



Sécurité Android

Support de cours

ENSIBS – 2^{de} année

Matthieu Le Lain

Objectif

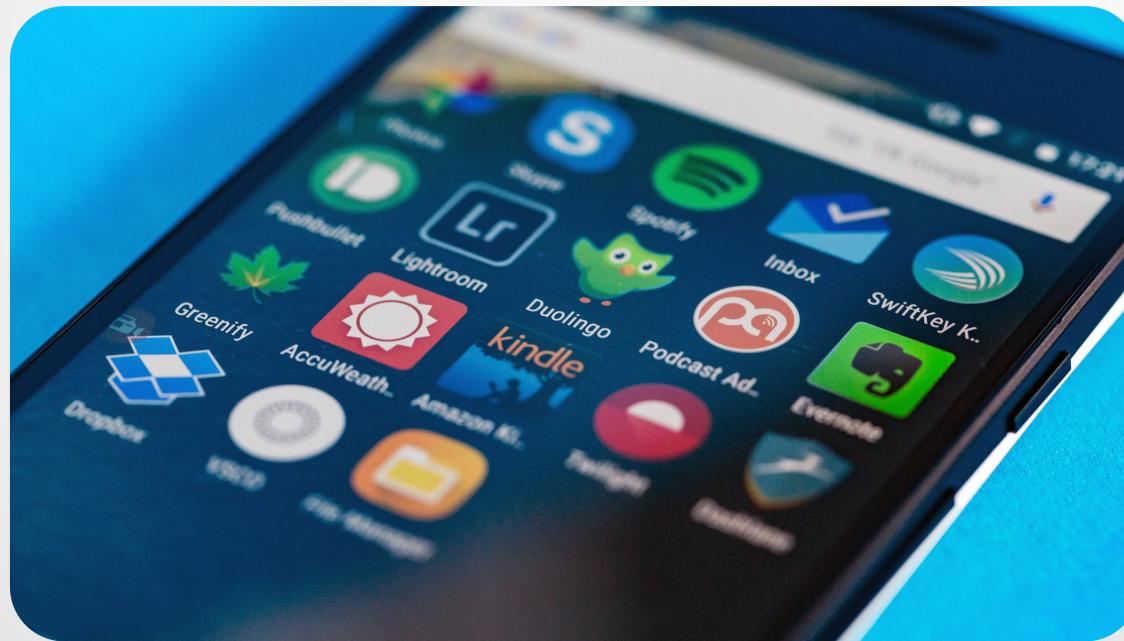
Ce support de cours se décompose en plusieurs parties à suivre
d'une manière générale dans l'ordre.

L'objectif global de ce cours est de vous permettre de pouvoir

**appréhender la sécurité au sein de
systèmes d'exploitations embarqués mobiles**

Périmètre

Le système d'exploitation étudié concernera le système Android.



Pour toutes questions, remarques, découvertes de coquilles,
envoyer un mail à mon adresse **matthieu.le-lain@univ-ubs.fr**,

Plan global

- Cours : Bases du développement Android
 - Comprendre l'architecture Android
 - S'acclimater avec l'environnement de développement
 - Savoir développer une application de base
 - Aspects sécurités
- TP : Etude et analyse des concepts de la sécurité Android

Plan – Chapitre 1

- 1. Architecture d'Android et création d'application
 - 1.1 Présentation du système Android
 - 1.2 Environnement de développement
 - 1.3 Manifeste d'application
 - 1.4 Notion d'activité
 - 1.5 Notion de ressource

Chapitre N° 1

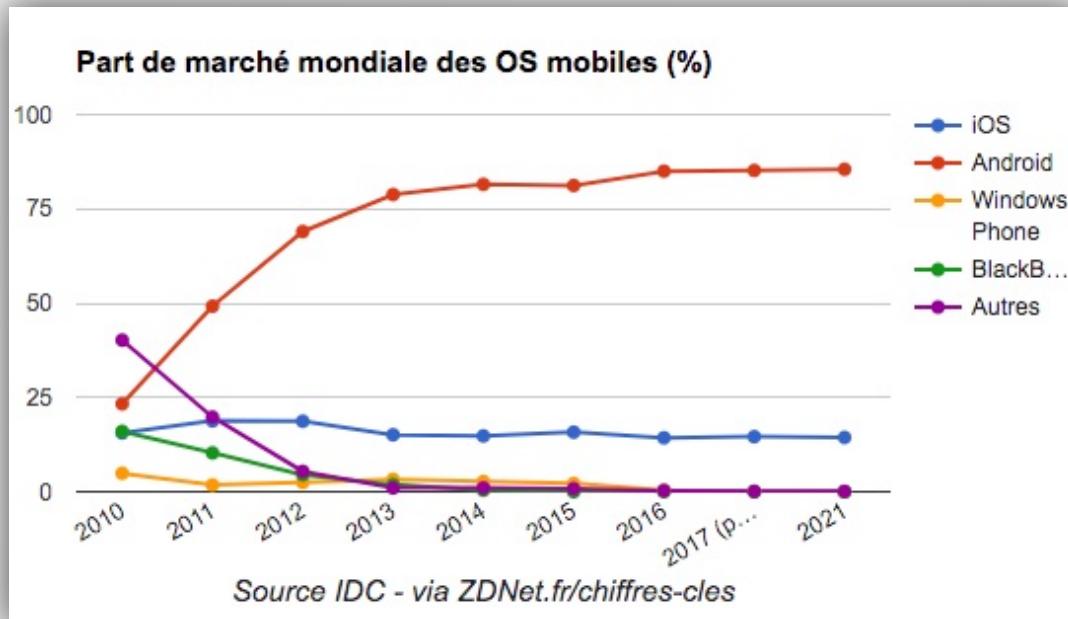
Architecture d'Android et création d'application

1.1 – Présentation du système Android



La mobilité, mais pourquoi ?

- Marché : émergence des téléphones mobiles.



- Popularité croissante des plateformes mobiles telles que les tablettes et objets connectés (montre, bracelets, lunettes, domotique).
- Multiplicité des capacités et applications des plateformes mobiles (appareil photo, écran tactile, capteur GPS, portefeuille...)

Présentation

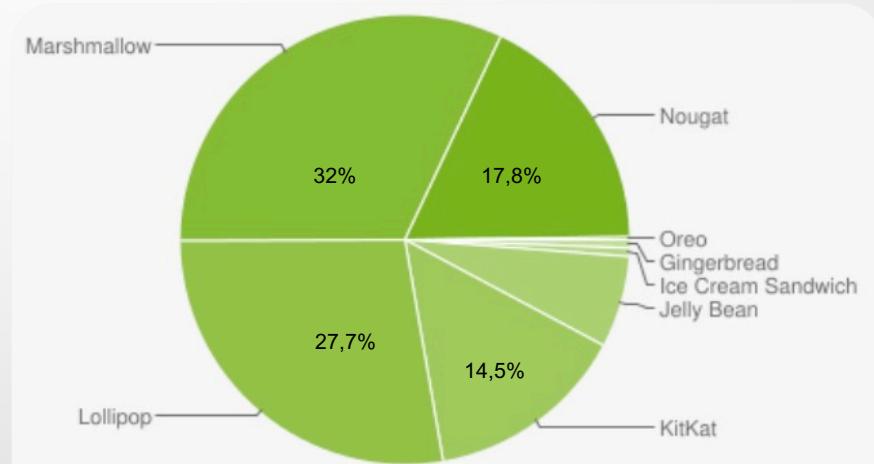


- Android est une pile logicielle open source incluant un **système d'exploitation**, un **middleware**, **des applications mobiles clés** ainsi qu'un **ensemble de bibliothèques d'API** destinées au développement d'applications adaptés aux matériels mobiles.
- Intégré dans de multiples supports (App. Mobiles, Télévision, GPS, caméras...).
- **Des applications natives aux applications tierces**, l'**écriture se fait grâce aux API et au moteur d'exécution de la plateforme Android**. Cela permet d'utiliser toutes les capacités des appareils tels que les capteurs, mais aussi toutes les méthodes de communication.
- Plateforme **ouverte** et complète.

Histoire

- Système initialement prévu pour être un OS d'appareil photo proposé (gratuitement et librement modifiable) aux fabricants de téléphones mobiles (2003).
- Startup (Android) racheté par Google en 2005.

Date	Versions
2009	Donut (1.6)
2009	Éclair (2.0)
2011	Honeycomb (3.0)
2011	Ice Cream Sandwich (4.0)
2014	Lollipop (5.0)
2015	Marshmallow (6.0)
2016	Nougat (7.0)
2017	Oreo (8.0)



Source : <http://www.numerama.com/tech/132165-les-versions-dandroid-les-plus-utilisees.html>

Aujourd’hui

- Maintenu par OHA (Open Handset Alliance) depuis 2007.
 - **Objectif** : Développement de normes ouvertes pour les appareils de téléphonie mobile
 - Fabricant semi-conductrice (Intel, ARM, Qualcomm...)
 - Fabricant Téléphone mobile (Asus, Acer, Dell, HTC...)
 - Éditeur logiciel (Google, Ebay..)
 - Opérateur téléphonique (Vodafone, Bouygues ..)
 - Distributeurs (TAT, Wind River...)

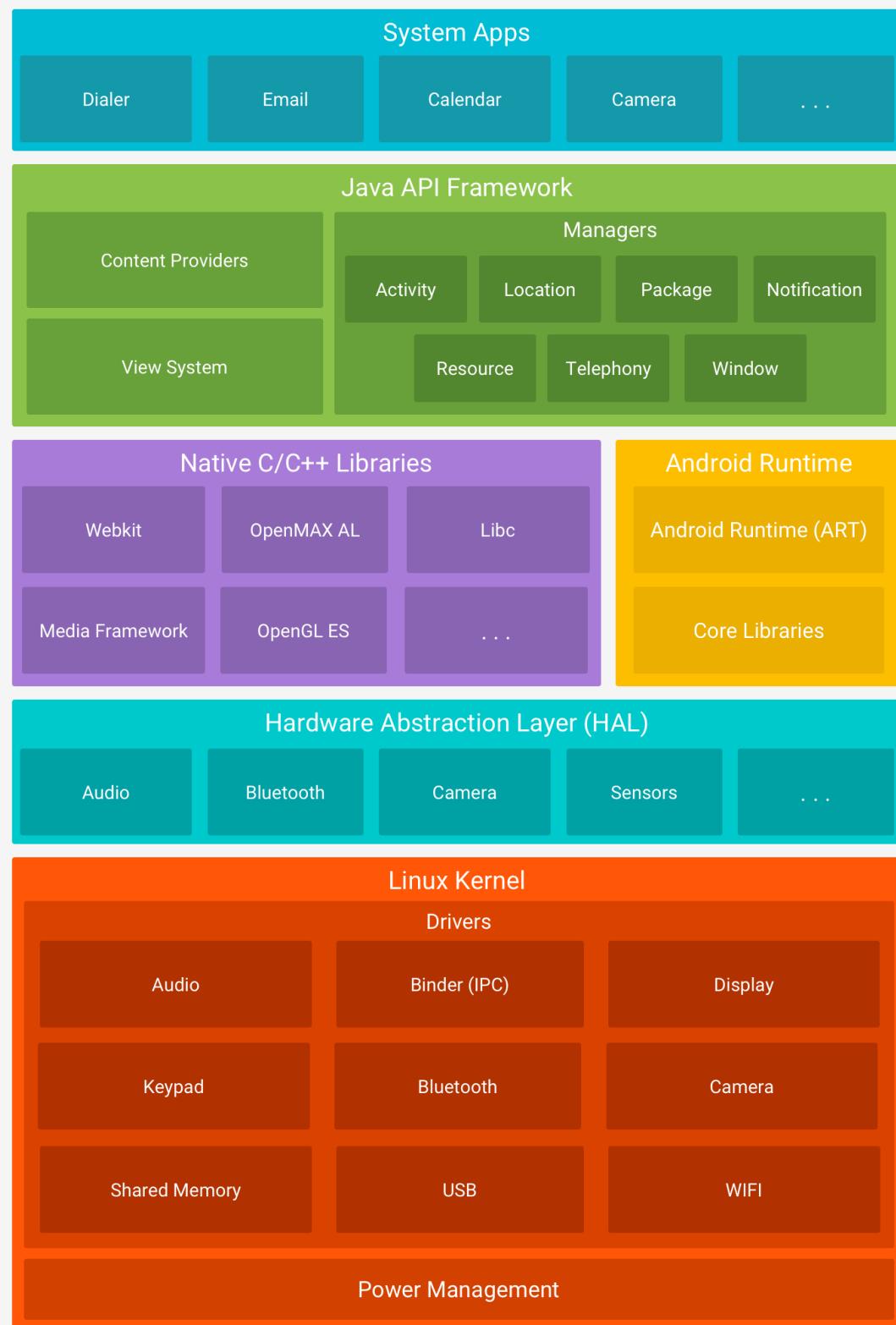
Constitution détaillée (1/2)

- Modèle de conception matérielle de référence décrivant les capacités nécessaires à un appareil mobile pour supporter la pile logicielle.
- Un système d'exploitation Linux assurant l'interface de bas niveau avec le matériel, la gestion de la mémoire et le contrôle des processus, de manière optimisée pour les mobiles
- Des bibliothèques Open-Source pour le développement d'applications, telles que SQLite, Webkit, OpenGL et un gestionnaire de médias.
- **Depuis la version 5.0, Android s'appuie sur une nouvelle machine virtuelle appelée ART, Android RunTime en remplacement de Dalvik.**

Constitution détaillée (2/2)

- Un framework applicatif qui expose de façon agnostique les services du système à la couche applicative (ex : Window manager, Location Manager, Content Providers, téléphone, capteurs...).
- Un framework pour l'interface utilisateur utilisé pour héberger et lancer les applications.
- Des applications essentielles préinstallées (courrier, gestion SMS, calendrier, gestionnaire de contact, navigateur internet, lecteur de musique, horloge...).
- Un kit de développement logiciel (**Software Developpement Kit – SDK**), utilisé pour créer des applications et comprenant des outils, des plug-ins et de la documentation
- Un kit de développement natif (Native Development Kit - NDK), en C/C++. Il permet de générer du code natif et contient un outil de création de fichier apk.

