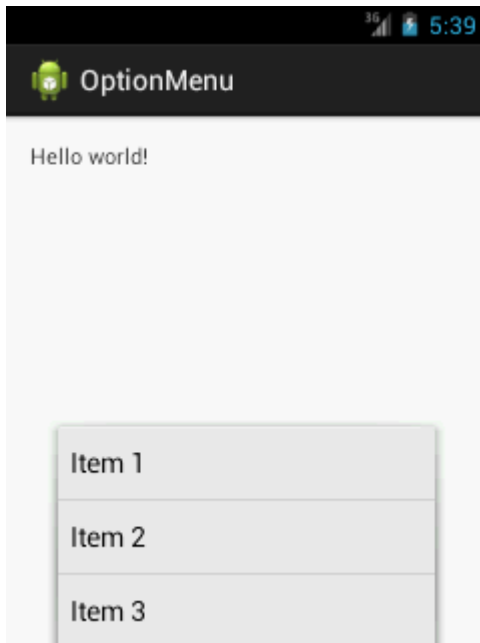
- Toasts

- Message compact et rapide à l'intention de l'utilisateur

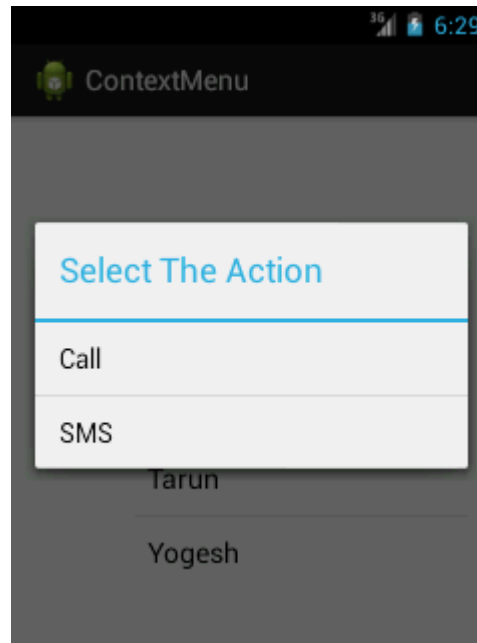




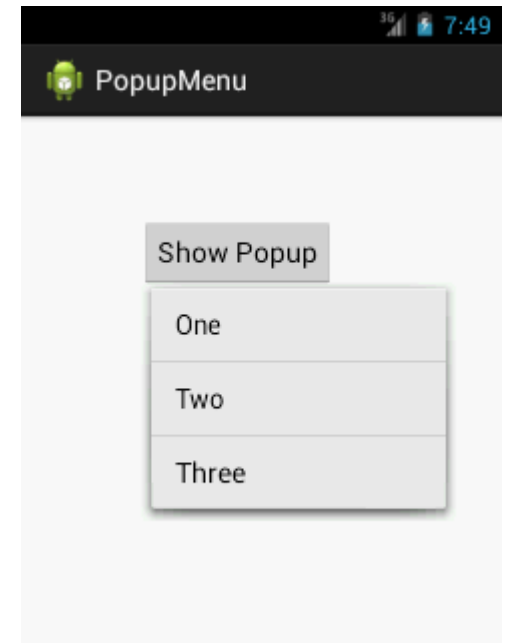
Menus



Option Menu
(activé par la touche
Menu du téléphone)



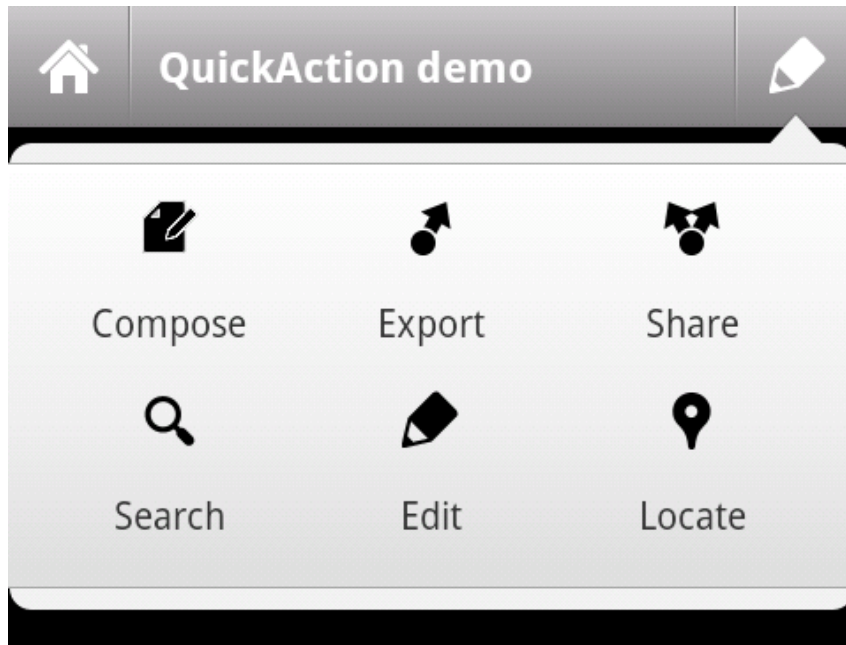
Context Menu
(activé par un click long
sur un élément)



