
CONCEPTION SUR OS NOMADES

INTRODUCTION A ANDROID

(ECOSYSTEME / ARCHITECTURE / PACKAGES)

Plan du cours

- Présentation d'Android et de son écosystème
- Architecture d' l'OS Android
- Le cycle de vie d'une application
- Les principales classes du SDK

- Configurer un environnement de développement
- Un premier projet Android

Un ensemble de composants logiciels

- Un système d'exploitation
- Un SDK et des API pour développer sur le système
- Des applications embarquées
- Un marché accessible via le store installé sur les devices

Les principales caractéristiques

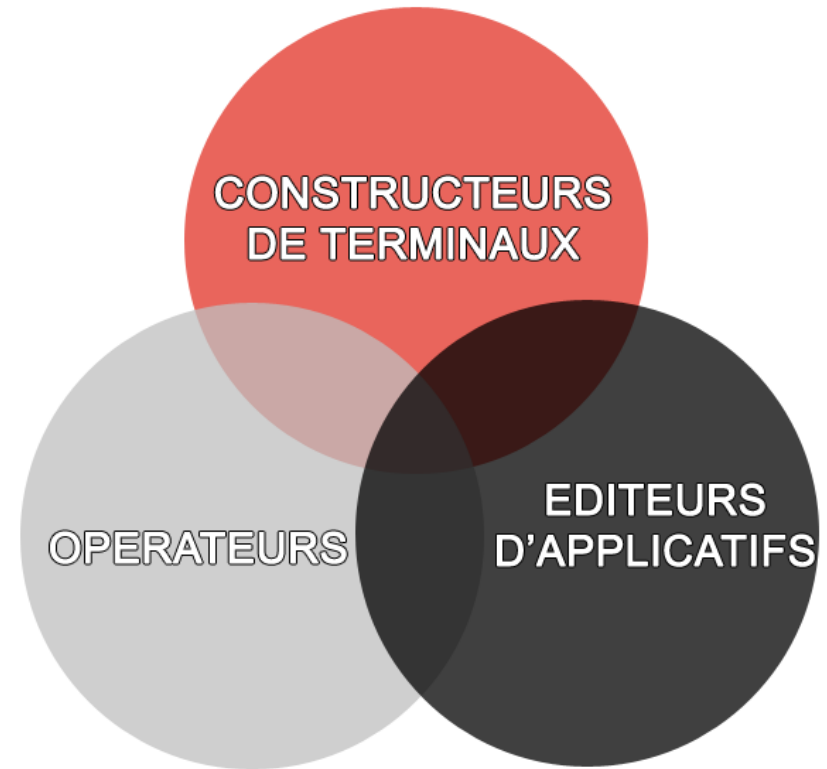
- Android est un système d'exploitation **open source** pour smartphones, PDA et terminaux mobiles conçu par Android, une startup rachetée par Google en 2005.
- Le système est également présent dans des téléviseurs, voitures, montres...
- Android est un système d'exploitation avec un noyau **Linux**.
- Il contient toutes les briques logicielles nécessaires à un constructeur ou un opérateur pour créer un téléphone mobile.
- Android a été conçu pour intégrer des applications existantes de Google : Gmail, Google Maps, Google +, YouTube, Play, ...
- Système **multi-process**: exécution de plusieurs applications en parallèle.
- Possibilité de définir des « **background services** »
- Android est positionné sur toute la couche logicielle (OS -> applications)

Les plus d'Android

- Java (communauté, expérience, portabilité, ...)
- Exécution d'applications en parallèle
- Utilisation du XML pour décrire les interfaces (un peu sur le même modèle que FLEX (ActionScript/MXML))
- Open source
- Modèle économique
- Widgets (comme sur Windows Mobile)
- Centralisation des notifications (sms, appel et toutes autres applications)
- Boussole électronique (accéléromètre + GPS) -> vue des rues
- Reconnaissance et synthèse vocale

Le contexte d'Android : les acteurs du marché mobile

- Les opérateurs veulent verrouiller leur position, protéger leur réseau en maîtrisant le trafic et valoriser leur base de clients (voix + data + contenu).
- Les constructeurs veulent se différencier par les fonctionnalités, l'aspect, la fiabilité et le prix. Ils souhaitent également valoriser leur base par le contenu.
- Les éditeurs veulent un accès complet au hardware, une faible fragmentation et peu de prélèvement pour pouvoir proposer des applications de pointes à des prix attractifs.



La stratégie de Google



- Google a une vue simple et claire du marché, comme sur internet, l'accès au média permettra de démocratiser l'usage.
- La différence entre les 2 médias, Internet s'est construit sur la gratuité, le mobile lui a grandit au côté de l'opérateur et du paiement.
- Le système Android, gratuit et Open source se positionne sur le marché de masse (Iphone = marché haut de gamme) et s'inscrit dans un modèle gagnant pour l'opérateur :
 - pas de coût de licence
 - redistribution sur Android Store
 - système ouvert de distribution d'application

Les 2/3 du marché des smartphones en France

ANDROID:

- 2/3 de parts de marché
- 17,5 millions de devices en France

iOS: 23% de parts de marché

Windows: 8% de parts de marché

France	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	63.3	65.1	1.8
BlackBerry	4.0	1.1	-2.9
iOS	21.2	23.4	2.2
Windows	7.2	8.3	1.1
Other	4.3	2.0	-2.3

EU5	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	69.2	70.7	1.5
BlackBerry	2.6	1.1	-1.6
iOS	19.1	19.2	0.1
Windows	6.5	8.1	1.6
Other	2.6	0.9	-1.7

Smartphone OS Sales Share (%)