Architecture



1.2 – Environnement de développement



Environnement de développement

L'écriture d'application nécessite le SDK d'Android, ainsi que le kit de développement JAVA (JDK). Pour faciliter les développements, il est également préférable d'utiliser un IDE, à savoir Android Studio, basé sur IntelliJ IDEA.

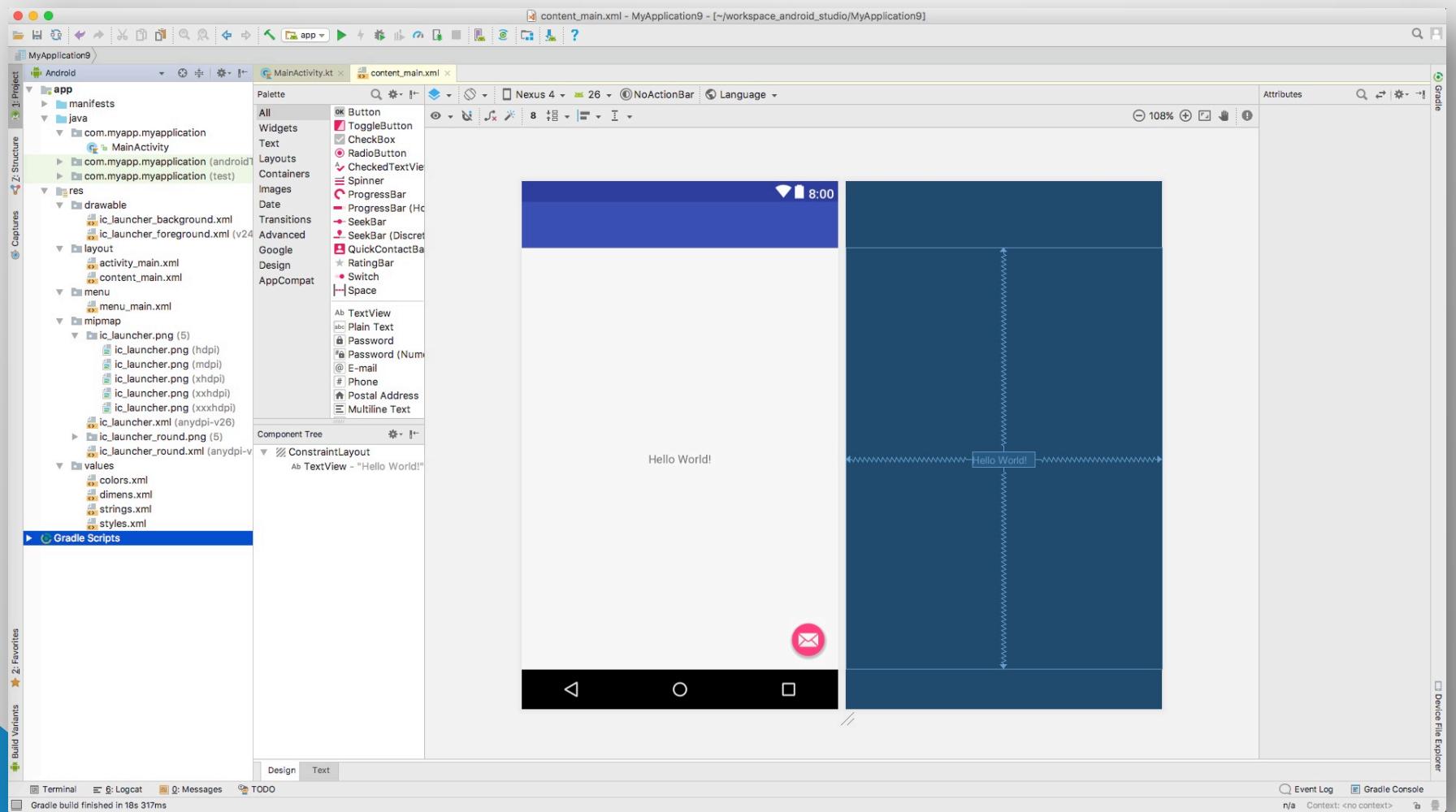
Android Studio est développé par Google principalement orienté vers le développement mobile sous Android.

Le SDK Android fonctionne sous la majorité des OS (Windows, Linux, Mac OS).

Le développement des applications se fait en JAVA.

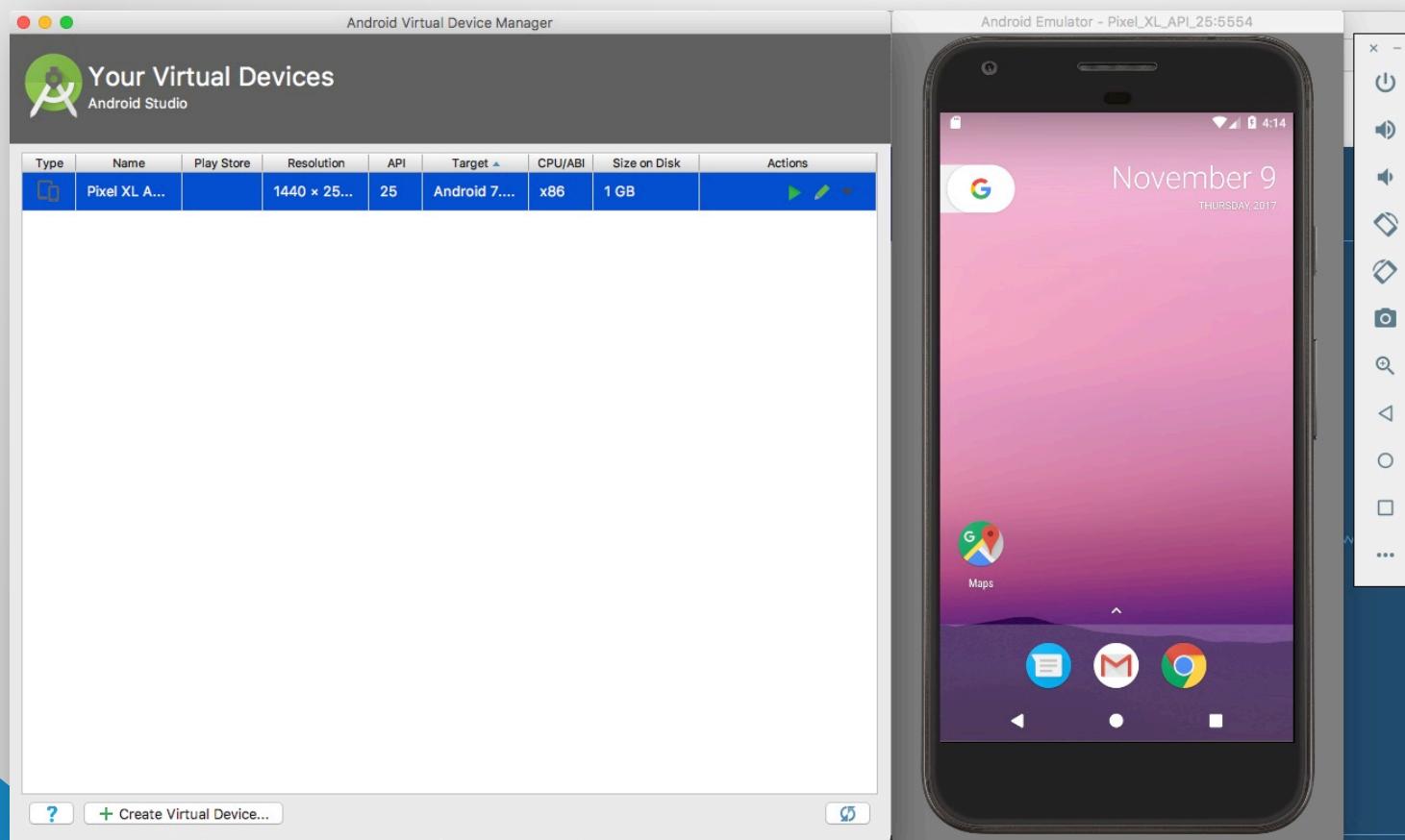
Android Studio

Par exemple, cela permet de construire facilement des interfaces grâce aux outils WYSIWYG (What You See Is What You Get) de l'éditeur de ressources.



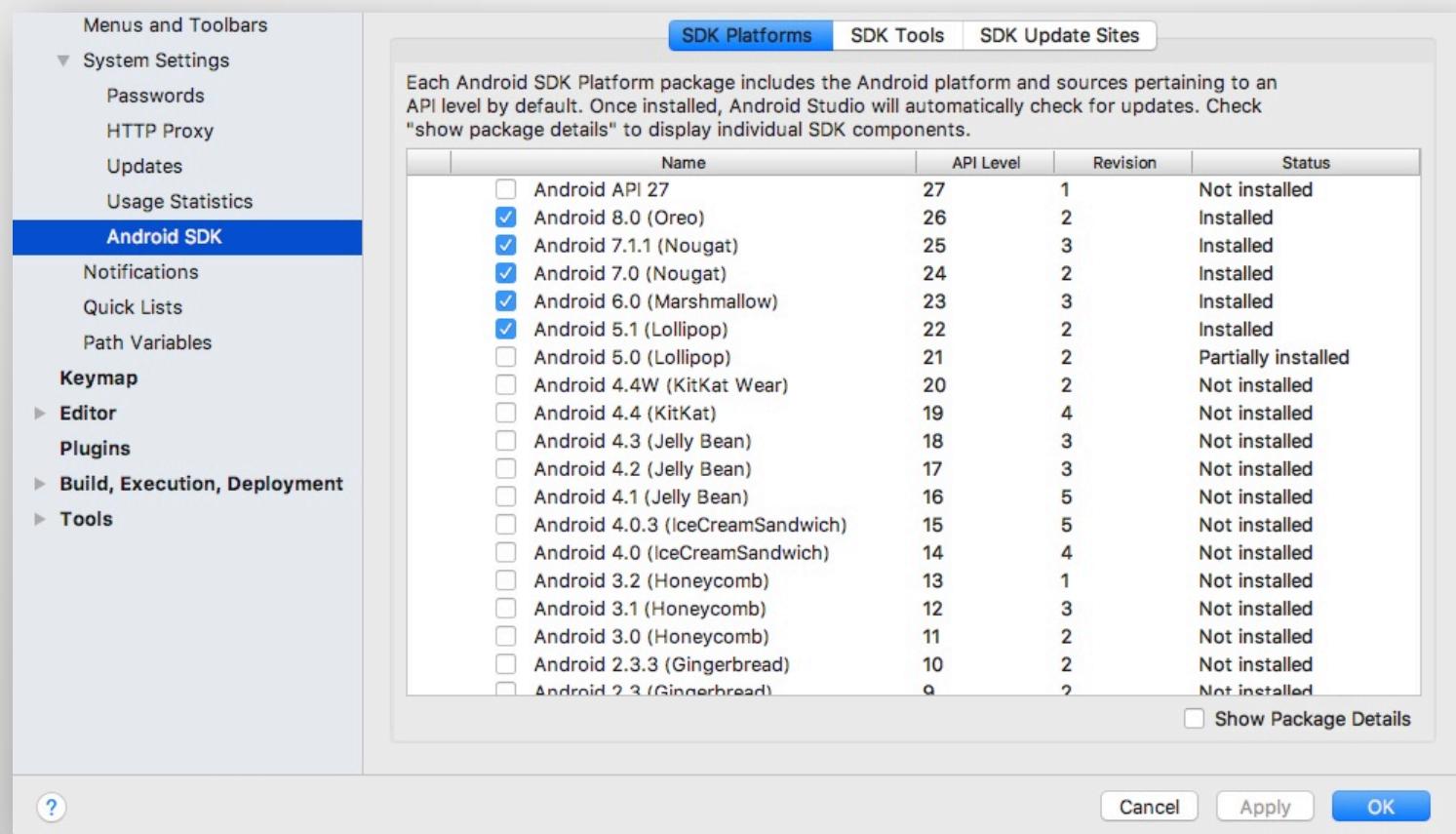
AVD – Android Virtual Device

Pour évaluer vos développements, il est possible de simuler l'environnement d'un mobile avec le gestionnaire de terminaux virtuel (AVD) ou en branchant directement vos périphériques mobiles à votre plateforme de développement. Il sera nécessaire cependant d'activer les options développeurs sur votre périphérique mobile.



Software Development Kit

Le SDK contient plusieurs outils, dont le **SDK manager** permettant de télécharger et mettre à jour les sources d'Android. C'est à partir de ce dernier qu'il est également possible de télécharger différentes librairies facultatives. Voir lien en annexe. Ce dernier est intégré à Android Studio.