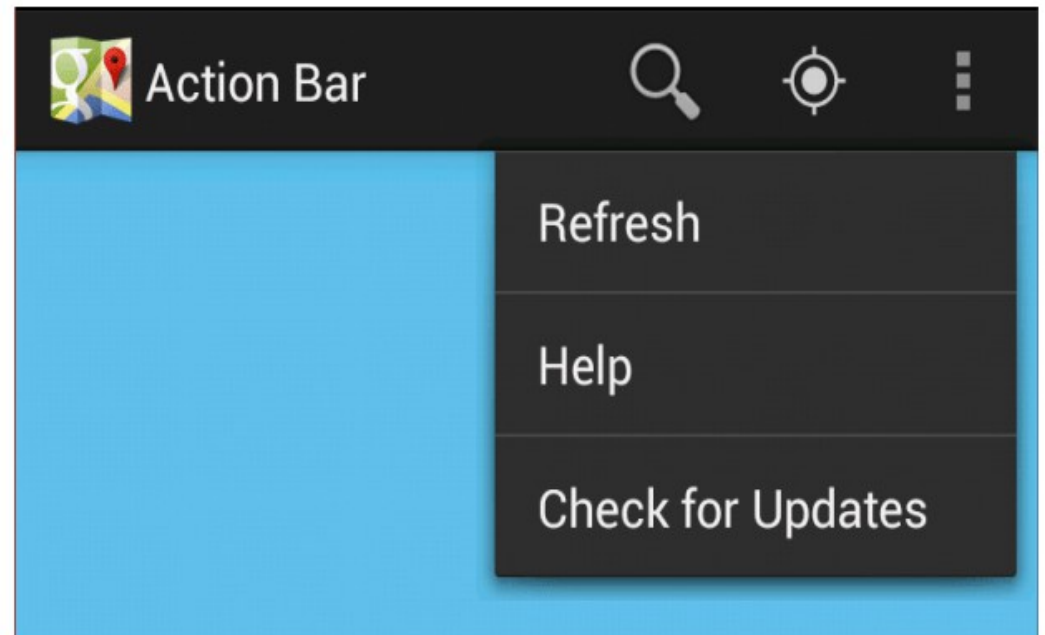
Popup Menu



Actions



Actions rapides



Barre d'action

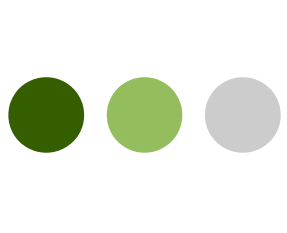


Instanciation des vues

- Il est nécessaire d'instancier les vues (i.e. obtenir des objets) pour pouvoir les manipuler
- Récupérer les vues 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
 - Désérialiser (*inflate*) un fichier XML décrivant un layout ou un menu, et greffer le sous-arbre ainsi obtenu
 - `View my_view = LayoutInflater.inflate(R.layout.main, null);`
 - `MenuInflater.inflate(R.menu.control, my_menu);`

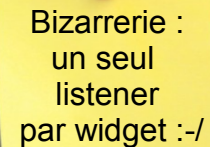


2nd partie



Gestion des évènements

- Principe des écouteurs de Java SE
 - 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)

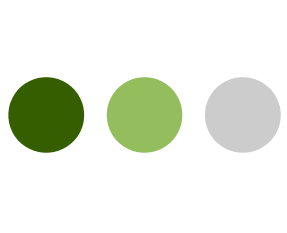


Bizarrerie :
un seul
listener
par widget :-/



Les adaptateurs

- Les adaptateurs sont des classes qui lient des données aux vues de l'UI
 - Les vues concernées étendent `android.widget.AdapterView`
- Classes d'adaptateurs
 - Héritent de `android.widget.BaseAdapter`
 - `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 (très vite incontournable)



Apparences des items

- L'apparence des items sont définies 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.univpau.bankster.R.layout.mon_bel_item`



MVP : code source

```
package fr.univpau.bankster;

public class CustomerList extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Objet M
        ArrayList<String> M = new ArrayList<String>();
        M.add("Dominique Strauss-Kahn");
        M.add("Liliane Bettencourt");
        //Objet V
        ListView V = (ListView) findViewById(R.id.listview1);
        //Objet P (connecté à M)
        ArrayAdapter<String> P = new ArrayAdapter<String>(this,
            fr.univpau.bankster.R.layout.mon_bel_item, M);

        V.setAdapter(P); //V connecté à P
    }
}
```



Application Multi-écrans

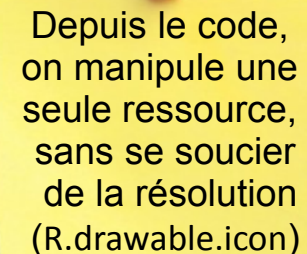
- Prévoir différentes variantes d'une même image