Germany	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	73.6	77.0	3.4
BlackBerry	0.5	0.5	0.0
iOS	16.9	15.3	-1.6
Windows	6.1	6.6	0.5
Other	2.9	0.6	-2.3
GB	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	58.4	56.2	-2.2
BlackBerry	5.1	2.3	-2.8
iOS	28.7	32.1	3.4
Windows	7.0	9.1	2.1
Other	0.9	0.2	-0.7
France	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	63.3	65.1	1.8
BlackBerry	4.0	1.1	-2.9
iOS	21.2	23.4	2.2
Windows	7.2	8.3	1.1
Other	4.3	2.0	-2.3
Italy	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	62.5	70.7	8.2
BlackBerry	2.5	1.2	-1.3
iOS	19.9	12.9	-7.0
Windows	10.9	13.9	3.0
Other	4.2	1.3	-2.9
Spain	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	93.7	88.6	-5.1
BlackBerry	0.2	0.0	-0.2
iOS	3.1	7.6	4.5
Windows	1.3	3.0	1.7
Other	1.8	0.8	-1.0
USA	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	49.3	57.6	8.3
BlackBerry	0.9	0.7	-0.2
iOS	43.7	35.9	-7.8
Windows	5.6	5.3	-0.3
Other	0.5	0.4	-0.1
China	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	71.9	80.0	8.1
BlackBerry	0.3	0.1	-0.2
iOS	23.3	17.9	-5.4
Windows	1.9	1.0	-0.9
Other	2.6	1.0	-1.6
Australia	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	61.6	57.3	-4.3
BlackBerry	0.5	1.0	0.5
iOS	31.1	33.1	2.0
Windows	4.1	6.9	2.8
Other	2.7	1.7	-1.0
Japan	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	46.0	41.5	-4.5
BlackBerry	0.7	0.0	-0.7
iOS	49.0	57.6	8.6
Windows	0.3	0.9	0.6
Other	3.9	0.0	-3.9
EU5	3 m/e Mar 2013	3 m/e Mar 2014	% pt. Change
Android	69.2	70.7	1.5
BlackBerry	2.6	1.1	-1.6
iOS	19.1	19.2	0.1
Windows	6.5	8.1	1.6
Other	2.6	0.9	-1.7

System Architecture



Pour en savoir plus sur l'architecture d'Android:

<http://www.youtube.com/watch?v=QBGfUs9mQYY>

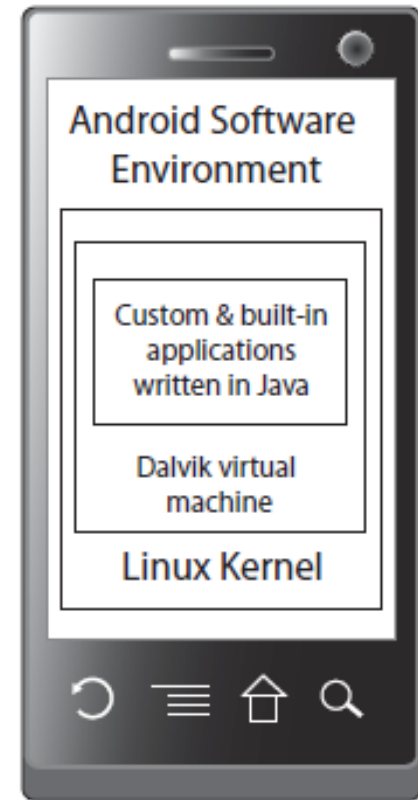
ARCHITECTURE

NOYAU LINUX

- Couche basse pour gérer les services du système
 - Gestion de la mémoire
 - Gestion des processus
 - Pilotes matériels
 - Gestion réseau
 - Sécurité
- Première abstraction entre matériel et couche applicative

POURQUOI LINUX ?

- Linux est une plateforme éprouvée et stable
- La stabilité est primordiale sur les performances pour un device mobile
- Linux fournit une couche d'abstraction facilitant l'intégration de nouveaux composants.



ARCHITECTURE

LIBRAIRIES NATIVES

- **Librairie système C : libc**
 - Dérivée de BSD (UNIX - Berkeley Software Distribution) optimisée pour les systèmes embarqués
 - Rapide
 - Services android tels que logging et system properties
- **Stockage, Rendu graphique, multimédia**
 - SQLite : un SGBD simple et puissant accessible pour toutes les applications
 - API Média basée sur OpenCore de PacketVidéo (mpeg4, h.264, mp3, amr, jpg, png..)
 - SGL/OpenGL ES, librairies 2D et 3D
 - LibWebCore (moteur de navigation web utilisé)
- **Surface Manager**
 - Pour créer des interfaces visuelles

ARCHITECTURE

ANDROID RUNTIME

- **DALVIK Virtual Machine**
 - VM optimisée pour les systèmes embarqués
 - Exécute un fichier .dex généré par dx tool
 - Une instance par processus

APPLICATION FRAMEWORK

- **Architecture conçue pour faciliter la réutilisation**
- **Chaque application est basée sur des services et systèmes :**
 - Views (UI)
 - Content Providers: pour accéder aux données d'autres applications
 - Ressources Manager: accès aux ressources statiques (images, fichiers, ...)
 - Notification Manager: permet d'afficher des alertes dans la barre status
 - Activity Manager: gère le cycle de vie des applis et l'enchaînement des vues

Pour en savoir plus sur la VM Dalvik:

<http://www.youtube.com/watch?v=ptjedOZEXPM>

Native Development Kit

LE NDK PERMET DE DEVELOPPER EN C/C++ SUR ANDROID

Le NDK est une suite d'outils permettant de compiler puis d'utiliser du code natif en C/C++ dans une application android. Code natif qui sera ensuite appelé via JNI (Java Native Interface), qui fournit des structures en C/C++ compatibles avec les types en java.

AVANTAGES / INCONVENIENTS

- Avantages
 - Permet de réutiliser des bibliothèques existantes en C (portabilité iOS par exemple)
 - Permet des traitements performant avec une faible occupation mémoire (3D, calculs, ...)
- Inconvénients
 - On se coupe de certains services facilités pour le sdk (en particulier les API externes)
 - Complexification de l'architecture de l'application

POUR ALLER PLUS LOIN SUR LE NDK

- Documentation & Exemples:
 - <http://developer.android.com/tools/sdk/ndk/index.html>
 - <http://developer.android.com/tools/sdk/ndk/index.html#Samples>
- Vidéos
 - <http://www.youtube.com/watch?v=5yorhsSPFG4>
 - <http://www.youtube.com/watch?v=sMcF08p5Maw>
- Livres
 - Android NDK Beginner's Guide (Sylvain Ratabouil) chez PackT
 - Pro Android C++ with the NDK (Onur Cinar) chez Apress
 - Android Native Development Kit Cookbook (Feipeng Liu) chez PackT

LES PRINCIPAUX PACKAGES

J2SE	java.util.* java.io.* java.lang.* etc
UI	android.widget.* android.view.* android.graphics.*
Telephony	android.telephony.IPhone
SMS	android.telephony.gsm.SmsManager

Web	android.webkit.WebView
Camera	android.hardware.CameraDevice
Local database	android.database.*
Maps	com.google.android.maps.MapView
Location	android.location.LocationManager
Multimedia	android.media.MediaPlayer
HTTP	org.apache.http.client.*

Pour en savoir plus: <http://www.youtube.com/watch?v=MPukbH6D-lY>

Les classes clés du framework

ACTIVITY

VIEW

SERVICE

CONTENT PROVIDER

INTENT

LAYOUT

BROADCAST RECEIVER

Pour en savoir plus: <https://sites.google.com/site/io/inside-the-android-application-framework>

DES COMPOSANTS, DES AUTORISATIONS, UNE CONFIGURATION & DES RESSOURCES

- Les applications Android sont composées d'un ou plusieurs composants parmi:
 - **Activity** (composant graphique correspondant à un écran)
 - **Services** (composant sans UI tournant en tâche de fond)
 - **Content Providers** (permet à l'application de partager des données)
 - **Broadcast Receivers** (répond aux notifications ou changement d'états)
- Chaque composant a un rôle différent dans le comportement de l'application et peut être activé individuellement y compris par d'autres applications.
- Tous les composants de l'application sont déclarés dans le fichier **manifest.xml**. On y retrouve également toutes les autorisations que demande l'application et la configuration nécessaire.
- Une application est également constituée de ressources (images, textes, sons, fichiers xml, ...).