Gradle

- Android Studio utilise Gradle pour :
 - La gestion des dépendances
 - L'automatisation de la compilation
 - La création de l'APK
 - Le déploiement de l'APK
- Configuration générale et modulaires par
 - Settings.gradle (contient les modules du projet)
 - Build.gradle (configurations communes à tous les modules du projet)



1.3 – Manifeste d'application



Manifeste

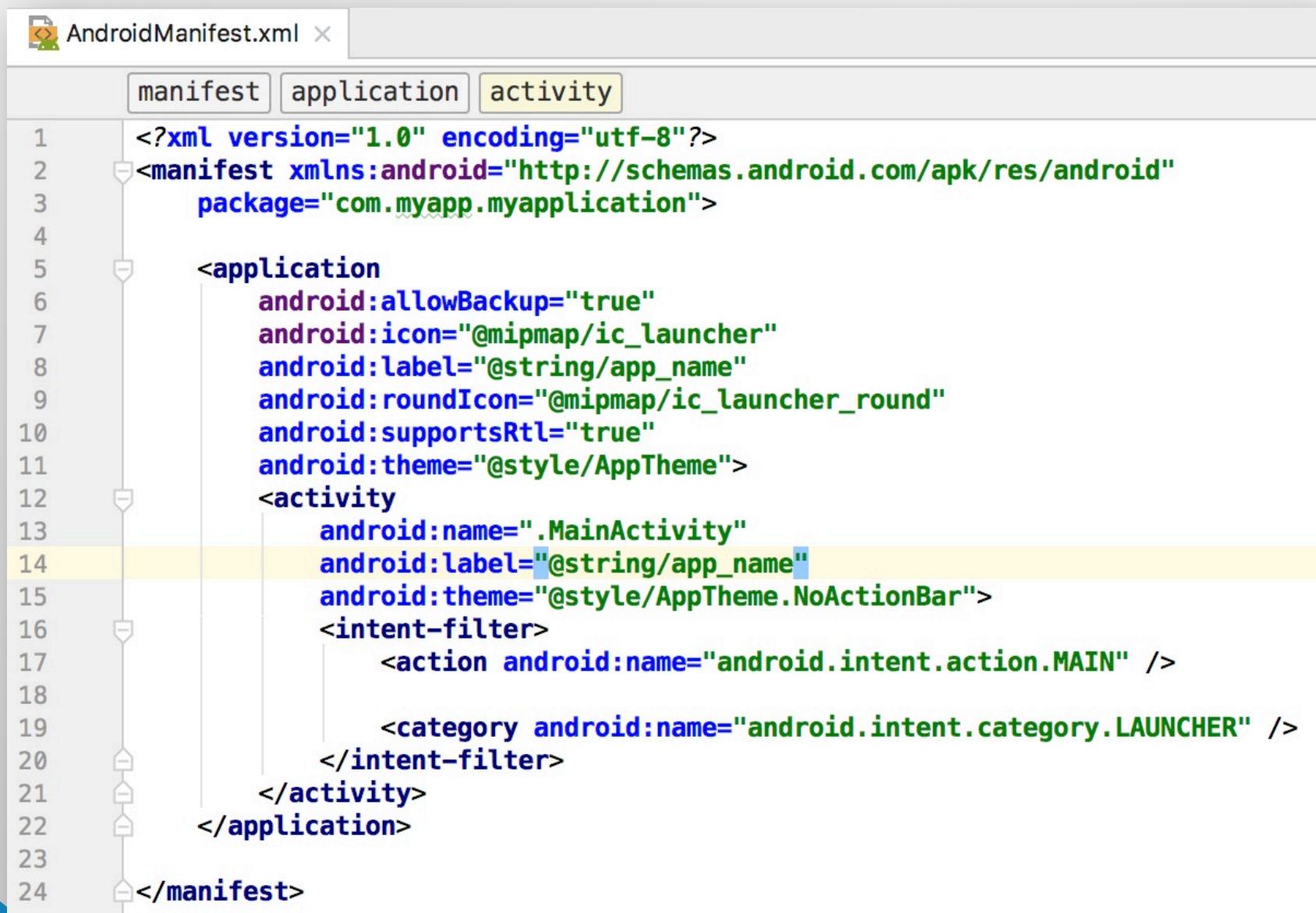
À la racine de chaque projet Android se trouve un fichier manifeste (AndroidManifest.xml). Ce fichier permet de décrire les informations générales de l'application :

- Version de l'application, du SDK, du thème
- Icône

Mais aussi les permissions et configurations. Ce fichier de métadonnées est construit sous la forme de nœud pour chaque composant :

- Activités
- Services
- Récepteur de diffusion
- Fournisseur de contenu

Manifest - Structure



The screenshot shows the AndroidManifest.xml file in an IDE. The code is color-coded and indented to show its structure. A yellow highlight covers the entire `<activity>` element, specifically from line 14 to line 20.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3   package="com.myapp.myapplication">
4
5   <application
6     android:allowBackup="true"
7     android:icon="@mipmap/ic_launcher"
8     android:label="@string/app_name"
9     android:roundIcon="@mipmap/ic_launcher_round"
10    android:supportsRtl="true"
11    android:theme="@style/AppTheme">
12     <activity
13       android:name=".MainActivity"
14       android:label="@string/app_name"
15       android:theme="@style/AppTheme.NoActionBar">
16       <intent-filter>
17         <action android:name="android.intent.action.MAIN" />
18
19         <category android:name="android.intent.category.LAUNCHER" />
20       </intent-filter>
21     </activity>
22   </application>
23
24 </manifest>
```

Manifeste – Balises globales

uses-sdk : Version minimale, maximale et cible du SDK nécessaire sur l'appareil pour le fonctionnement de l'application.

Les attributs **minSdkVersion** et **maxSdkVersion** permettent de restreindre l'utilisation de l'application sur les appareils.

```
1 <uses-sdk android:minSdkVersion="8"  
2           android:targetSdkVersion="13">  
3 </uses-sdk>
```

Manifeste – Balises globales

uses-configuration : Indique les combinaisons de dispositifs d'entrée supportée par l'application. Utilisé par exemple dans le cas de jeux qui exigent des contrôles d'entrée particuliers.

```
1 <uses-configuration  
2   android:reqTouchScreen="finger"  
3   android:reqNavigation="trackball"  
4   android:reqHardKeyboard="true"  
5   android:reqKeyboardType="qwerty"/>
```

reqTouchScreen : Type d'écran tactile (*notouch, stylus, finger, undefined*)

reqNavigation : Périphérique de navigation requis (*nonav, trackball, whel, dpad, undefined*)

reqHardKeyboard : Requiert un clavier physique (*true / false*)

reqKeyboardType : Type de clavier (*nokeys, qwerty, twelvekey, undefined*)

reqFiveWayNav : Requiert un périphérique d'entrée (*true / false*)

Manifeste – Balises globales

uses-feature Cette balise permet de préciser les spécificités matérielles requises par l'application. Cela permet par exemple de ne pas installer une application qui ne dispose pas d'un capteur essentiel. Ex : Pas d'accéléromètre → pas de CardBoard !

```
1 <uses-feature  
2   android:name="android.hardware.camera"/>
```

sensors : capteurs physiques

Bluetooth : connectivité Bluetooth requise

camera : Appareil photo (mais aussi autofocus, flash, capteur frontal)

audio : traitement audio à fiable latency

Location : Géolocalisation (réseau ou GPS)

NFC : Connexion en champ proche

wifi : Connexion Wifi

telephony : besoin de la fonction téléphone

microphone : enregistrement de sons

Liste des matériels : <http://developer.android.com/guide/topics/manifest/uses-feature-element.html#features-reference>

Manifeste – Balises globales

supports-screen : Cette balise permet de préciser les dimensions d'écrans qui peuvent être supportés par l'application. Dans le cas d'appareils non supportés, le système applique parfois un « mode de compatibilité ».

```
1 <supports-screens android:smallScreens="false"  
2           android:normalScreens="true"  
3           android:largeScreens="true"  
4           android:xlargerScreens="true"  
5           android:requiresSmallestWidthDp="480"  
6           android:compatibleWidthLimitDp="600"  
7           android:largestWidthLimitDp="720"/>
```

Manifeste – Permissions

<uses-permission: Cette balise permet d'indiquer les permissions nécessaires à une application pour qu'elle fonctionne convenablement et sont nécessaire pour un certain nombre de services natifs d' Android. C'est un véritable aspect de sécurité à prendre en compte. Les permissions seront ainsi présentées à l'utilisateur avant l'installation. Elles lui permettront de détecter les applications trop intrusives, comme une lampe torche ayant besoin d'un accès à l'identité du téléphone, aux SMS, au carnet de contact...

```
1 <uses-permission android:name="android.permission.INTERNET" />
```

La capture d'écran ci-dessus montre une permission d'avoir accès à la connexion internet du matériel. Il en existe un nombre élevé, écriture sur le périphérique, accès internet, accès au carnet d'adresses, accès au vibreur, accès au calendrier, etc..

La liste des permissions se trouve sur la documentation officielle Android :
<http://developer.android.com/reference/android/Manifest.permission.html>

Manifeste – Vos Permissions

<permission: Il est également possible de déclarer ses propres permissions, pour autoriser par exemple un service tiers à accéder aux données de votre application. Pour cela il faut préciser plusieurs attributs au sein de la balise permission, nom, niveau d'accès, libellé, description du risque lié à la permission.

```
1 <permission android:name="com.monservice.mapermission"  
2     android:protectionLevel="dangerous"  
3     android:label="Ma permission"  
4     android:description="@string/description_mapermission">  
5 </permission>
```

Il est ainsi nécessaire de préciser cette permission dans l'application ou le service tiers voulant accéder à ces données.