- /res/drawable-hdip/icon.png
- /res/drawable-mdip/icon.png
- /res/drawable-ldip/icon.png

	Low density (120), <i>ldpi</i>	Medium density (160), <i>mdpi</i>	High density (240), <i>hdpi</i>
Small screen	QVGA (240x320)		
Normal screen	WQVGA400 (240x400) WQVGA432 (240x432)	HVGA (320x480)	WVGA800 (480x800) WVGA854 (480x854)
Large screen		WVGA800* (480x800) WVGA854* (480x854)	

- Prévoir les deux orientations possibles (*portrait* ou *landscape*)

- /res/layout-port/main.xml
- /res/layout-land/main.xml



Depuis le code,
on manipule une
seule ressource,
sans se soucier
de la résolution
(R.drawable.icon)



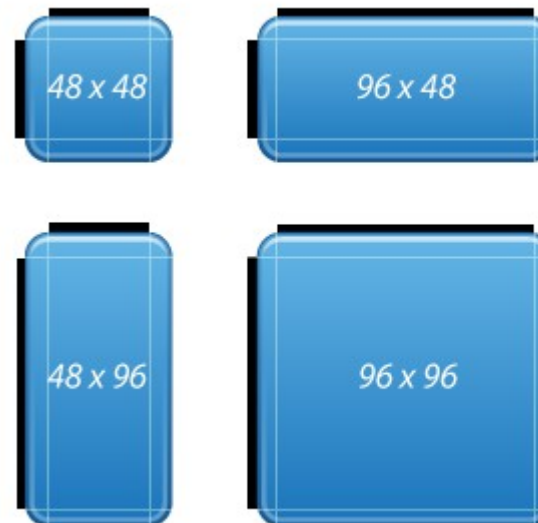
Images redimensionnables

- Utiliser les images 9 patches
 - 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

Scalable Area



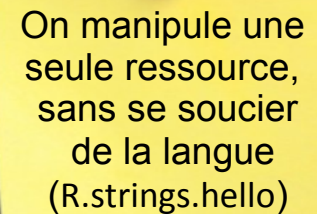
*48x48 button can be
scaled to any size larger
than 48x48*





Application Multi-langues

- Prévoir différentes variantes d'une même chaîne
 - /res/values-fr/strings.xml
 - /res/values-en/strings.xml
 - /res/values-it/strings.xml
- 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



On manipule une seule ressource, sans se soucier de la langue (R.strings.hello)

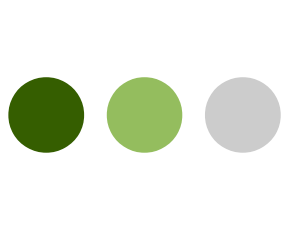


Persistence et threading



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");`



Gestion de fichiers plats

- Mécanisme de lecture/écriture de fichiers
 - Exploite l'espace de stockage interne ou externe
 - API habituelle java.IO
- Sauvegarde et chargement
 - Flux de sortie : `FileOutputStream fos = openFileOutput("CAC40.dat", Context.MODE_PRIVATE)`
 - Flux d'entrée : `FileInputStream fis = openFileInput("CAC40.dat")`
- Cas d'un fichier statique (lecture uniquement)
 - Déposez-le dans le répertoire `res/raw/` de votre projet
 - Accès avec `openRawResource(R.raw.cac40)`




Sérialisation d'objets

- Chaque classe implémente l'interface Serializable (+champs serialVersionUID)
 - Idem que pour java SE
 - Format de stockage binaire
- Basé sur la gestion de fichiers ci-avant
 - Sérialisation
 - `ObjectOutputStream oos = new ObjectOutputStream(fos);`
 - `oos.writeObject(myObject);`
 - Désérialisation
 - `ObjectInputStream ois = new ObjectInputStream(fis);`
 - `myObject = (myClass) ois.readObject();`

●●● | XML et JSON

- XML (API javax.xml.parsers)

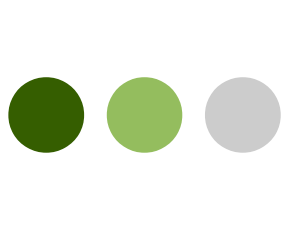
- Parsing de ressources au format XML
- Approche hiérarchique (DOM) : org.w3c.dom
 - Le parseur transforme le document en une arborescence d'objets, où chaque balise (tag) est un noeud
- Approche évènementielle (SAX) : org.xml.sax
 - Considère le document comme un flux de caractères, où chaque balise (tag) ouvrante et fermante reconnue par le parseur déclenche un traitement



Note
SAX plus
performant
que DOM !