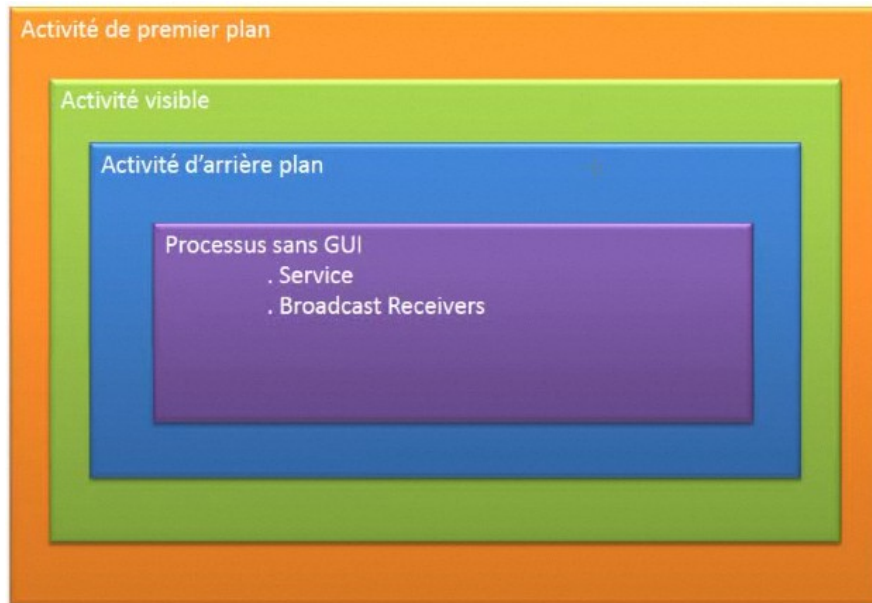
LA CLASSE ACTIVITY

- La classe *Activity* est l'unité d'exécution simple et visible.
- La classe *Activity* est en charge de la création d'un écran dans lequel vous pouvez placer votre UI avec *setContentView(int)*.
- Les *Activity* sont souvent présentées comme des écrans Fullscreen, elles peuvent aussi être utilisées comme des fenêtres flottantes (via *R.attr.windowIsFloating*) ou fixée à l'intérieur d'une autre *Activity* (utilisant *ActivityGroup*).
- 2 méthodes doivent impérativement être implémentées :
 - *onCreate(android.os.Bundle)*: appelé lors de la création de l'*activity*. Vous y appellerez *setContentView(int)* avec un *Layout* défini pour votre UI.
 - *onPause*: appelé lorsque l'utilisateur quitte votre *activity*. Tous les changements réalisés par l'utilisateur sont à sauvegarder (en règle générale par le *ContentProvider*).

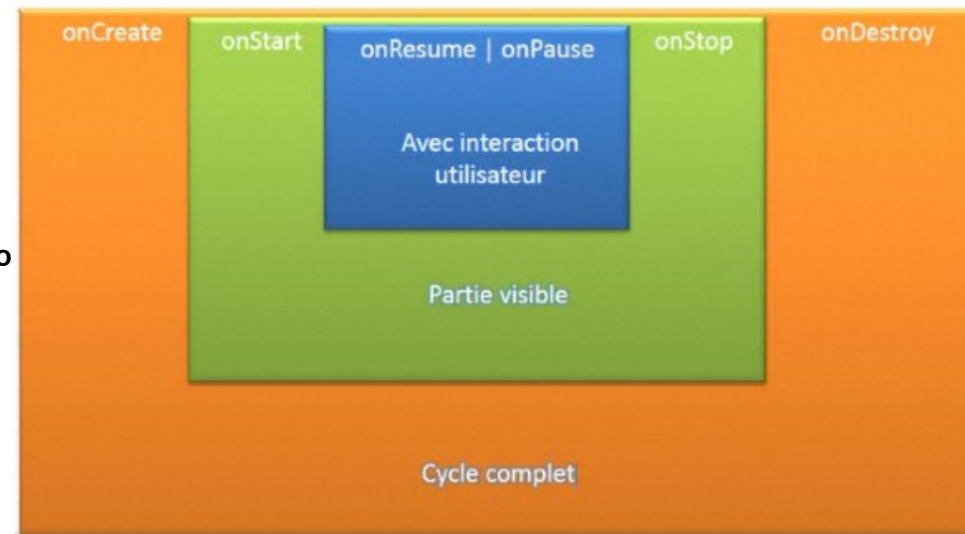
Doc: <http://developer.android.com/guide/components/activities.html>

VISIBILITE DES PROCESSUS & GESTION DE LA MEMOIRE

- Vos activités peuvent être interrompues/stimulées par l'extérieur.
- Il est important de gérer la mémoire et l'occupation CPU de son application en fonction de son état et de sa visibilité.