```
1 <activity  
2     android:name=".ActivityTiers"  
3     android:label="@string/app_name"  
4     android:permission="com.monservice.mapermission"  
5 </activity>
```

Manifeste – Application

<application : La balise « application » permet de définir toutes les informations relatives à votre application. On y retrouve l'icône (affiché sur le bureau de la plateforme), le logo, le nom de l'application le thème utilisé, mais aussi l'activation ou non du mode débogage.

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

Au sein de cette balise se trouveront les balises servant à indiquer la liste des composants de l'application, à savoir activity, service, provider, receiver.

Manifeste – Application – Activity

<activity : Chaque activité qui sera effective dans l'application devra être déclarée dans un de ces nœuds « activity », tout comme les « services », « provider », « receiver ».

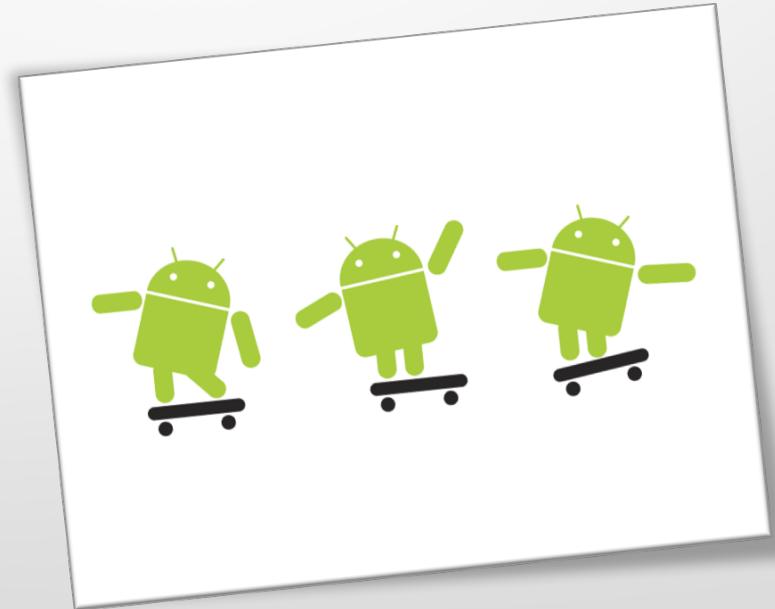
Ces nœuds sont composés de plusieurs attributs comme le nom de l'activité et son label, mais également des balises **<intent-filter>** qui permettent de préciser les intentions lançant l'activité.

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

    <activity
        android:name=".MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

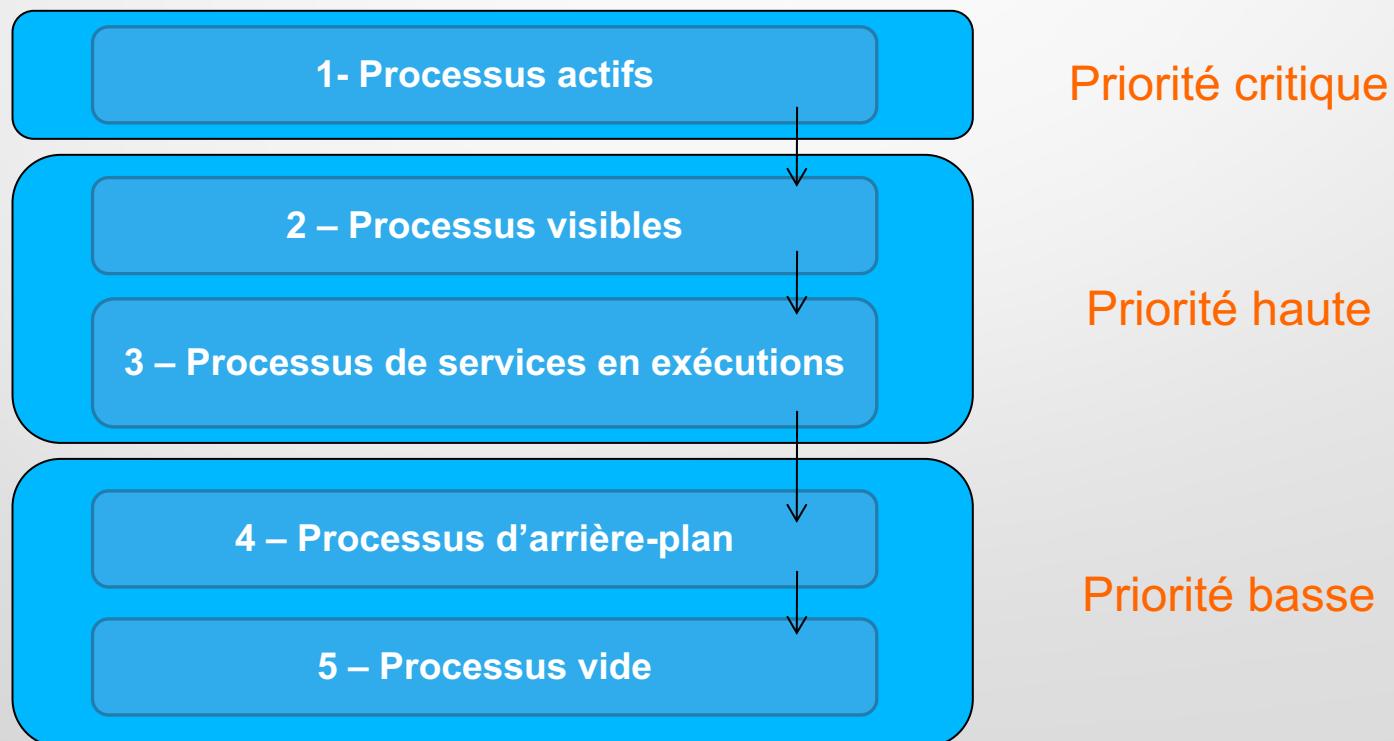
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

1.4 – Notion d'activité



Activité

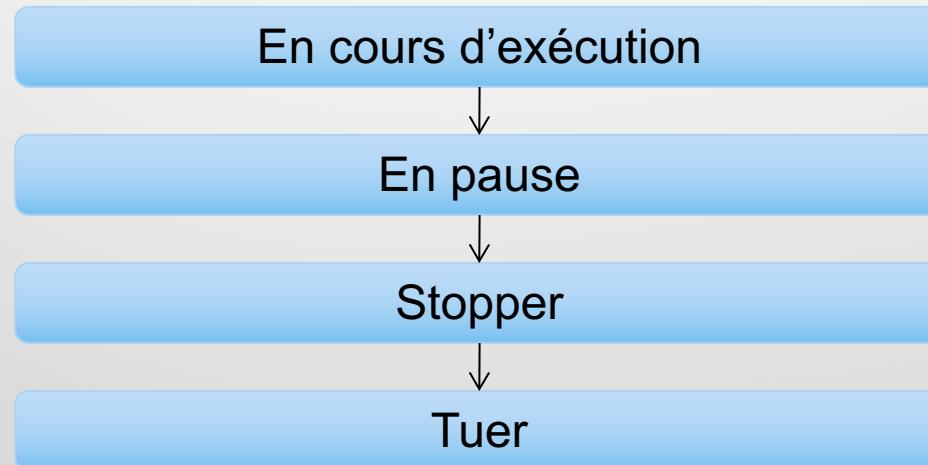
Chaque application est exécutée dans son propre processus. À tout moment le framework Android peut décider de tuer le processus pour libérer de l'espace ou de la ressource pour d'autres applications, considérées comme plus prioritaires, dans le cas où votre application n'est plus au premier plan. La figure ci-dessus décrit les priorités de processus.



Activité

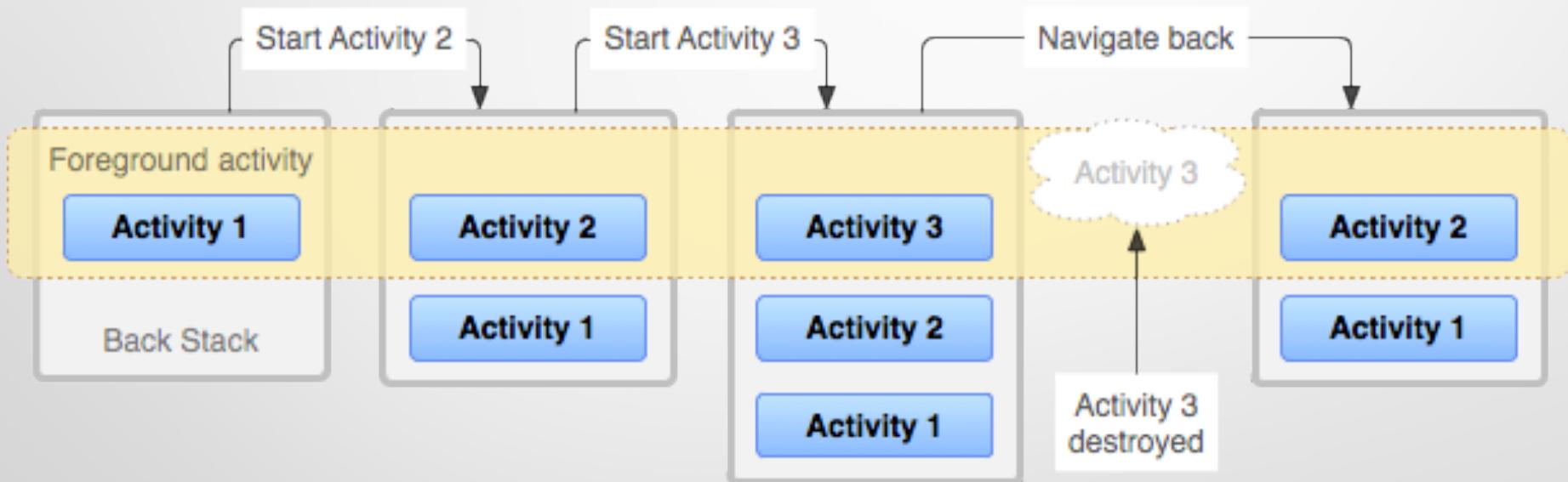
Au sein de chaque application Android se trouvent plusieurs composants faiblement couplés et dont les interactions sont décrites dans le manifeste. Les activités font partie de ces composants. Une activité correspond à la couche présentation d'une application. Elles utilisent des vues et dans les versions récentes des fragments, et servent à mettre en œuvre des interfaces et interagir avec les actions de l'utilisateur.

Une activité est une classe qui étend la classe « Activity ». Comme vu précédemment, il est indispensable de déclarer chaque activité au sein du manifeste. Cette activité peut se retrouver dans plusieurs états :



Activité – Back Stack

Chaque activité lancée est empilée dans ce que l'on appelle la « Back Stack », autrement dit une pile LIFO (Last In First Out).

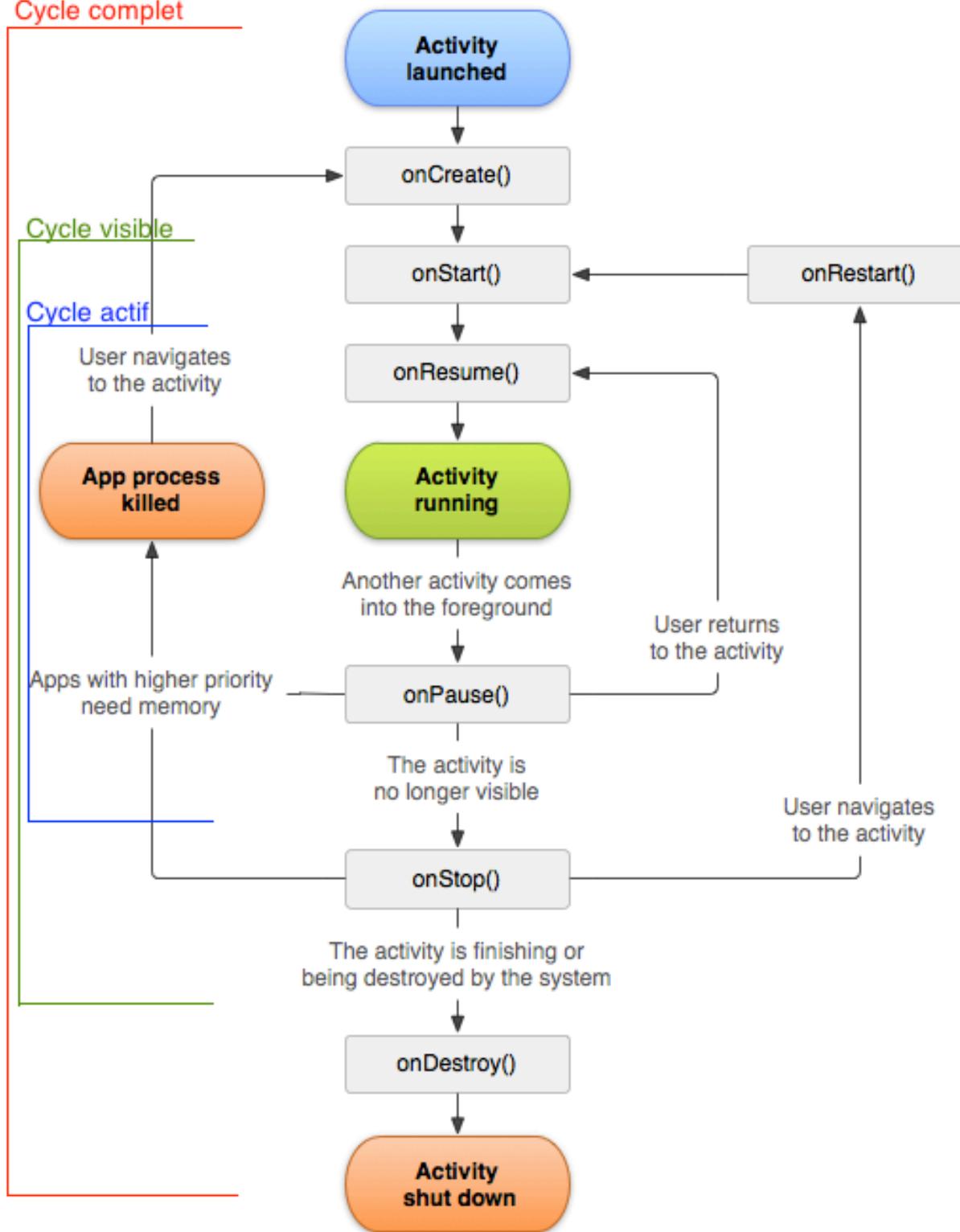


Activité – Cycle de vie

Il est important de prendre en compte les changements qui peuvent être opérés volontairement par l'utilisateur, ou involontairement si c'est un choix fait par le système, dans la programmation des activités.

Ainsi pour chaque changement d'état, plusieurs méthodes correspondantes ont été prévues. **Il est donc indispensable de bien comprendre le cycle de vie des activités pour développer efficacement les applications.**

Cycle complet



Activité – Cycle de vie

```
1 @Override
2 public void onCreate(Bundle savedInstanceState){ // Appelé au début du cycle complet
3     super.onCreate(savedInstanceState);
4 }
5
6 @Override
7 public void onRestoreInstanceState(Bundle savedInstanceState){ // Restaure l'interface de l'utilisateur
8     super.onRestoreInstanceState(savedInstanceState);
9 }
10
11 @Override
12 public void onRestart(){ // Appelé avant les cycles visibles d'un processus activité
13     super.onRestart();
14 }
15
16 @Override
17 public void onStart(){ // Appelé au début du cycle visible
18     super.onStart();
```

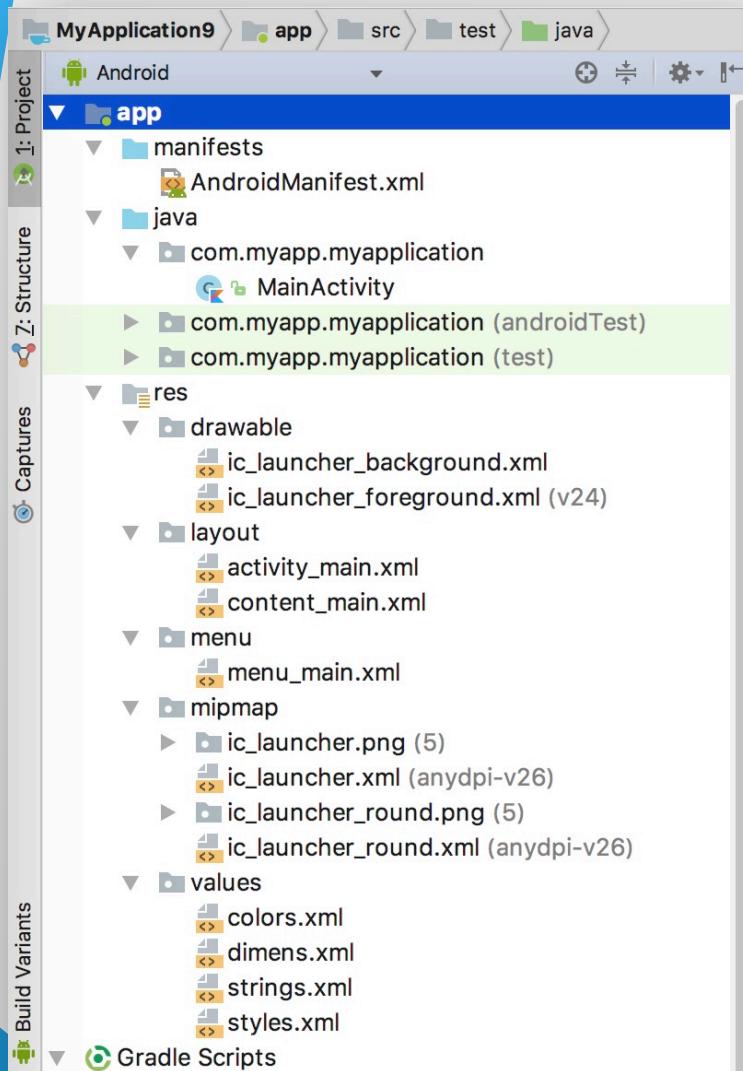
Activité – Cycle de vie

```
1 @Override
2 public void onResume(){ // Appelé au début du cycle actif
3     super.onResume();
4 }
5
6 @Override
7 public void onSaveInstanceState(Bundle savedInstanceState){ // Sauvegarde les changement d'état de
8     super.onSaveInstanceState(savedInstanceState);           // l'interface à la fin du cycle actif
9 }
10
11 @Override
12 public void onPause(){ // Appelé à fin du cycle actif
13     super.onPause();
14 }
15
16 @Override
17 public void onStop(){ // Appelé à la fin du cycle visible
18     super.onStop();
19 }
20
21 @Override
22 public void onDestroy(){ // Appelé à la fin du cycle complet
23     super.onDestroy();
24 }
```

1.5 – Notions de ressources



Ressources



Les ressources, autrement dit les données d'une application Android, se trouvent **dans le dossier « res »** de tout projet Android. Ce dossier **contient plusieurs types de ressources**, allant de la simple chaîne de caractères aux images, au total il y a 9 types de ressources primaires (valeurs simples, drawables, layouts, animation, styles, menus, recherches, XML, ressources brutes).

À chaque ajout d'une ressource au sein du projet, celle-ci est précompilée et sa référence est ajoutée au fichier R.java, ce dernier **décrit toutes les ressources !**

Convention : Les noms des fichiers ne doivent contenir que des minuscules, chiffres, points et underscore. Aucun espace n'est accepté.

Ressources – values

Les valeurs simples se trouvent dans des fichiers XML, au sein du dossier « res/values ». On retrouve ainsi les chaînes de caractères dans le fichier « strings.xml », les couleurs dans le fichier colors.xml, des dimensions de tableaux dans le fichier « dimens.xml »...

Dossier values

strings.xml : chaîne de caractères
colors.xml : couleurs
styles.xml : style du thème



Les autres dossiers commençant par values-xxx permettent de remplacer les fichiers par défaut du dossier values lorsque la configuration de l'appareil est particulière. Par exemple, le dossier values-v11 contient les fichiers qui remplacent les fichiers par défaut lorsque l'appareil est sous une version d'Android 3.X.

Le dossier mipmap contient les icônes de l'application dans les différentes résolutions.

Ressources – strings.xml

```
1 <resources>
2   <string name="applicaiton_name">Mon application</string>
3   <string name="title_activity_main">MainActivity</string>
4   <string name="secondary_notconnected">Aucun écran secondaire n'est connecté.</string>
5   <string name="change_color">Changer la couleur</string>
6 </resources>
```

Ressources – colors

colors.xml