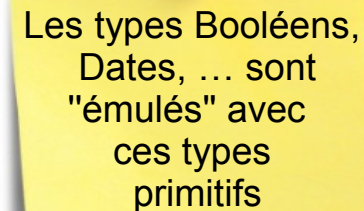
- JSON (API org.json)

- Parsing de ressources au format JSON
 - Le parseur permet de récupérer les objets JSON (paires clé/valeur) et les tableaux JSON décrits dans le document



Base de données embarquée

- Android embarque le SGBD-R *SQLite*
 - Léger et puissant
 - Typage dynamique des colonnes
 - Ne gère pas les contraintes d'intégrité référentielle
- Types de données
 - NONE, INTEGER, REAL, TEXT, BLOB
- Implémentation
 - Support du standard SQL-92
 - Mais manque RIGHT OUTER JOIN et FULL OUTER JOIN...
 - Support partiel des déclencheurs (triggers)



Les types Booléens,
Dates, ... sont
"émulés" avec
ces types
primitifs



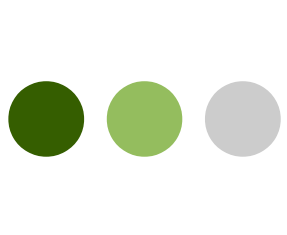
Bonnes pratiques

- Créer un helper

- Étendre `android.database.sqlite.SQLiteOpenHelper`
- Classe abstraite qui gère la création, l'ouverture et la montée de version d'une base de données
- `myHelper = new BanksterHelper(context, "bankster.db", null, 1)`

- L'instance de la BDD est ensuite obtenue à l'aide du helper, selon deux modes

- `SQLiteDatabase db;`
- `db = myHelper.getWritableDatabase() //lecture et écriture`
- `db = myHelper.getReadableDatabase() //lecture seule`



Interrogation de la base

- Approche par SQL brut

- La requête est fournie sous forme de chaîne de caractères du dialecte SQL
- `db.rawQuery("SELECT * FROM Customer WHERE id>? AND id<?", new String[]{"47645", "58421"})`

- Approche par composante

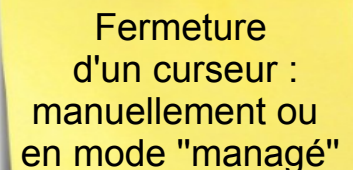
- Une requête est fournie via ses composantes relationnelles (projection, sélection, groupement, tri...)
- `db.query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)`



Résultats

- L'ensemble des tuples (ou n-uplets) retournés par une requête est un curseur
 - `Cursor res = db.query(...)`
- On peut ensuite parcourir cet ensemble
 - Même principe que les itérateurs de collection Java
 - Utiliser les méthodes de la forme `get<Type>(int columnIndex)` pour récupérer la valeur d'un champ

```
//vérifie qu'il y a au moins un tuple
if (res.moveToFirst()) {
    //itère sur chaque tuple
    do {
        String customerName = res.getString(3); //3e champ
    } while (res.moveToNext())
}
```



Fermeture
d'un curseur :
manuellement ou
en mode "managé"



Ajout, suppression, m-a-j

- Représentation d'un tuple

- Un n-uplet est une instance de `android.content.ContentValues`
- n paires nom-de-champ/valeur-de-champ

- Ajout de tuples

- `db.insert(String table, String nullColumnHack, ContentValues values)`

- Supression de tuples

- `db.delete(String table, String whereClause, String[] whereArgs)`

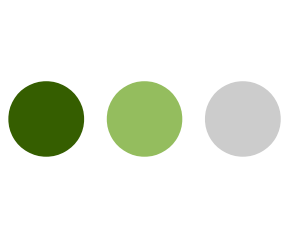
- Mise à jour de tuples

- `db.update(String table, ContentValues values, String whereClause, String[] whereArgs)`



Threading : éviter les ANR

- Gérer les traitements qui ralentissent l'UI et donc qui dégradent l'expérience utilisateur
 - Éviter une "*Application Not Responding*" (ANR)
 - Fermeture forcée de l'application au bout de n secondes
- Deux véritables cas d'écoles :
 - 1) Communications réseaux
 - Une `NetworkOnMainThreadException` est levée depuis Android 3.x
 - 2) Manipulations SQLite
 - Voir du côté de `android.app.LoaderManager`
- Et pensez à faire patienter votre utilisateur
 - Barre de progression, SplashScreen au démarrage...



Thread principal : **UIThread**

- Tous les composants d'une application démarrent dans le thread principal **UIThread**
 - Gère l'affichage graphique et les interactions utilisateur
 - Vos traitements "consommateurs" et lents bloqueront tous les autres composants (dont affichage + interactions) :-(
- Nécessité de déplacer ces traitements en tâches de fond (i.e. processus légers)
 - A l'aide de tâches asynchrones
 - A l'aide de vos propres Threads enfants
- Puis les synchroniser avec l'interface graphique
 - Car le UIThread est le seul habilité à modifier les vues !