ou



LA CLASSE SERVICE

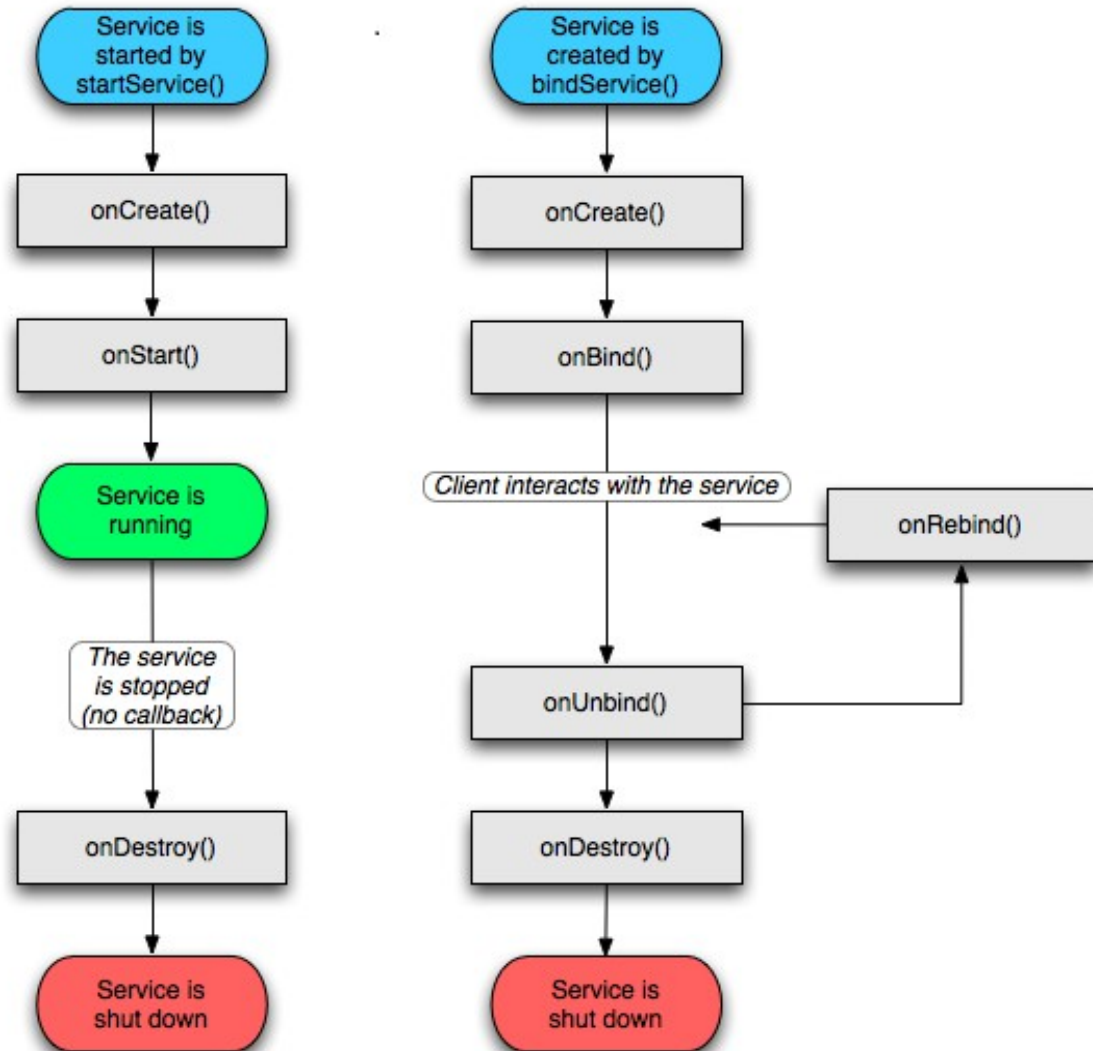
- La classe *Service* est un composant d'application qui tourne en tâche de fond sans interface utilisateur permettant une interaction durant une période indéfinie.
- Comme tout composant d'une application, chaque *Service* doit avoir une déclaration dans son package *AndroidManifest.xml*.
- Un *Service* peut être démarré par *Context.startService()* et *Context.bindService()* et arrêté par *stopService()* et *unbindService()*.
- Attention, un *Service* comme toute autre application s'exécute dans le thread principal du process qui l'héberge. Cela signifie que si votre service utilise intensivement les ressources ou bien lance des commandes bloquantes (requête http), il est nécessaire de le threader.

LA CLASSE SERVICE

- Un *Service* n'est pas un process séparé. L'objet Service n'implique pas nécessairement qu'il tourne dans son propre process.
- Un *Service* n'est pas un *Thread*.
- Un *Service* fournit 2 fonctionnalités:
 - Communiquer une demande de traitement en tâche de fond au système.
Cela correspond à l'appel à *Context.startService()* qui demande au système de scheduler le *Service*.
 - Exposer des fonctionnalités de votre application aux autres applications.
Cela correspond à l'appel à *Context.bindService()* qui permet une connexion vers le *Service* afin d'interagir avec lui.

Les classes clés : *Service*

CYCLE DE VIE



LA CLASSE SERVICE : cycle de vie

- Un *Service* peut être démarré par le système via *Context.startService()* ou bien par un appel à *onCreate()*.
- Quelque soit le nombre d'appel à *StartService*, l'appel à *Context.stopService()* stoppera l'unique instance du service.
- Android conserve le service tant qu'il n'est pas stoppé par le client. Toutefois si le manque de mémoire se fait sentir, le service peut être détruit en fonction de sa priorité.
- Si le service exécute les méthodes *onCreate()*, *onStart()*, or *onDestroy()* alors il devient le process de premier plan afin d'être sûr de ne pas être tué durant l'exécution de ces méthodes.
- Tous les process visibles ont une priorité supérieure aux instances de classe dérivée de Service.

CODE SAMPLE

```
Package monpackage.p8.service;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

Public class MonService extends Service implements Runnable {

Private      int      compteur = 0;
Private      Thread    MonThread;

@Override
Public void onCreate() {
    super.onCreate();
    MonThread = new Thread(this);
    MonThread.start();
}

public void run() {
    while(true) {
        try {
            Log.i("MonService", "CPT: "+ compteur++);
            Thread.sleep(60000); //Service sleeping
        } catch (Exception e) {
            Log.e("MonService1", e.getMessage());
        }
    }
}

@Override
public IBinder onBind(Intent intent) {
    return null;
}

}
```

```
Package monpackage.p8.service;

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

Public class MonActivityService extends Activity {

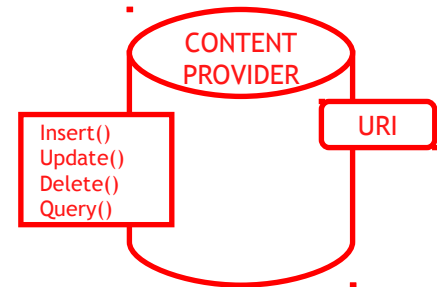
@Override
Public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // appeler le service
    Intent MonIntent = new Intent( this, MonService.class );
    startService(MonIntent);
}

}
```


LA CLASSE CONTENT PROVIDER

- La classe *ContentProvider* permet de gérer l'accès à des données depuis d'autres applications. C'est une manière de publier des données et de définir des règles de modifications de ces données depuis l'extérieur.
 - Android propose des *ContentProvider* par défaut sur les fonctions de base d'un téléphone (audio, vidéo, images, contacts...).
 - Un *ContentProvider* se compose :
 - D'une Uri (Uniform Resource Identifier)
 - De méthodes (Insert, Update, Delete, Query).
- 

The diagram illustrates the structure of a ContentProvider. It features a central oval labeled 'CONTENT PROVIDER'. To its left is a rectangular box containing the methods 'Insert()', 'Update()', 'Delete()', and 'Query()'. To its right is a rectangular box labeled 'URI'. A line connects the 'CONTENT PROVIDER' oval to the 'URI' box, and another line connects it to the methods box, indicating their relationship.
- Afin d'accéder aux données d'un *ContentProvider*, on utilise un *ContentResolver*.
 - Le *ContentResolver* communique avec la classe dérivée d'un *ContentProvider*. Il reçoit les demandes, effectue la requête demandée et renvoie les résultats.
 - Vous n'avez pas besoin de développer votre propre fournisseur si vous n'avez pas l'intention de partager vos données avec d'autres applications.