```
9 <?xml version="1.0" encoding="utf-8"?>
10 <resources>
11   <color name="menu_title_color">#CCC333</color>
12   <color name="background_color">#FF00FF</color>
13   <color name="text_button">#FF99CC</color>
14 </resources>
```

strings.xml

Les ressources couleur peuvent être déclarées au sein de deux fichiers, soit dans le fichier **colors.xml**, ou dans le fichier **strings.xml**.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <color name="red">#FF0000</color>
4   <color name="green">#33FF00</color>
5   <color name="blue">#00FFFF</color>
6 </resources>
```

Ressources – drawable

(définition d'écran)

Le dossier drawable contient tous les composants graphiques utilisés par l'application, de l'icône du bureau aux images internes, mais aussi des « shapes » autrement dit des formes. Il existe plusieurs dossiers correspondants aux diverses définitions d'écrans et chacun d'entre eux contient donc diverses résolutions d'images.

drawable-ldpi

- Basse résolution - 120 dpi

drawable-mdpi

- Résolution moyenne – 160 dpi

drawable-hdpi

- Haute résolution – 240 dpi

drawable-xhdpi

- eXtra Haute résolution – 320 dpi

drawable-xxhdpi

- eXtra eXtra Haute résolution – 480 dpi

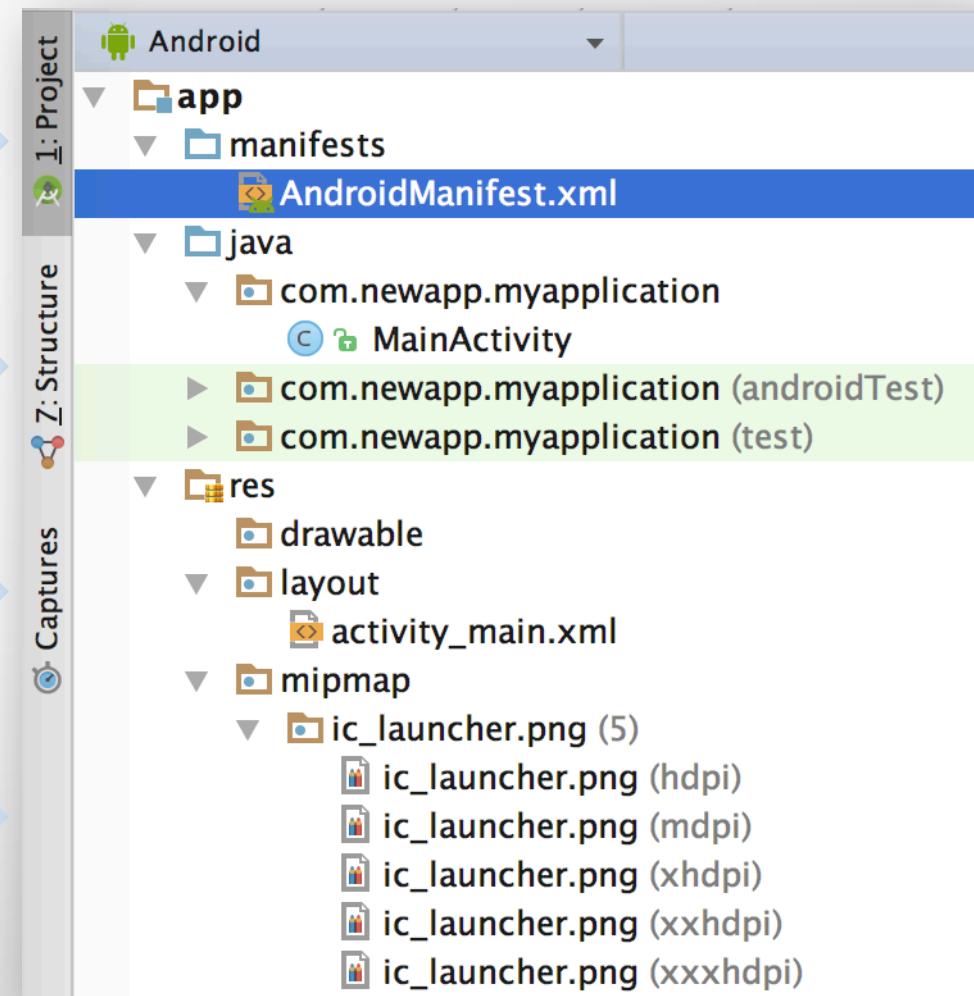
drawable-xxxhdpi

- eXtra eXtra eXtra haute résolution – 640 dpi

Ressources – drawable

(Résolution minimum d'écran en dp)

- Small • 426 x 320
- Normal • 470 x 320
- Large • 640 x 480
- eXtra large • 960 x 720



Ressources – Accès

L'accès aux ressources se fait ainsi par la classe statique R. Cette classe est générée à partir des ressources externes et créée à la compilation.

Pour accéder à une ressource, il est donc nécessaire d'utiliser une instance de la classe Resources qui contient les méthodes (accesseurs, etc.) d'accès à ces ressources. Le référencement d'une ressource dans une classe est de la forme : **R.type.identificateur**. Par exemple R.string.nom_application, comme le montre l'image ci-dessous :

```
2 //récupération de la chaîne de caractère "application_name"  
3 //du fichier strings.xml  
4 Resources maResources = get Resources();  
5 String nomApplication = maResources.getText(R.string.application_name);  
6  
7 //récupération de la couleur "background_color"  
8 // encodé sous la forme d'un entier !  
9 int couleurFond = maResources.getColor(R.color.background_color);
```

Attention cependant, le référencement d'une ressource dans un fichier XML (type layout) est lui de la forme : **@type/ressources**. Par exemple : @color/blue.

Ressources – layout

Les layouts (gabarits), en tant que ressources, permettent de créer des interfaces en XML. Autrement dit, d'une manière globale, l'interface utilisateur de tout composant visuel, activité, fragment, widget..

Ce découplage très pratique permet ainsi de mettre au point des interfaces statiques au sein de ces layouts (Boutons, texte...) et de gérer toute la partie dynamique au sein des fichiers java (comme activity.java) pour les interactions et traitements.

Ces fichiers XML de layout, se trouvent dans le dossier de ressources « **res/layout** ». **Le nom du fichier correspond à son identifiant de ressources.** Tous les éléments d'interface, layout ou widgets héritent de la classe « **ViewGroup** », héritant elle-même de la classe « **View** ».

Ressources – layout

```
activity_main.xml
```

```
1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:paddingBottom="@dimen/activity_vertical_margin"
6     android:paddingLeft="@dimen/activity_horizontal_margin"
7     android:paddingRight="@dimen/activity_horizontal_margin"
8     android:paddingTop="@dimen/activity_vertical_margin"
9     tools:context="com.example.myapplication.MainActivity" >
10
11     <TextView
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:text="@string/hello_world" />
15
16 </RelativeLayout>
```

Ressources – layout

La déclaration de la vue dans une classe nécessite que cette dernière hérite de la classe « Activity ». Il est ainsi possible de lier l'activité à la vue au sein de la méthode surchargée « onCreate » (au minimum) avec la méthode « setContentView » qui permet de déserialiser le layout. La méthode prenant en paramètre l'identifiant de ce dernier.

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

Si l'activité s'est arrêtée de manière inattendue, la méthode « onCreate » reçoit un objet Bundle contenant l'état de l'interface utilisateur sauvegardée lors du dernier appel à la méthode « onSavedInstanceState ».

Chapitre N° 2

Interface et réception d'évènements

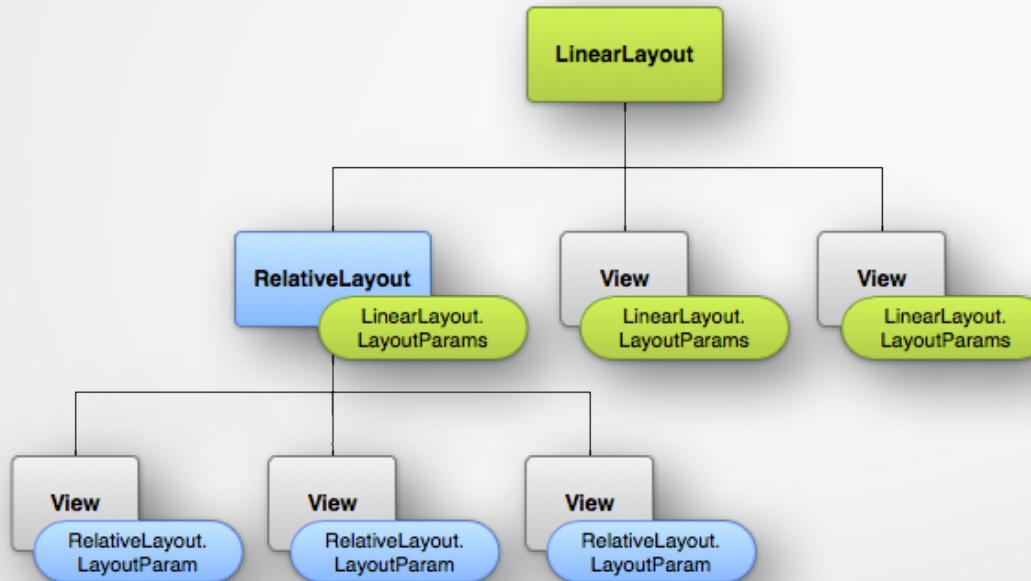
2.1 – Layouts

Interfaces / Vues / Fragments

- . Vues : Tous les éléments visuels des interfaces sont considérés comme des vues. Il en va ainsi de même pour les contrôles et les layouts qui dérivent de la classe vue.
- . Groupes de vues : Les groupes de vues sont des extensions de la classe View pouvant contenir plusieurs vues filles. L'extension de la classe ViewGroup permet de créer des contrôles composites constitués de vues filles interconnectées.
- . Fragments : Les fragments ont été introduits à partir de la version Android 3.0, soit l'API 11. Ces derniers permettent d'encapsuler certaines parties d'interface afin de créer des composants réutilisables par exemple.

Layout

Les layouts héritent de la classe ViewGroup, elle-même héritant de la classe View. Chaque ViewGroup peut donc contenir lui-même d'autre ViewGroup. Ces layouts sont ainsi utilisés pour disposer des vues filles dans l'interface utilisateur.



Le SDK Android contient déjà quelques classes layouts. Il est donc possible de les utiliser, modifier et même de créer vos propres layouts. Pour l'implémentation des layouts on utilise les ressources externes en XML. Chaque layout ne contient qu'un seul élément racine, et ce dernier peut contenir lui autant de layouts et de vues imbriqués que nécessaire.

Layout

```
activity_main.xml
```

```
1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:paddingBottom="@dimen/activity_vertical_margin"
6     android:paddingLeft="@dimen/activity_horizontal_margin"
7     android:paddingRight="@dimen/activity_horizontal_margin"
8     android:paddingTop="@dimen/activity_vertical_margin"
9     tools:context="com.example.todoit.MainActivity" >
10
11     <TextView
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:text="@string/hello_world" />
15
16 </RelativeLayout>
```

Layout – attributs communs

Chaque layout et vue se voit appliquer divers attributs en fonction de leur rôle ou leur utilité. Certains attributs sont communs à la majorité des layouts et vus comme la largeur et la hauteur.

```
android:layout_width="match_parent"  
android:layout_height="match_parent"
```

wrap_content

Taille minimum
pour pouvoir
afficher le contenu

match_parent /
fill_parent (deprecated)

Remplit l'espace
disponible dans la
vue parente

Fixe (pixels)

Taille précisée par
le développeur

Layout – Implémentation

Pour affecter une interface à une activité, il est nécessaire d'appeler la méthode `setContentView` au sein de la méthode `onCreate`. La méthode `setContentView` prend en paramètre soit un identifiant de ressource layout, soit une instance de `View`.

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

Pour avoir la référence d'une vue qui se trouve dans un layout, on utilise la méthode `findViewById`, qui prend également en paramètre un identifiant de ressource.

```
TextView tv = (TextView) findViewById(R.id.title);
```

Layout – Implémentation

Comme vu précédemment, les layouts permettent de découpler la couche de présentation du code de la vue ou de l'activité. Il est toutefois possible d'implémenter les layouts directement dans le code. Il est alors nécessaire d'appliquer LayoutParameters avec la méthode setLayoutParams ou avec addView.

```
LinearLayout monLayout = new LinearLayout(this);
monLayout.setOrientation(LinearLayout.VERTICAL);

TextView tv = new TextView(this);
Button bouton = new Button(this);

tv.setText("Ne cliquez pas !");
bouton.setText("1 clic = 1 euro ");

int height = LinearLayout.LayoutParams.MATCH_PARENT;
int width = LinearLayout.LayoutParams.MATCH_PARENT;

monLayout.addView(tv, new LinearLayout.LayoutParams(height, width));
monLayout.addView(bouton, new LinearLayout.LayoutParams(height, width));
```

Linear Layout

Le Linear Layout permet d'aligner chaque vue fille verticalement ou horizontalement. Le layout horizontal met en œuvre une colonne de vues et le layout vertical lui met en œuvre une ligne de vues. La modification de l'attribut « weight » de ce layout permet par ailleurs de préciser un poids pour chaque vue fille, permettant ainsi de gérer la taille de ces dernières. Plus le poids est important, plus il peut s'étendre et occuper de place.

Ce layout est simple et permet surtout de construire des éléments qui seront imbriqués dans d'autres layouts.