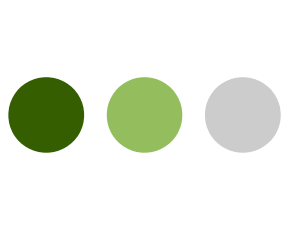


Exploiter les dispositifs matériel



Les capteurs

- Un périphérique Android peut posséder aucun ou plusieurs capteurs (*sensors* en anglais)
 - Cinémomètre (ou accéléromètre)
 - Gyroscope (ou boussole)
 - Luminomètre
 - Magnétomètre
 - ...
- Constantes supportées par la classe `android.hardware.Sensor`
 - `TYPE_AMBIENT_TEMPERATURE`, `TYPE_GRAVITY`,
`TYPE_GYROSCOPE`, `TYPE_LIGHT`...



Principes des capteurs

- **Système d'abonnement à un capteur**
 - Votre programme est à l'écoute des événements qui surviennent au niveau d'un capteur
 - Le `SensorManager` (`android.hardware.SensorManager`) permet de gérer facilement les abonnements en cours
 - Méthodes `registerListener()` et `unregisterListener()`
- **Surtout, bien penser à se désabonner**
 - Car les données continuent d'être acquises (même si elles ne sont pas traitées) et cela consomme de l'énergie !
 - Se gère au niveau du cycle de vie des composants concernés
 - Typiquement, dans les méthodes `onPause()` et `onDestroy()`



Capteurs : code source

```
package fr.univpau.bankster;

import android.app.Service;
import android.hardware.*;

public class FallingDown extends Service {

    @Override
    public void onCreate() {
        // Mise en place de l'écoute de l'accéléromètre
        sm = (SensorManager) getSystemService(SENSOR_SERVICE);
        a = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sm.registerListener(new SensorEventListener() {...}, a);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        sm.unregisterListener(...);
    }
}
```



Acquisition des données

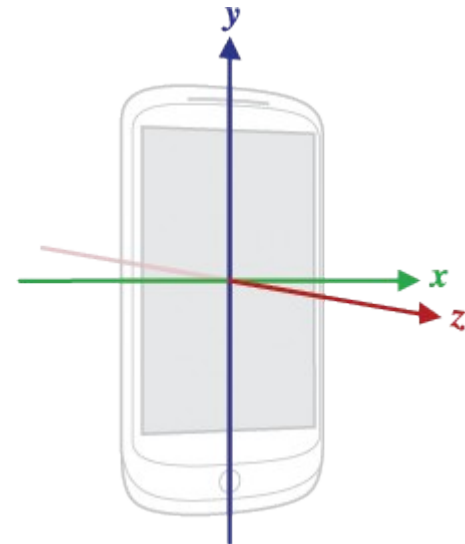
- Choisir son taux d'acquisition (toutes les n microsecondes)
 - Exemple de constantes de la classe Sensor :
SENSOR_DELAY_NORMAL, SENSOR_DELAY_FASTEST...
- Interface android.hardware.SensorEventListener commune à tous les capteurs
 - onAccuracyChanged() : la précision a changée (+/-)
 - onSensorChanged() : une nouvelle valeur brute est disponible
- Chaque évènement est un objet instance de la classe android.hardware.SensorEvent



Evènement de capteur

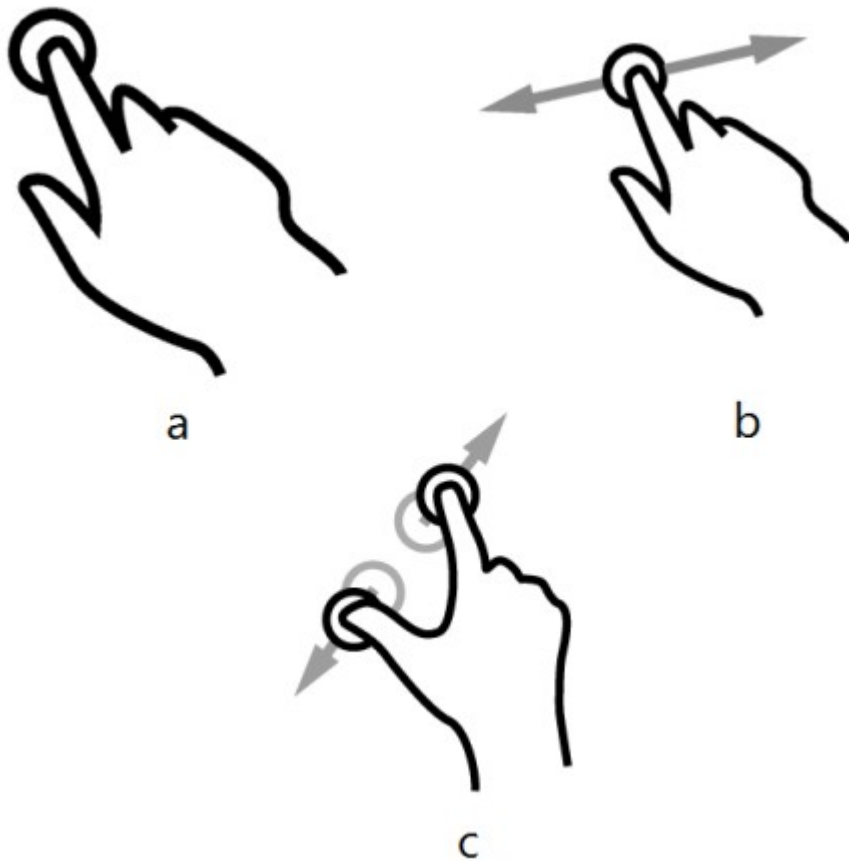
- L'évènement est modélisé par :
 - la précision de l'acquisition
 - le capteur qui l'a généré
 - son horodatage
 - les valeurs brutes
 - Vecteur de flottants dont la taille et l'interprétation dépendent du capteur ►