UNIFORM RESSOURCE IDENTIFIER

content://com.application.parishuit.democontentprovider/etudiant/123456

A B C D

- A: content est le préfixe standard servant à indiquer que les données sont contrôlées par un *ContentProvider*.
- B: Nom du package (com.application.parishuit) et de la classe (democontentprovider) contrôlant cette URI. Identifie le *ContentProvider* de cette URI.
- C: Permet au *ContentProvider* de savoir quelle donnée est requêtée. Ce segment est optionnel. Un *ContentProvider* peut exposer plusieurs données.
- D: Identifiant de la donnée souhaitée. Ce segment est optionnel.

CRÉER UN CONTENTPROVIDER

- Stocker les données avec SQLite.
- Étendre la classe *ContentProvider* pour proposer votre propre *ContentProvider*.
- Surcharger les 6 méthodes suivantes :
 - *query()* : Cette méthode retourne un objet Cursor sur lequel vous pouvez itérer pour récupérer les données.
 - *insert()* : Cette méthode est utilisé pour ajouter des données.
 - *update()* : Cette méthode est utilisé pour mettre à jour une données existantes.
 - *delete()* : Cette méthode permet de supprimer une donnée.
 - *getType()* : Retourne le type MIME des données contenues.
 - *onCreate()* : Appelé afin d'initialiser le Content Provider
- Ne pas oublier de déclarer votre *ContentProvider* dans *AndroidManifest.xml*.

LA CLASSE BROADCASTRECEIVER

- La classe *BroadcastReceiver* permet à une application de recevoir et répondre à un événement global (*Intent*), comme la sonnerie du téléphone ou un sms.
- Une application annonce qu'elle est à l'écoute de messages en déclarant des *IntentFilters* dans le fichier *AndroidManifest.xml* auquel elle associe son *BroadcastReceiver*.

```
<receiver
    class="com.parishuit.broadcastreceiver.smsreceiver"
    android:name="com.tuto.android.SMSReceiver">
    <intent-filter android:priority="50">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

- Le *BroadcastReceiver* n'a pas besoin d'être lancé afin d'être déclenché. Lorsque l'événement se produit, l'application est lancée automatiquement dès la notification de l'événement par Android.

LA CLASSE BROADCASTRECEIVER

- Comme les *Service*, les *BroadcastReceiver* ne possède pas d'interface graphique.
- L'interface *BroadcastReceiver* ne possède qu'une seule méthode `onReceive()` que votre classe devra implémenter.
- Attention: un *BroadcastReceiver* est détruit après l'appel à son callback *`onReceive(Context, Intent)`*, donc vous ne pouvez y déclencher des actions asynchrones.
- Si le *BroadcastReceiver* réalise un traitement long, il est recommandé de lancer le traitement au sein d'un *Service*.
- Android envoie l'*Intent* à tous les *BroadcastReceiver* abonnés par ordre de priorité (priorité définie dans le fichier *`AndroidManifest.xml`*).
- Si un *BroadcastReceiver* souhaite interrompre la réception du Broadcast à ceux de moindre priorité, il utilise la méthode *`abortBroadcast()`*.

TYPE DE BROADCAST TRANSMIS AU BROADCASTRECEIVER

Il existe 2 types de Broadcast qui peuvent être reçus par un *BroadcastReceiver*:

- Les Broadcast classiques envoyés par *sendBroadcast* qui sont complètement asynchrones. Tous les *BroadcastReceiver* sont exécutés dans un ordre indéfini et en même temps.
- Les Broadcast ordonnés envoyés par *sendOrderedBroadcast* sont émis aux *BroadcastReceiver* séquentiellement. Chaque *BroadcastReceiver* peut propager un résultat au *BroadcastReceiver* suivant ou bien arrêter la séquence et être le dernier *BroadcastReceiver* à traiter le message.

DES COMPOSANTS, DES AUTORISATIONS, UNE CONFIGURATION & DES RESSOURCES

Les applications Android sont composées:

- D'un ou plusieurs composants
 - *Activity*
(composant graphique correspondant à un écran)
 - *Services*
(composant sans UI tournant en tâche de fond)
 - *Content Providers*
(permet à l'application de partager des données)
 - *Broadcast Receivers*
(répond aux notifications ou changement d'états)
- De ressources (images, textes, sons, fichiers xml, ...).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.storybird.bubblebuster"
    android:versionCode="4"
    android:versionName="1.3"
    android:screenOrientation="portrait" >
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.READ_CONTACTS"></uses-permission>
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.SEND_SMS" />

    <application android:icon="@drawable/icon" android:label="@string/app_name"
        android:launchMode="singleInstance" >
        <activity android:name=".BubbleBuster"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="com.google.ads.AdActivity" android:configChanges="keyboard|keyboardHidden|orientation"/>
    </application>

    <uses-sdk android:minSdkVersion="7" />

    <supports-screens
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="true"
        android:anyDensity="true"
    />
</manifest>
```

LE MANIFEST DE L'APPLICATION

- Description de l'application
 - Package
 - Version
 - Orientation
- Permissions
- Composants (Activity(s)...)
 - Version minimum
 - Ecrans supportés