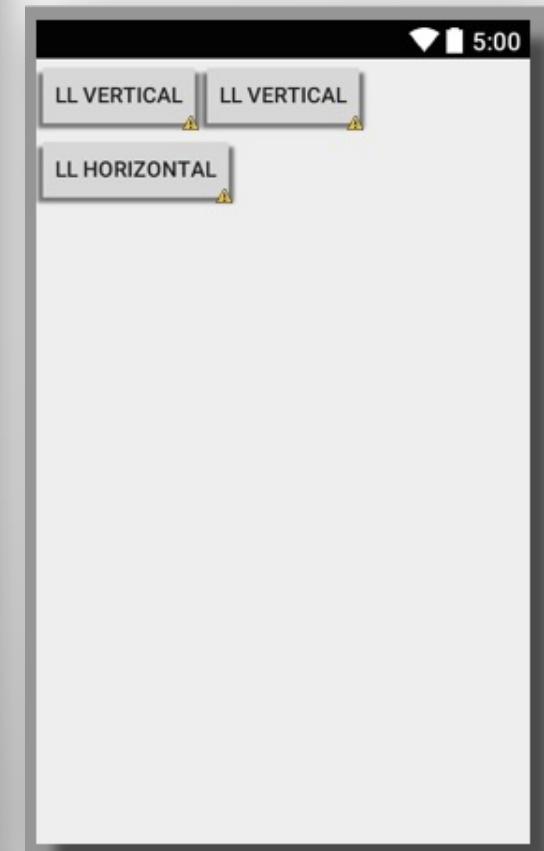


The screenshot shows a code editor window titled "linear_layout.xml". The code is an XML snippet defining a Linear Layout with the following structure:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:orientation="vertical" >
6
7 </LinearLayout>
```

Linear Layout

```
linear_layout.xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:orientation="vertical" >
6
7     <LinearLayout
8         android:layout_width="match_parent"
9         android:layout_height="wrap_content" >
10        <Button
11            android:id="@+id/button1"
12            android:layout_width="wrap_content"
13            android:layout_height="wrap_content"
14            android:text="LL Vertical" />
15        <Button
16            android:id="@+id/button2"
17            android:layout_width="wrap_content"
18            android:layout_height="wrap_content"
19            android:text="LL Vertical" />
20    </LinearLayout>
21
22    <Button
23        android:id="@+id/button3"
24        android:layout_width="wrap_content"
25        android:layout_height="wrap_content"
26        android:text="LL Horizontal" />
27
28 </LinearLayout>
```



Grid Layout

Le Grid Layout crée une grille rectangulaire permettant de disposer vos vues selon une suite de lignes et de colonnes. Bien qu'ils soient souples et permettent la mise en œuvre d'interface complexe, il est souhaitable d'utiliser l'éditeur de layout pour le mettre en œuvre (*API 14*).

L'attribut « `layout_gravity` » permet de préciser dans quelle direction les éléments doivent s'étendre si nécessaire.

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:columnCount="1"  
    android:orientation="vertical" >  
  
    </GridLayout>
```

GridLayout

```
grid_layout.xml ✎
1 <GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="fill_parent"
3     android:layout_height="fill_parent"
4     android:columnCount="1"
5     android:orientation="vertical" >
6
7     <TextView
8         android:id="@+id/conditionText"
9         android:textAppearance="?android:attr/textAppearanceMedium"
10        android:layout_gravity="fill"
11        android:text="@string/conditions"/>
12
13    <LinearLayout
14        android:layout_gravity="fill_horizontal"
15        android:orientation="horizontal"
16        android:padding="10dp" >
17        <!-- On a besoin ici que de 10dp ! -->
18
19        <Button
20            android:layout_width="match_parent"
21            android:layout_height="wrap_content"
22            android:layout_weight="1"
23            android:text="Accepter" />
24
25        <Button
26            android:layout_width="match_parent"
27            android:layout_height="wrap_content"
28            android:layout_weight="2.5"
29            android:text="Refuser" />
30    </LinearLayout>
31
32 </GridLayout>
```

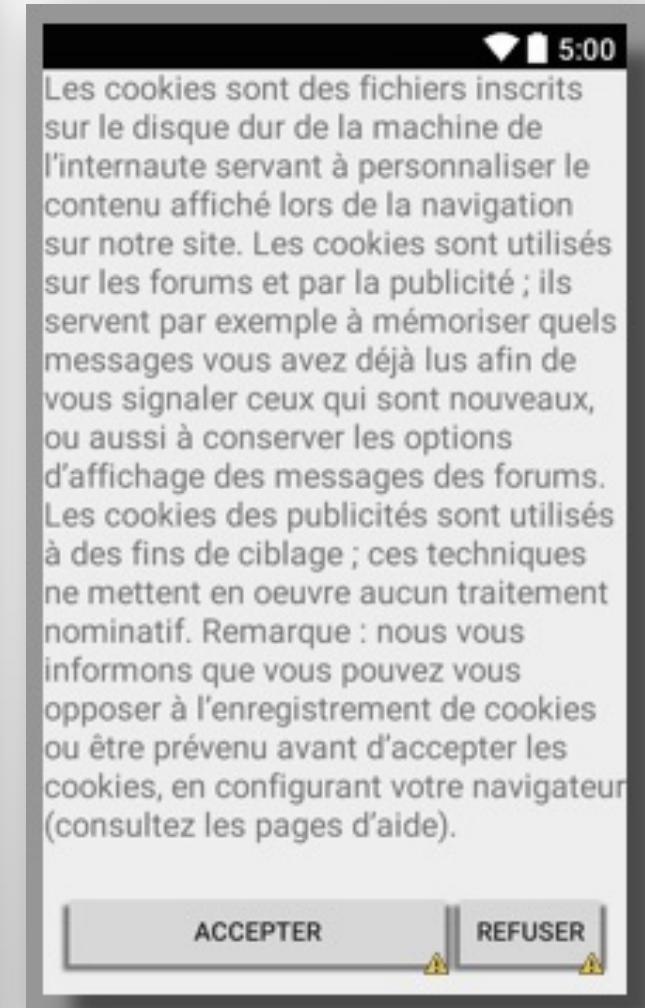


Table Layout

La Table Layout permet de positionner les vues filles en ligne et en colonne, les cellules ne peuvent pas être étendues, mais peuvent être laissées vides.

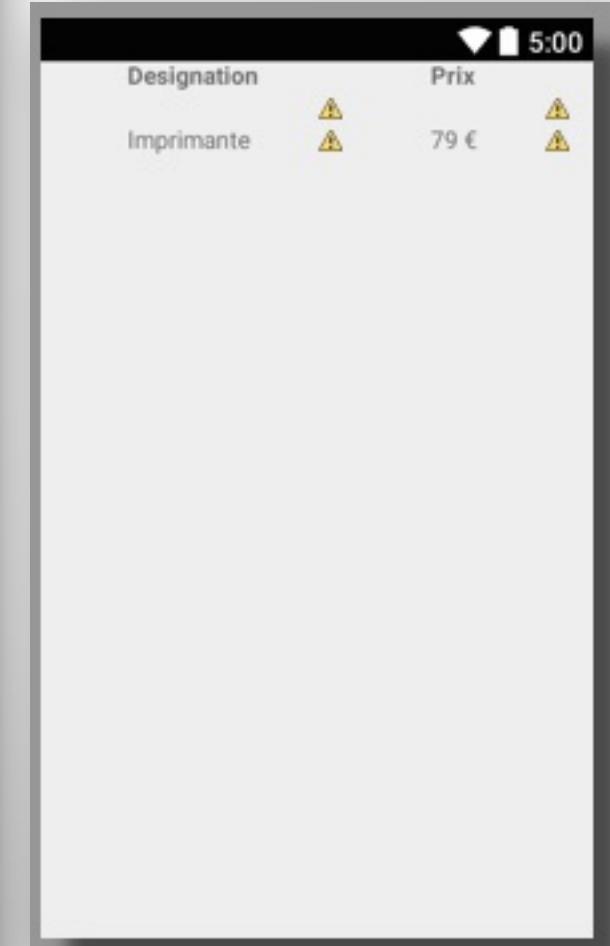
La création de ligne se fait grâce à l'élément TableRow avec 0 ou plusieurs colonnes.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" >  
  
</TableLayout>
```

Table Layout

*grid_layout.xml

```
1 <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="match_parent" >
4     <TableRow>
5         <!-- Line1/Col1 -->
6         <TextView
7             android:paddingLeft="50dp"
8             android:paddingRight="50dp"
9             android:paddingBottom="20dp"
10            android:textStyle="bold"
11            android:text="Designation"/>
12        <!-- Line1/Col2 -->
13        <TextView
14            android:paddingLeft="50dp"
15            android:paddingRight="50dp"
16            android:paddingBottom="20dp"
17            android:textStyle="bold"
18            android:text="Prix"/>
19    </TableRow>
20
21    <TableRow>
22        <!-- Line2/Col1 -->
23        <TextView
24            android:paddingLeft="50dp"
25            android:paddingRight="50dp"
26            android:text="Imprimante"/>
27        <!-- Line2/Col2 -->
28        <TextView
29            android:paddingLeft="50dp"
30            android:paddingRight="50dp"
31            android:text="79 €."/>
32    </TableRow>
33 </TableLayout>
```



Relative Layout

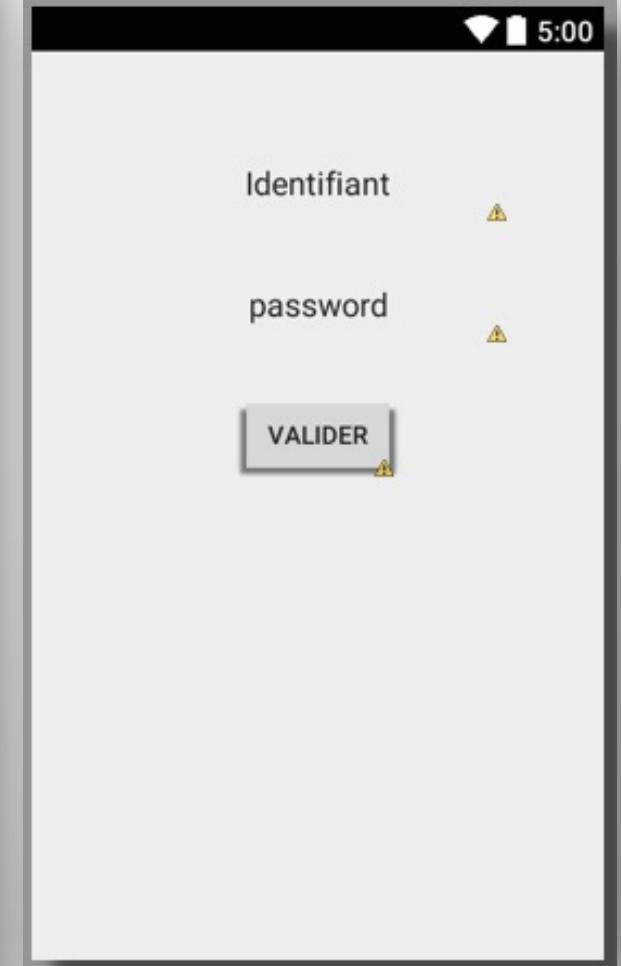
Le Relative Layout est un des layouts les plus souples. Il permet de positionner les différentes vues les unes par rapport aux autres, ou à son parent. On utilise alors différents attributs de positionnement pour préciser son placement.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

</RelativeLayout>
```

Relative Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <EditText
        android:id="@+id/editTextIdentifiant"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="60dp"
        android:ems="10"
        android:gravity="center_vertical|center_horizontal"
        android:text="Identifiant"/>
    <EditText
        android:id="@+id/passText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editTextIdentifiant"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="30dp"
        android:ems="10"
        android:gravity="center_vertical|center_horizontal"
        android:text="password" />
    <Button
        android:id="@+id/validerBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/passText"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="25dp"
        android:text="Valider" />
</RelativeLayout>
```



Relative Layout - Positionnement

Positionnement par rapport au conteneur :

android:layout_alignParentTop : alignement avec le haut du conteneur

android:layout_alignParentBottom : alignement avec le bas du conteneur

android:layout_alignParentLeft : alignement avec la gauche du conteneur

android:layout_alignParentRight : alignement avec la droite du conteneur

Positionnement par rapport à un élément (désigné par son id) :

android:layout_above : au-dessus de l'élément

android:layout_below : en dessous de l'élément

android:layout_toLeftOf : à gauche de l'élément

android:layout_toRightOf : à droite de l'élément

android:layout_alignTop : le haut de l'élément en cours aligné avec l'élément désigné

android:layout_alignLeftOf : la gauche de l'élément en cours aligné avec l'élément désigné

android:layout_alignRightOf : la droite de l'élément en cours aligné avec l'élément désigné

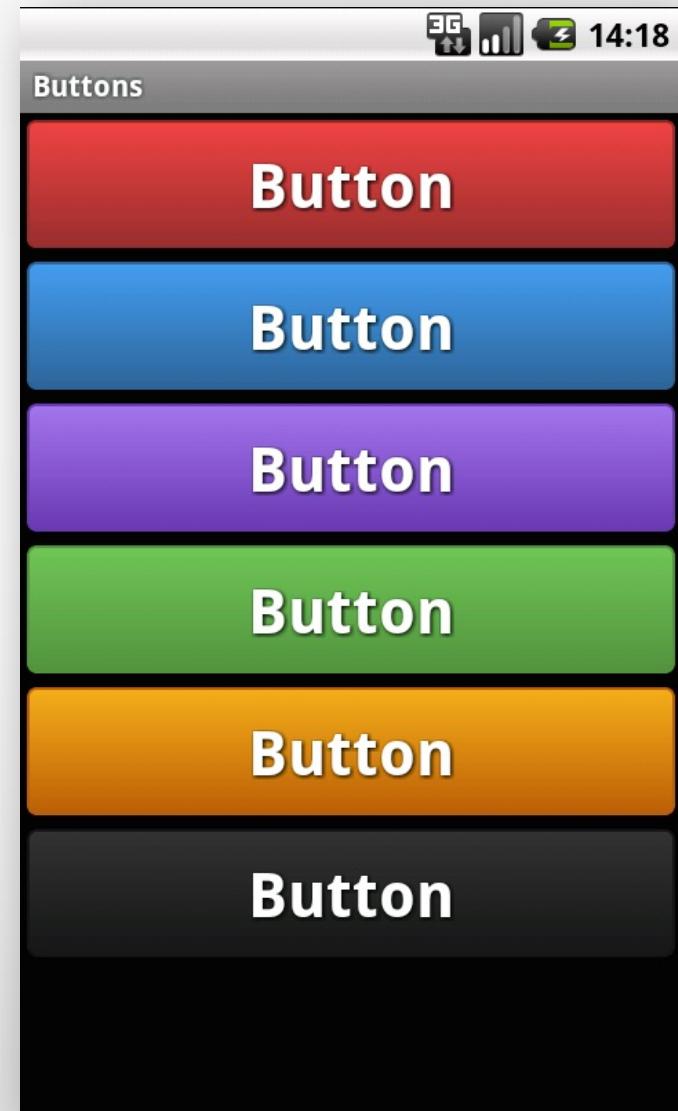
Autre

android:layout_centerHorizontal : centrer l'élément horizontalement

android:layout_centerVertical : centrer l'élément verticalement

android:layout_centerInParent : centrer un élément dans le conteneur parent

2.2 – Vues / Widget



Vues / Widgets

Un certain nombre de vues sont déjà intégrées dans le framework Android, permettant de créer des interfaces particulièrement riches. Quelques une de ces vues sont présentées ci-dessous, pour les autres consultés la documentation officielle :

TextView

- Libellé de texte, pas de modification par l'utilisateur

EditText

- Chance de saisie modifiable par l'utilisateur

ListView

- Groupe de vues permettant de créer une liste verticale sous forme de ligne

Spinner

- Composite de ListView et TextView permettant de créer un bouton affichant une liste déroulante

Button

- Bouton standard

ImageButton

- Bouton donc le design peut être préciser avec une image

CheckBox

- Case à cocher (Bouton à deux états)

RadioButton

- Bouton à deux état utilisé en groupe. L'utilisateur peut faire un choix unique parmi plusieurs .

Vues – Design et interactions

Tout comme les layouts, les vues disposent d'attributs permettant de préciser leur design, mais aussi la façon dont ils interagissent et le champ de restriction que l'on peut appliquer aux utilisateurs. Par exemple dans le cas d'un champ de texte éditable : (<https://developer.android.com/guide/topics/ui/controls/text.html>)

Attribut de design

android:layout_height : hauteur du champ

android:layout_width : largeur du champ

Attribut d'interaction

android:inputType : type de texte attendu

text

Normal

textUri

url web

textEmailAddress

adresse mal

textPassword

Mot de passe

number, phone,
date..

Clavier
numérique

Vues – EditText

Le type de texte attendu étant une adresse mail, lorsque l'utilisateur va cliquer pour saisir une valeur, le clavier de saisie d'adresse email s'affichera alors. Si le texte attendu était un numéro de téléphone, le clavier affiché aurait alors été celui du composeur de numéros comportant uniquement des chiffres.