- Une seule valeur
 - Luminomètre, pression, proximité...
- Trois valeurs (X-Y-Z)
 - Accéléromètre, gravité, gyroscope...





Multi-touch



- Supporte plusieurs pointeurs simultanés
 - Doigt, stylet...
 - Limite théorique : 256
 - `event.getPointerCount()`
 - À chaque pointeur actif est attribué un ID
- Trois gestes de base
 - a) Tap
 - b) Drag
 - c) Pinch-Zoom



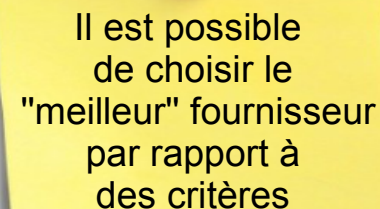
Évènements gestuels

- Une même instance de `android.view.MotionEvent` peut ainsi modéliser de multiples évènements
 - Autant qu'il y a de pointeurs actifs à la surface
 - Technique du masque binaire pour extraire les informations
 - `event.getActionMasked()`
- Différencie le premier pointeurs des suivants
 - Cycle de vie (ou d'actions) du premier pointeur
 - `ACTION_DOWN` → `(ACTION_MOVE)*` → `ACTION_UP`
 - Cycle de vie (ou d'actions) des suivants
 - `ACTION_POINTER_DOWN` → `(ACTION_MOVE)*` → `ACTION_POINTER_UP`
 - L'ID d'un pointeur reste inchangé lors de son cycle de vie

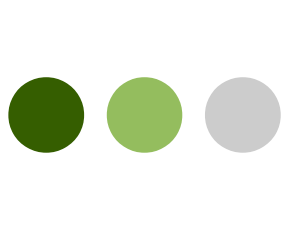


Géolocalisation

- Exploiter le gestionnaire de position
 - Instance de `android.location.LocationManager`
- Requière le choix d'un fournisseur de position
 - Instance de `android.location.LocationProvider`
 - Exemple de constantes de la classe `LocationManager` :
`LocationManager.GPS_PROVIDER`,
`LocationManager.NETWORK_PROVIDER`, ...
 - Chaque fournisseur offre diverses caractéristiques
 - Consommation d'énergie
 - Précision
 - Capacité à déterminer l'altitude
 - ...



Il est possible
de choisir le
"meilleur" fournisseur
par rapport à
des critères



Évènements de position

- Interface `android.location.LocationListener` pour écouter les changements de position
 - `onLocationChanged()` : une nouvelle position est disponible
 - `onProviderDisabled()/Enabled()` : fournisseur désactivé/activé
 - `onStatusChanged()` : le statut du fournisseur a changé
 - `OUT_OF_SERVICE`, `TEMPORARILY_UNAVAILABLE`, `AVAILABLE`
- Chaque position est un objet instance de la classe `android.location.Location`
 - Modélisé par : latitude, longitude, cap, altitude, vitesse, et horodatage
 - Parfois, détails additionnels dans les extras (*Bundle*)



Géolocalisation : code source

```
package fr.univpau.bankster;

import android.app.Activity;
import android.location.*;

public class FindNearestAgency extends Activity {

    @Override
    public void onCreate() {
        // Choix du fournisseur de position GPS
        lm = (LocationManager) getSystemService(LOCATION_SERVICE);
        String gps_prov = LocationManager.GPS_PROVIDER;
        // Mise en place de l'écoute des changements de position
        loc_list = new LocationListener() {...}
        // Laps (5sec) et distance (6m) minimums entre 2 updates
        lm.requestLocationUpdates(gps_prov, 5000, 6, list);
    }

    @Override
    public void onDestroy() {
        lm.removeUpdates(loc_list); // Ne pas oublier !
    }

}
```




Cartographie

- La géolocalisation est un tremplin naturel vers la cartographie
 - Service d'images tuilées pour permettre une visualisation fluide et performante
 - Géocodage (avant/inverse) : coordonnées ↔ adresses
 - + services ad-hoc : itinéraires, cadastre, lieux d'intérêts, trafic, street view, ...
- Choisir une tierce API de cartographie
 - Fournie par des entreprises qui se sont spécialisées dans la cartographie
 - Google, Mappy, IGN (Territoire Français), Nokia Here Maps...
 - L'API Google Maps est logiquement favorisée sous Android

●●● | Google Maps – Éléments clés

● MapView

- Instance de `com.google.android.maps.MapView`
- Vue composite (*ViewGroup*) destinée à afficher une carte



● MapController

- Instance de `com.google.android.maps.MapController`
- Utilisé pour contrôler la carte, vous permettant de centrer et de régler le niveau de zoom...