LES CLASSES INTENT

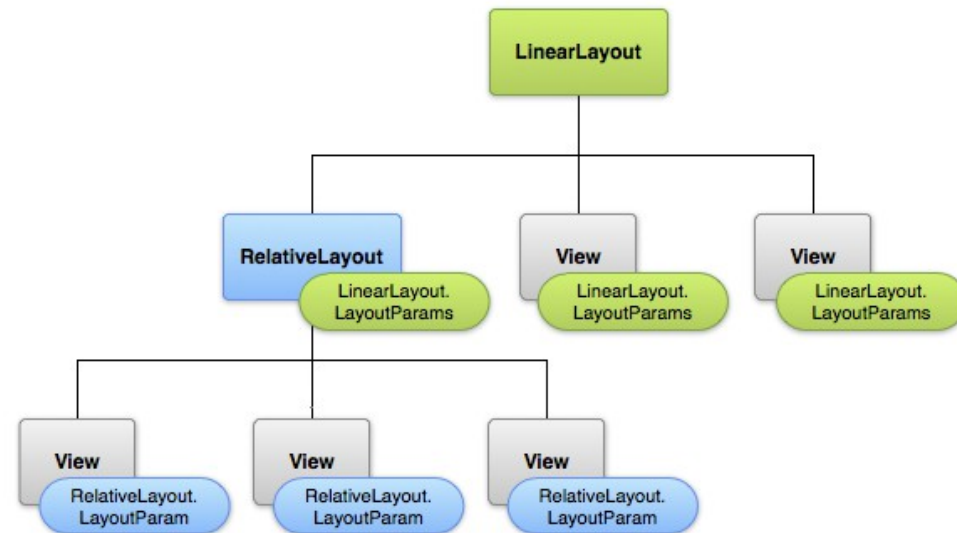
- Android **sandbox** les applications (bac à sable) en les exécutant dans une instance de JVM distincte. Les différentes applications sont donc totalement cloisonnées (sécurité, performance).
- Les *Intent* sont comme des messages. Ils permettent de communiquer entre les applications. Un *Intent* est une déclaration de besoin.
- Un *Intent* est composé principalement de :
 - Une action (`ACTION_VIEW`, `ACTION_EDIT`,..)
 - Une donnée (La donnée à transmettre)
- Un *Intent* peut également être composé de :
 - Category (information additionnelle spécialisant le comportement)
 - Type (MIME de la data)
 - Component (pour préciser une classe spécifique capable de traiter le message)
 - Extras (Bundle d'informations additionnelles)

INTENT & INTENT FILTERS

- Un *Intentfilter* est un trigger, une déclaration de capacité et d'intérêt à prendre en charge un besoin. Il définit la relation entre l'*Intent* et l'application.
- Un *Intentfilter* peut être spécifique à une part des données de l'*Intent* ou à une part de l'action de l'*Intent* ou des deux.
- Quand un *Intent* est émis, le système évalue les *Activity*, *Service*, et *BroadcastReceiver* enregistrés et transmet l'*Intent* au destinataire le plus approprié.

LA CLASSE LAYOUT

- Le Layout est la classe de base pour les **conteneurs** d'éléments **visuels**.
Il contient des **Vues** ou d'autres **Layout**.
- Le **Layout** gère le positionnement des éléments qu'il contient.
- Ils sont représentés sous forme de fichiers XML ou peuvent être instanciés dans le code.
- Il existe plusieurs types de Layout:
 - Le Linear
 - Le relatif
 - Layout avec adapters
 - List View
 - Grid View



LA CLASSE LINEAR LAYOUT

- La classe *LinearLayout* est un *Layout* qui positionne ses fils sur une ligne ou une colonne.
- La direction de l’affichage est fixée par la méthode *setOrientation()* (*LinearLayout.VERTICAL*/*LinearLayout.HORIZONTAL*).
- Le point d’attraction de l’affichage est fixée par la méthode *setGravity()*.
(Pour aligner à droite / haut / bas / gauche /centre de l’écran).
- Vous pouvez également définir une distance pour déplacer l’affichage des fils du *Layout* via la méthode *setPadding()* (*left,top,right,bottom*).
- *SetBackgroundDrawable()* (*Bitmap*) vous permet de changer le fond du *Layout*.

LA CLASSE LINEAR LAYOUT (exemple)

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    //setContentView(R.layout.main);  
  
    layout = new LinearLayout(this);  
    layout.addView(createTextView());  
    layout.addView(createButton());  
  
    layout.setOrientation(LinearLayout.VERTICAL);  
    layout.setGravity(Gravity.RIGHT);  
    layout.setPadding(20, 20, 0, 0);  
  
    BitmapDrawable bmp = (BitmapDrawable) this.getResources().getDrawable(R.  
    layout.setBackgroundDrawable(bmp);  
  