```
<EditText  
    android:id="@+id/email_address"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/email_hint"  
    android:inputType="textEmailAddress" />
```



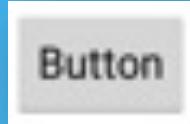
Vues / Widgets

Pour chacune des vues, un certain nombre d'attributs permettent de les préciser selon les besoins. D'une manière générale, on retrouve :

- La taille de la vue
- Son positionnement relatif ou non par rapport à son conteneur
- Le texte affiché si cela est possible ou le chemin vers la ressource

```
<Button  
    android:id="@+id/validerBtn"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Valider" />
```

Button



```
<Switch  
    android:id="@+id/switch1"  
    android:text="Visibilité du flux"  
    android:textOn="On"  
    android:textOff="Off"/>
```

Switch (4.0) /
ToggleButton



```
<ImageView  
    android:id="@+id/imageView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/ic_launcher" />
```

Image



2.3 – Listeners



Vues - Interactions

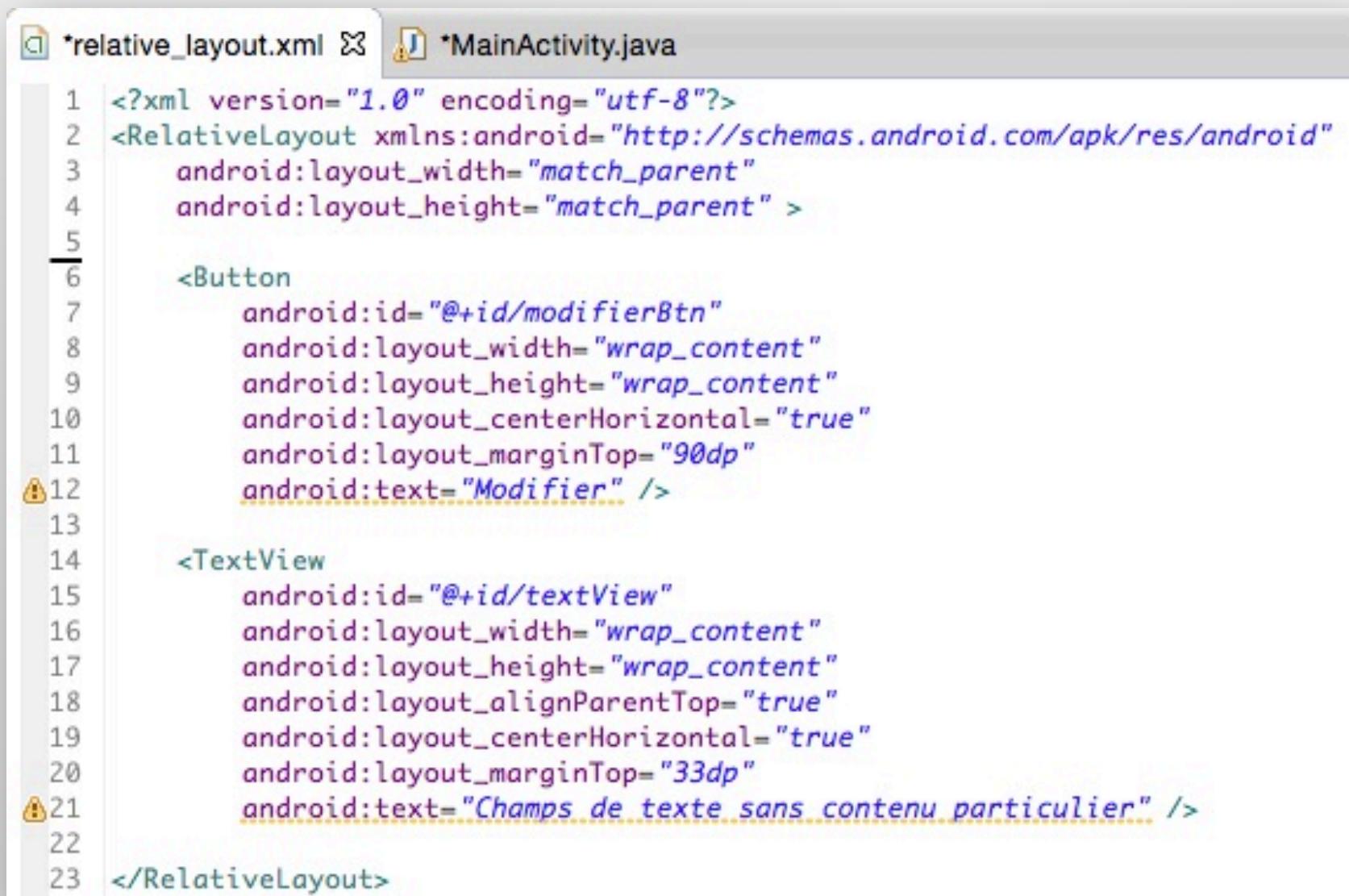
Pour toutes les vues utilisables par l'utilisateur, bouton, champs de texte éditables, spinner, switch, etc.., il est nécessaire de prendre en compte les actions effectuées par ce dernier. Il faut donc implémenter des listeners sur ces vues.

Par exemple si l'utilisateur effectue un clic sur un bouton, il y a deux façons de mettre en œuvre les listener afin de récupérer le l'évènement.

1 – Ajout d'un listener sur chaque vue séparément

```
Button btn = (Button) findViewById(R.id.modifierBtn);
btn.setOnClickListener(new OnClickListener(){
    @Override
    public void onClick(View v) {
    }
});
```

Vues – Listener onClick (1)



The screenshot shows an Android Studio interface with two tabs: "relative_layout.xml" and "MainActivity.java". The "relative_layout.xml" tab is active, displaying the following XML code:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent" >
5
6     <Button
7         android:id="@+id/modifierBtn"
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:layout_centerHorizontal="true"
11        android:layout_marginTop="90dp"
12        android:text="Modifier" />
13
14     <TextView
15         android:id="@+id/textView"
16         android:layout_width="wrap_content"
17         android:layout_height="wrap_content"
18         android:layout_alignParentTop="true"
19         android:layout_centerHorizontal="true"
20         android:layout_marginTop="33dp"
21         android:text="Champs de texte sans contenu particulier" />
22
23 </RelativeLayout>
```

The code defines a RelativeLayout containing a Button and a TextView. Both views have yellow warning icons next to them, indicating potential issues or deprecated usage.

Vues – Listener onClick (1)

The screenshot shows the Android Studio interface with two tabs open: `*relative_layout.xml` and `*MainActivity.java`. The `MainActivity.java` tab is active, displaying Java code for an Activity. The code implements the `ActionBarActivity` and overrides the `onCreate` method to set the content view and add a click listener to a button. The `onClick` method updates a text view.

```
1 package com.example.todoit;
2
3+import android.support.v7.app.ActionBarActivity;..
4
5 public class MainActivity extends ActionBarActivity {
6
7     @Override
8     protected void onCreate(Bundle savedInstanceState) {
9         super.onCreate(savedInstanceState);
10        setContentView(R.layout.activity_main);
11
12
13        Button btn = (Button) findViewById(R.id.modifierBtn);
14        btn.setOnClickListener(new OnClickListener(){
15
16            @Override
17            public void onClick(View v) {
18                TextView tv = (TextView) findViewById(R.id.textView);
19                tv.setText("Modification du texte");
20            }
21        });
22    }
23
24}
25
26
27
28
29
30 }
```

Vues – Listener onClick (2)

2 – Implémentation de l'interface OnClickListener

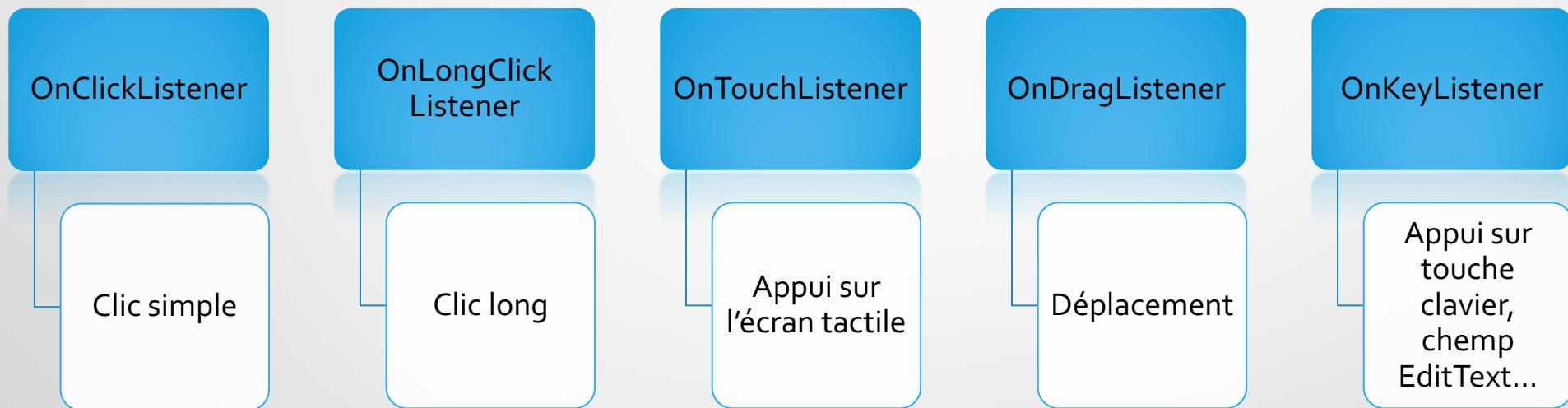
```
relative_layout.xml MainActivity.java
```

```
1 package com.example.todoit;
2
3+import android.support.v7.app.ActionBarActivity;[]
12
13 public class MainActivity extends ActionBarActivity implements OnClickListener {
14
15@Override
16protected void onCreate(Bundle savedInstanceState) {
17    super.onCreate(savedInstanceState);
18    setContentView(R.layout.activity_main);
19
20    Button btn = (Button) findViewById(R.id.modifierBtn);
21    btn.setOnClickListener(this);
22}
23
24@Override
25public void onClick(View v) {
26    TextView tv = (TextView) findViewById(R.id.textView);
27    tv.setText("Texte modifié !");
28}
```

Nécessite de vérifier l'ID du bouton s'il y en a plusieurs, dans la méthode onClick

Vues – Listeners

Les évènements provenant de l'interaction utilisateur sont ainsi captés par plusieurs types de listeners, en voici une liste non exhaustive.



<https://developer.android.com/reference/android/view/package-summary.html>

2.4 – ListView & Adaptateurs

A - ListView

Il existe une vue qui permet d'afficher des items les uns à la suite des autres, à savoir une ListView. Ces items, autrement dit les éléments qui composent ces listes peuvent contenir une à trois lignes. Il y a plusieurs façons d'implémenter ces ListView. La première façon consiste à hériter de la classe ListActivity, qui est la façon la plus simple, mais qui a aussi ses limites. L'autre façon consiste à customiser les listes grâce à un adaptateur.