● Overlay

- Instance de `com.google.android.maps.Overlay`
- Permet d'utiliser un canvas pour dessiner autant de couches que nécessaires, affichées au dessus de la carte



Google Maps - Coordonnées

- Chaque coordonnée est un objet instance de la classe `com.google.android.maps.GeoPoint`
 - Moins riche qu'une instance de `android.location.Location`
 - Latitude et Longitude uniquement
 - Exprimées en microdegrés et non plus en degrés (donc $\times 10^6$)
- Projection des coordonnées
 - Traduire des coordonnées géographique (latitude, longitude) en coordonnées écran (x,y)
 - `com.google.android.maps.GeoPoint` \leftrightarrow `android.graphics.Point`
 - Interface `com.google.android.maps.Projection`
 - Méthodes `toPixels()` et `fromPixels()`



Bluetooth : vue d'ensemble





Divers



Services système

- Il est fréquent de récupérer des Managers à partir de services systèmes préconfigurés
 - Appel de la méthode `getSystemService(Context.XXXXX)`
 - Retour : `LocationManager`, `LayoutInflater`, `WifiManager`...
- Exemples de services (constantes)
 - `Context.LOCATION_SERVICE`
 - `Context.LAYOUT_INFLATER_SERVICE`
 - `Context.STORAGE_SERVICE`
 - `Context.TELEPHONY_SERVICE`
 - `Context.WIFI_SERVICE`



Alarmes et Timers

- Les alarmes sont un moyen de déclencher des intents (et donc des composants)
 - à des heures déterminées
 - à des intervalles déterminés
- Prise en charge par le gestionnaire d'alarmes
 - `AlarmManager am = (AlarmManager) getSystemService(Context.ALARM_SERVICE)`
 - `am.set(TYPE, TIME, pendingIntent)`
- Les timers eux, gèrent l'exécution de tâches
 - `new Timer().schedule(new TimerTask() {...}, new Date());`



Linkify, TextWatcher

- Transformer des textes (TextView) en liens cliquables (i.e. générateurs d'intents)
 - Classe utilitaire `android.text.util.Linkify`
 - Reconnaissance de motifs : n° tel, email, URL, map...
- Contrôler la saisie d'un texte (EditText)
 - Classe utilitaire `android.text.TextWatcher`
 - Gère des événements : avant, après, pendant la saisie



Application : variable globale

- Comment partager des données à travers tous les composants d'une application ?
 - Créer une sous classe de `android.app.Application`
 - Gère des événements propres à la vie de l'application
 - `onCreate()`, `onTerminate()`, `onLowMemory()`, `onConfigurationChanged()`
 - Se déclare au niveau du nœud `<application>` du manifeste
- Principe du singleton
 - Une seule instance de la classe pour toute l'application
 - Les variables d'instances sont les données à partager



Mapping objet-relationnel

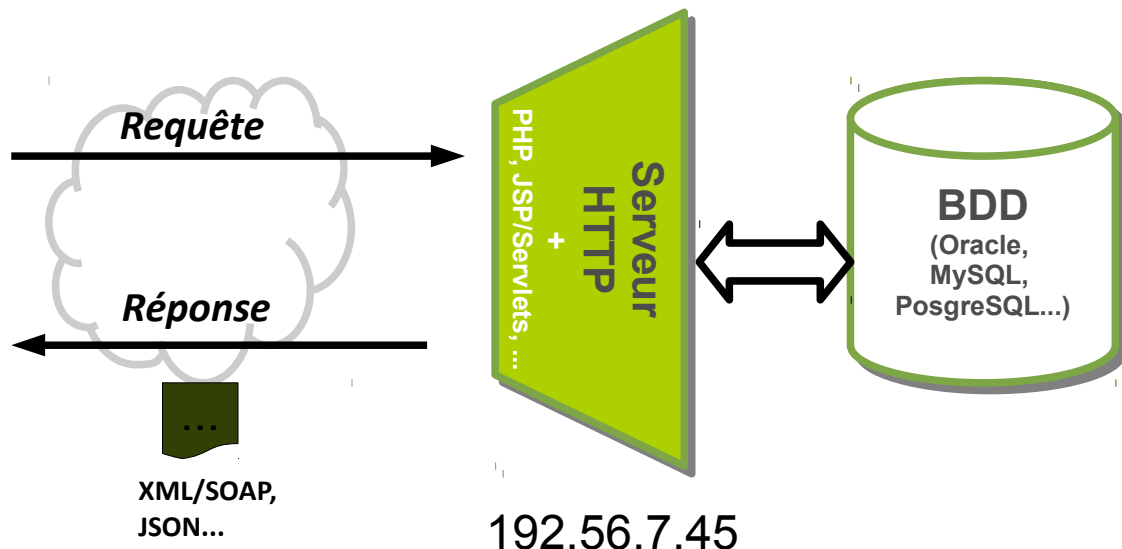
- Le vas-et-viens entre la BDD embarquée et les objets métiers est fastidieuse
 - Appliquer le fameux pattern DAO
- Il existe toutefois des solutions ORM adaptées à Android (car légères)
 - ORMLite
 - DB4O
 - Andorm
 - ActiveAndroid
 - Orman
 - NeoDatis
 - DataFramework