    setContentView(layout);  
}
```

LA CLASSE RELATIVELAYOUT

- La classe *RelativeLayout* est un *Layout* qui positionne chacun de ses fils par rapport aux autres et au Layout lui même.
➔ Pas de dépendances circulaires.

LA CLASSE VIEW

- La classe *View* est la classe de base de l'interface graphique.
- Une *View* occupe une zone rectangulaire sur l'écran et est responsable du dessin et de la gestion des événements d'interaction.
- *View* est la classe de base pour les widgets.
- Les *View* d'une fenêtre sont arrangées sous la forme d'un arbre. Vous pouvez ajouter des *View* depuis le code ou depuis un ou plusieurs fichiers XML *Layout*.
- Il existe de nombreuses classes spécialisées qui fournissent des services pour afficher du texte, des images ...
- La classe *View* est une classe haut niveau.

LA CLASSE VIEW : opérations classiques

- ***Setproperties:*** les propriétés dépendent de la sous-classe de *View* adressée. Quelque soit cette classe les propriétés peuvent être initialisée dans le fichier XML layout. (Ex: le texte de *TextView*).
- ***Setfocus:*** Le framework affecte le focus en fonction des actions de l'utilisateur. Pour forcer le focus sur une vue spécifique, appeler *requestFocus()*.
- ***Setuplisteners:*** Les *View* permettent aux clients de définir des *listeners* qui seront notifiés en cas de changements affectant la *View*. (Ex: changement de Focus *setOnFocusChangeListener()*).
- ***Setvisibility:*** Vous pouvez afficher ou masque une *View* en utilisant *setVisibility(int)*.
- **Note:** Le framework Android est en charge du layout et du dessin des *View*. Vous ne devez pas appeler de méthodes qui effectuent ces actions à moins d'implémenter des *ViewGroup*.

LA CLASSE VIEW : implémenter une View customisée

- Pour implémenter une *View* customisée, vous allez surcharger des méthodes comme *onDraw(android.graphics.Canvas)* ou le constructeur.
- Evènements customisables :
 - CREATION
 - *onFinishInflate()* Appelé après qu'une *View* et ses sous classes soient créées d'un XML.
 - LAYOUT
 - *onMeasure()* Appelé pour déterminer la taille d'une *View* et de ses enfants.
 - *onLayout()* Appelé quand une vue repositionne l'ensemble de ses fils.
 - *onSizeChanged()* Appelé quand la taille de la vue change.
 - EVENT PROCESSING
 - *onKeyDown()* Appelé quand une touche est pressée.
 - *onKeyUp()* Appelé quand une touche est relâchée.
 - *onTrackballEvent()* Appelé quand une action est réalisée sur la trackball.
 - *onTouchEvent()* Appelé quand une action est réalisée sur l'écran tactile.

LA CLASSE VIEW : implémenter une View customisée

- Evènements customisables (suite):

- FOCUS

- *onFocusChanged()* Appelé quand la *View* perd le focus.
 - *onWindowFocusChanged()* Appelé quand la fenêtre contenant la *View* perd ou gagne le focus.

- ATTACHING

- *onAttachedToWindow()* Appelé quand la *View* est attachée à la fenêtre.
 - *onDetachedFromWindow()* Appelé quand la *View* est détachée à la fenêtre.
 - *onWindowVisibilityChanged()* Appelé quand la visibilité de la fenêtre contenant la *View* a changé.

- DRAWING

- *onDraw()* Appelé quand la *View* doit dessiner son contenu.

LA CLASSE VIEW : éléments complémentaires

- Les Ids (fichier XML) permettent d'identifier votre vue
- Le dimensionnement d'une vue peut être vu par 2 types de méthode :
 - Les dimensions souhaitées ou nécessaires
 - *getMeasuredWidth()*
 - *getMeasuredHeight()*
 - Les dimensions réelles
 - *getWidth()*
 - *getHeight()*
- Le layout calcule le positionnement des sous vues en 2 étapes:
 - Une passe de mesure des tailles souhaitées (*getMeasuredWidth()*, *getMeasuredHeight()*)
 - Une passe de calcul de positionnement d'application de contraintes sur les vues.
- Il est possible d'invoquer un dessin de la vue par la méthode *invalidate()*

Les options pour développer sous Android

Vous disposez de plusieurs choix pour vous mettre à Android, de Android Studio à Eclipse en passant IntelliJ ou Netbeans.

Android Studio encapsule le sdk alors que les autres IDE impliquent d'installer l'IDE, puis le SDK Android et ADT, plugin permettant l'intégration du SDK (pour Eclipse).

Merci d'utiliser Eclipse pour vos TP & Projets:

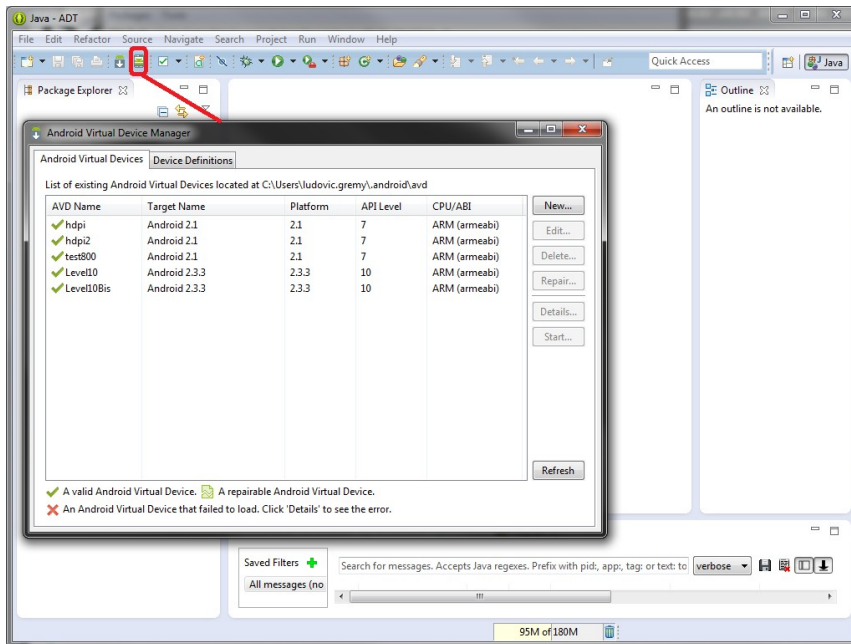
<http://www.eclipse.org/downloads/packages/eclipse-standard-431/keplersr1>

<http://developer.android.com/sdk/installing/index.html>

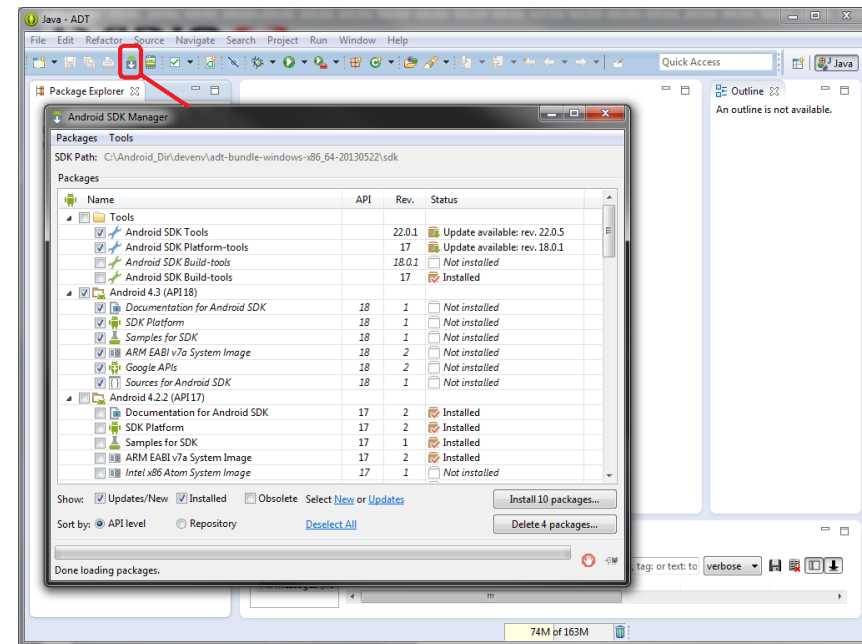
<http://developer.android.com/sdk/index.html#download>

<http://developer.android.com/sdk/installing/installing-adt.html>

AVD MANAGER & ANDROID SDK MANAGER



Définir des émulateurs sur AVD Manager



Mettre à jour le SDK & charger les anciennes versions

Des processus définis et ouverts

DEVELOPPER

- Un sdk intégré sous eclipse pour développer confortablement
- Communauté de développeurs animée (forum, code, open source)
<http://source.android.com/>, <http://android.git.kernel.org/>
- Outils (Emulateur, plugin eclipse, ligne de commande, doc des APIs)

DISTRIBUER

- S'enregistrer en tant que développeur/distributeur
- Sur android market place (développeur 70% - 30 % opérateur), pas de revenu pour google
- Publier vos applications dans votre propre boutique
- Distribuer via un tiers distributeur (Pixtel, cellfish, portail opérateur, ...)

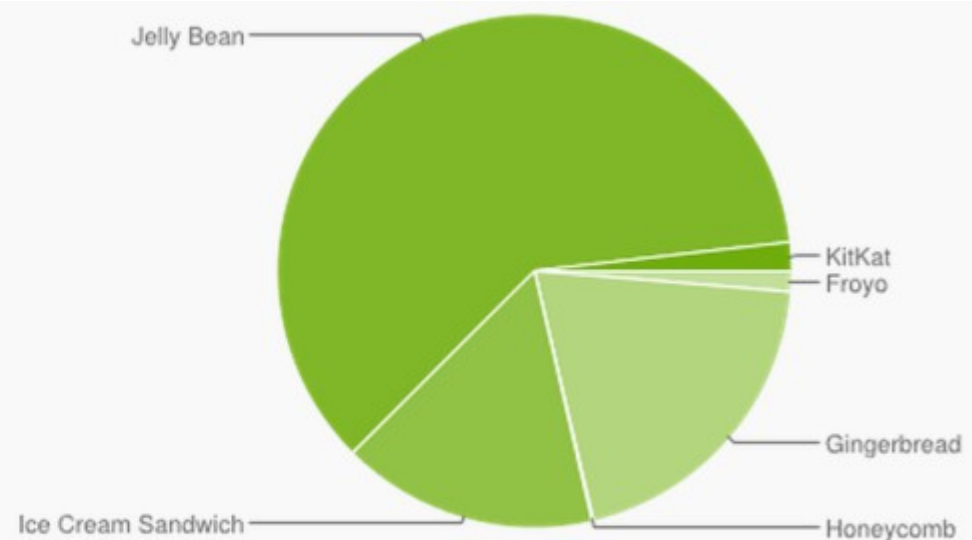
Content type : *[application/vnd.android.package-archive](#)*

De Android 1.5 à 4.4.4 (juin 2014)

Version 5.x en preview pour commercialisation à l'automne

LA FRAGMENTATION DU PARC → GERER LES VERSIONS

Version	Codename	API	Distribution
2.2	Froyo	8	1.3%
2.3.3 - 2.3.7	Gingerbread	10	20.0%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	16.1%
4.1.x	Jelly Bean	16	35.5%
4.2.x		17	16.3%
4.3		18	8.9%
4.4	KitKat	19	1.8%



→ SE CONCENTRER SUR LES VERSIONS 2.3 ET PLUS POUR ETRE MASS MARKET

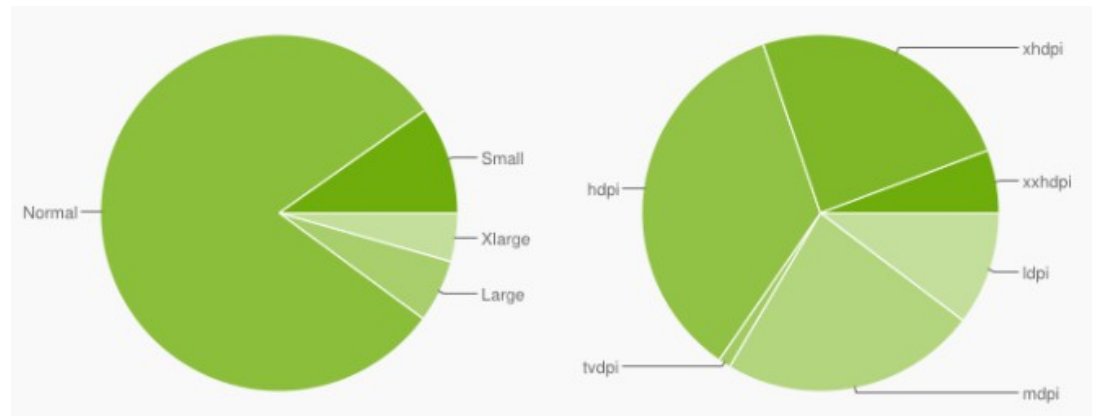
De multiples version de sdk et une variété de terminaux

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	9.7%			0.1%			9.8%
Normal	0.1%	15.9%		34.5%	23.9%	5.7%	80.1%
Large	0.5%	3.2%	1.1%	0.4%	0.5%		5.7%
Xlarge		4.1%		0.2%	0.1%		4.4%
Total	10.3%	23.2%	1.1%	35.2%	24.5%	5.7%	

LES DIFFERENTS ECRANS

- Tailles d'écrans
- Densité
- Gestion du répertoire de ressources

GERER LES CAPACITES DU TELEPHONE



Préciser la compatibilité de son application

- Il est nécessaire de préciser l'attribut **android:minSdkVersion** pour indiquer le SDK nécessaire pour le fonctionnement de l'application.

Non renseigné → Full Compatible !

- En couplant **android:targetSdkVersion** et **android:minSdkVersion**, on indique que l'application a été testée sur une version précise mais qu'elle est compatible avec les versions précédentes jusqu'à **minSdkVersion**.

Exemple pour imposer la version 2.1 :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
package="com.android23.app">
...
<uses-sdk android:minSdkVersion="7" />
</manifest>
```