Base de données distante

- La connexion d'un périphérique Android à une BDD distante n'est pas prévue (ni souhaitable)
 - Pas de pont "JDBC" en quelque sorte
- Il est nécessaire de mettre en place une approche par "WebService"

Astuce :
masquez le tout
avec un
ContentProvider






Interactions Client/Serveur

- Socket

- Classes Socket, ServerSocket, ...



Pas de RMI
sous Android

- HTTP

- Classes HttpClient, HttpResponse, NameValuePair, ...
- Les trois usages CRUD du protocole HTTP
 - Function-oriented : GET <http://www.bankster.com/customerDelete?id=45642>
 - Service-oriented : GET <http://www.bankster.com/customer?cmd=del&id=45642>
 - REST : **DELETE** <http://www.bankster.com/customer/45642/>

- SOAP

- Classes SoapObjet, SoapSerializationEnvelope, ...

Abstraction





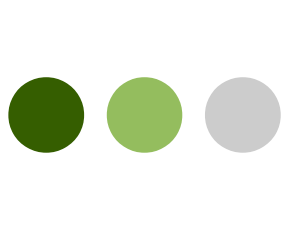
Mockups pour MobileApps



- Prototypage de l'UI de votre MobileApp
 - Dessiner les écrans avant de démarrer le développement
- Outils de mockups
 - Balsamiq (web demo)
 - MobiOne Studio (free)
 - OmniGraffle (Mac, \$99)
 - <http://yeblon.com/androidmockup/>
 - <http://mokk.me>
 - ...



Tests et déploiement



Test de la couche métier

- Tests unitaires de trois types de composants applicatifs Android
 - Activités, Services, et Providers
- Extension JUnit pour Android
 - `android.test.AndroidTestCase` étend `junit.framework.TestCase`
 - `junit.framework.Assert`
 - `android.test.InstrumentationTestRunner`
- Mise en place d'un projet de test
 - Sous-répertoire `tests/` de votre projet android
 - Code source de vos tests + manifeste




Test de l'UI



- Le test du singe sur votre IHM
 - Idée : "si elle résiste au comportement anarchique d'un singe, elle résistera à l'utilisateur"
 - Génère des événements pseudoaléatoires, tels que des clics, des pressions de touches, des gestes, etc.
- Le programme s'exécute dans votre émulateur
 - Se lance en ligne de commande
 - `adb shell monkey [options] <event-count>`
 - Vous suivez les événements générés dans le log
 - `:Sending Pointer ACTION_DOWN x=437.0 y=183.0`
 - `:SendKey (ACTION_DOWN): 90 // KEYCODE_FORWARD=`
 - ...

● ● ● | Exporter et signer

- 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 débogage 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
 - Utilitaire jarsigner.exe du JDK
 - 3 Optimiser l'APK qui vient d'être signé
 - Utilitaire zipalign.exe du SDK Android



Procédure
entièrement
automatisée
avec ADT

● ● ● | Distribution des MobileApps

● 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.bankster.org/bankster.apk>
- Type MIME pour le téléchargement : `application/vnd.android.package-archive`

○ Choisir un magasin alternatif

- App Brain, Aptoid (anciennement Bazaar), AndroLib, SlideMe, GetJar, Opera Mobile Store, Amazon App Store, Mob One, F-Droid...



Add-ons Google



Analyse d'audience

- Google Analytics SDK pour Android
 - Nombre d'utilisateurs actifs de votre application
 - Leur localisation à travers le monde
 - Impact de votre campagne de pub (réalisation d'objectifs)
 - Et plein d'autres métriques...
- Principes de fonctionnement
 - Affection d'un *UA number* de la forme UA-xxxxx-yy
 - Insertion d'instructions de tracking (événements, affichages...) dans le code
 - Tableau de bord disponible sur www.google.com/analytics

●●● | Google Cloud Messaging

● Technologie « Push » de Google

- Le serveur prend contact avec le client (i.e. le périphérique Android)
- Solution gratuite et sans quota, mais la taille du message délivré n'exède pas 4000 octets
 - Suffit à certaines application (ex: messagerie instantanée), pas à d'autres
 - Sert plutôt à indiquer au client que des données fraîches sont à récupérer

● Architecture

- Codez votre partie cliente
- Codez votre partie serveur
- Google fait le pont entre les deux parties