Numérotation des versions:

```
18 -> Android 4.3
...
7 -> Android 2.1
6 -> Android 2.0.1
5 -> Android 2.0
4 -> Android 1.6
3 -> Android 1.5
2 -> Android 1.1
1 -> Android 1.0
```


Supporter les différentes résolutions

- Documentation Google très exhaustive sur le sujet, appuyez-vous sur l'exemple MultiResolution.
- Pour gérer les différentes résolutions il faut utiliser **<supports-screens>** dans AndroidManifest.xml ainsi que le répertoire **res** qui vous permet de stocker des variantes de ressources.
- Pour les ressources non présentes, le système adaptera les ressources par défaut (**res/drawable**).
- Pensez à :
 - ➔ Tester votre application sur différents gabarits d'émulateurs
 - ➔ Tester la réaction de votre application aux changements d'orientation

Supporter les différentes résolutions

	Low density (120), <i>ldpi</i>	Medium density (160), <i>mdpi</i>	High density (240), <i>hdpi</i>	Extra high density (320), <i>xhdpi</i>
<i>Small screen</i>	QVGA (240x320)		480x640	
<i>Normal screen</i>	WQVGA400 (240x400) WQVGA432 (240x432)	HVGA (320x480)	WVGA800 (480x800) WVGA854 (480x854) 600x1024	640x960
<i>Large screen</i>	WVGA800** (480x800) WVGA854** (480x854)	WVGA800* (480x800) WVGA854* (480x854) 600x1024		
<i>Extra Large screen</i>	1024x600	WXGA (1280x800)[†] 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

Pour en savoir plus: http://developer.android.com/guide/practices/screens_support.html

- Le Logcat -

- Logcat vous permettra d'avoir toutes les informations sur l'exécution de l'émulateur.
- Logcat vous permet de définir plusieurs niveaux de logs qui seront filtrables:
 - ➔ *Verbose*
 - ➔ *Debug*
 - ➔ *Info*
 - ➔ *Warning*
 - ➔ *Error*
- Ajouter Logcat dans votre vue java: Window>ShowView>Others>Logcat
- Appel: *Log.e*(« *groupe* », « *msg* ») ou *Log.i*(« *groupe* », « *msg* »)