Extends de la classe ListActivity

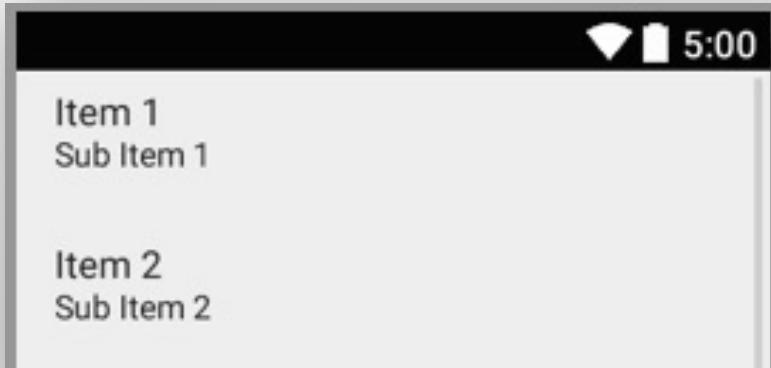


La vue ListView
doit avoir l'ID :
@android:id/list

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:orientation="vertical" >  
  
    <ListView  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:id="@+id/list" />  
  
</LinearLayout>
```

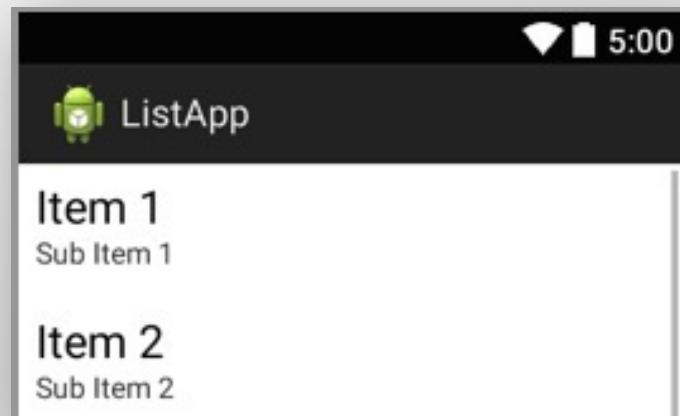
A - ListView

```
public class MainActivity extends ListActivity {  
  
    private String[] cityList = {"Vannes", "Lorient", "Rennes", "Quimper"};  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        ArrayAdapter adapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1, cityList);  
        setListAdapter(adapter);  
  
    }  
  
}
```



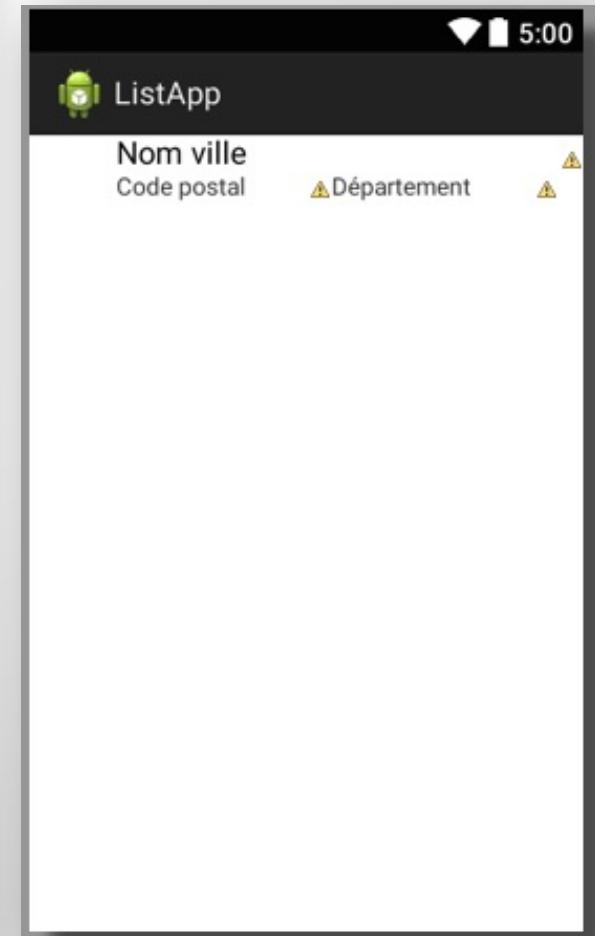
B - ListView – Adapter – Layout

```
city_list_item.xml activity_city_list.xml
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="fill_parent"
3     android:layout_height="fill_parent"
4     android:orientation="vertical" >
5
6     <ListView
7         android:layout_width="match_parent"
8         android:layout_height="wrap_content"
9         android:id="@+id/cityList"/>
10
11 </LinearLayout>
```



B - ListView – Adapter – Layout Item

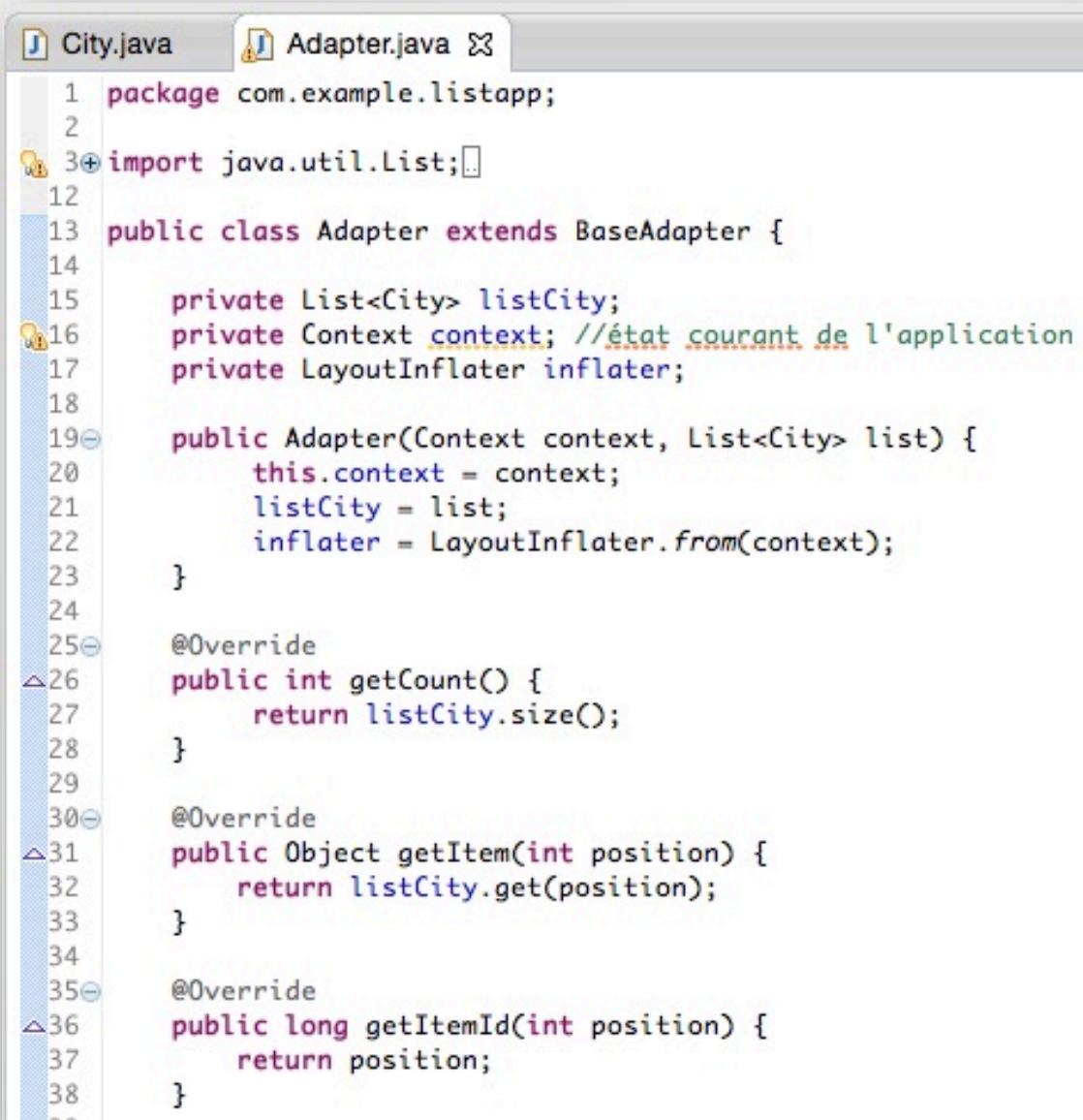
```
city_list_item.xml activity_city_list.xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:orientation="vertical" >
6     <TextView
7         android:id="@+id/cityNameText"
8         android:layout_width="match_parent"
9         android:layout_height="wrap_content"
10        android:text="Nom ville"
11        android:paddingLeft="50dp"
12        android:textAppearance="?android:attr/textAppearanceMedium" />
13     <LinearLayout
14         android:layout_width="wrap_content"
15         android:layout_height="match_parent"
16         android:orientation="horizontal">
17         <TextView
18             android:id="@+id/postalCodeText"
19             android:layout_width="match_parent"
20             android:layout_height="wrap_content"
21             android:text="Code postal"
22             android:paddingLeft="50dp"
23             android:paddingRight="50dp"
24             android:textAppearance="?android:attr/textAppearanceSmall"/>
25         <TextView
26             android:id="@+id/cityDepartmentText"
27             android:layout_width="match_parent"
28             android:layout_height="wrap_content"
29             android:text="Département"
30             android:paddingRight="50dp"
31             android:textAppearance="?android:attr/textAppearanceSmall"/>
32     </LinearLayout>
33 </LinearLayout>
```



B - ListView – Adapter – City

```
public class City {  
  
    private String cityName;  
    private String cityDepartment;  
    private int postalCode;  
  
    public City(String name, String department, int postalcode){  
        cityName = name;  
        cityDepartment = department;  
        postalCode = postalcode;  
    }  
  
    public String getCityName() {  
        return cityName;  
    }  
  
    public void setCityName(String cityName) {  
        this.cityName = cityName;  
    }  
  
    public String getCityDepartment() {  
        return cityDepartment;  
    }  
  
    public void setCityDepartment(String cityDepartment) {  
        this.cityDepartment = cityDepartment;  
    }  
  
    public int getPostalCode() {  
        return postalCode;  
    }  
  
    public void setPostalCode(int postalCode) {  
        this.postalCode = postalCode;  
    }  
}
```

B - ListView – Adapter – Adapter (City)



The screenshot shows a Java code editor with two tabs: "City.java" and "Adapter.java". The "Adapter.java" tab is active, displaying the following code:

```
1 package com.example.listapp;
2
3+ import java.util.List;
4
5 public class Adapter extends BaseAdapter {
6
7     private List<City> listCity;
8     private Context context; //état courant de l'application
9     private LayoutInflator inflater;
10
11    public Adapter(Context context, List<City> list) {
12        this.context = context;
13        listCity = list;
14        inflater = LayoutInflater.from(context);
15    }
16
17    @Override
18    public int getCount() {
19        return listCity.size();
20    }
21
22    @Override
23    public Object getItem(int position) {
24        return listCity.get(position);
25    }
26
27    @Override
28    public long getItemId(int position) {
29        return position;
30    }
31
32}
```

The code defines a class named Adapter that extends BaseAdapter. It has three private fields: listCity, context, and inflater. The constructor takes a Context and a List<City>. The class overrides getCount(), getItem(), and getItemId() methods.

B - ListView – Adapter – Adapter (City)

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {

    View view;

    if (convertView == null) {
        //Initialisation de la vue city_list_item
        view = (View) inflater.inflate(R.layout.city_list_item, parent, false);
    } else {
        view = (View) convertView;
    }

    //Initialisation des vues du layout
    TextView nameCity = (TextView) view.findViewById(R.id.cityNameText);
    TextView departementCity = (TextView) view.findViewById(R.id.cityDepartmentText);
    TextView postalCodeCity = (TextView) view.findViewById(R.id.postalCodeText);

    //modification des vues
    nameCity.setText(listCity.get(position).getCityName());
    departementCity.setText(listCity.get(position).getCityDepartment());
    postalCodeCity.setText(Integer.toString(listCity.get(position).getPostalCode()));

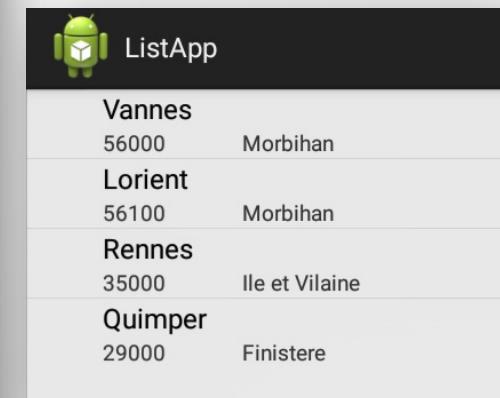
    //on retourne la vue créée
    return view;
}

}
```

B - ListView – Adapter – Activity

*CityListActivity.java

```
15
16 public class CityListActivity extends Activity {
17
18    @Override
19    protected void onCreate(Bundle savedInstanceState) {
20        super.onCreate(savedInstanceState);
21        setContentView(R.layout.activity_city_list);
22
23        //Création d'une arrayList de ville
24        ArrayList<City> cityListArray = new ArrayList<City>();
25        City vannes = new City("Vannes", "Morbihan", 56000);
26        City lorient = new City("Lorient", "Morbihan", 56100);
27        City rennes = new City("Rennes", "Ile et Vilaine", 35000);
28        City quimper = new City("Quimper", "Finistere", 29000);
29        cityListArray.add(vannes);
30        cityListArray.add(lorient);
31        cityListArray.add(rennes);
32        cityListArray.add(quimper);
33
34        //Initialisation de l'adapter pour city
35        Adapter adapter = new Adapter(this, cityListArray);
36        //Récupération de la ListView
37        ListView list = (ListView)findViewById(R.id.cityList);
38        //Passage des données à la ListView
39        list.setAdapter(adapter);
```



B - ListView – Adapter – Activity

```
//On ajoute un listener (clic sur un item)
list.setOnItemClickListener(new OnItemClickListener(){
    @Override
    public void onItemClick(AdapterView<?> adapter, View v, int position, long id) {

        City selectedItem = (City) adapter.getItemAtPosition(position);
        Log.v("Element : ", "Ville :" + selectedItem.getCityName()); //LOG
    }
});
```

2.5 – Toast

Toast

Les Toasts correspondent à des très courtes notifications visibles. Elles s'effacent au bout de quelques secondes et permettent de passer un message simple à l'utilisateur sans perturber le fonctionnement de l'activité (aucune interruption).

Pour mettre en œuvre ces petits messages, on appelle la méthode statique « `makeText` » de la classe `Toast`. Cette méthode prend en paramètre le contexte de l'application, le message à afficher, et la durée durant laquelle ce message doit rester affiché (`LENGTH_LONG` ou `LENGTH_SHORT`). Pour l'afficher, il est nécessaire d'appliquer la méthode « `show` » à ce dernier.

```
Toast.makeText(this, "Je suis un Toast !", Toast.LENGTH_LONG).show();
```



2.6 – ActionBar

ActionBar

Les Actions Bar (API 11 – Android 3.0) permettent d'intégrer des composants ou des vues au sein de la barre de titre. Cela permet par exemple d'ajouter certaines options sans surcharger les interfaces utilisateurs, comme en y intégrant une barre de recherche, ou encore en s'en servant comme système de navigation. Chaque activité peut disposer de son Action bar, toutefois il est également possible de la masquer.

Pour l'activer, il suffit de préciser une version minimum de SDK à 11 et de l'appeler au sein de la méthode OnCreate(), ou tout simplement d'hériter de la classe ActionBarActivity au lieu d'Activity, il n'y a alors qu'à ajouter les boutons, icônes (...) supplémentaires.

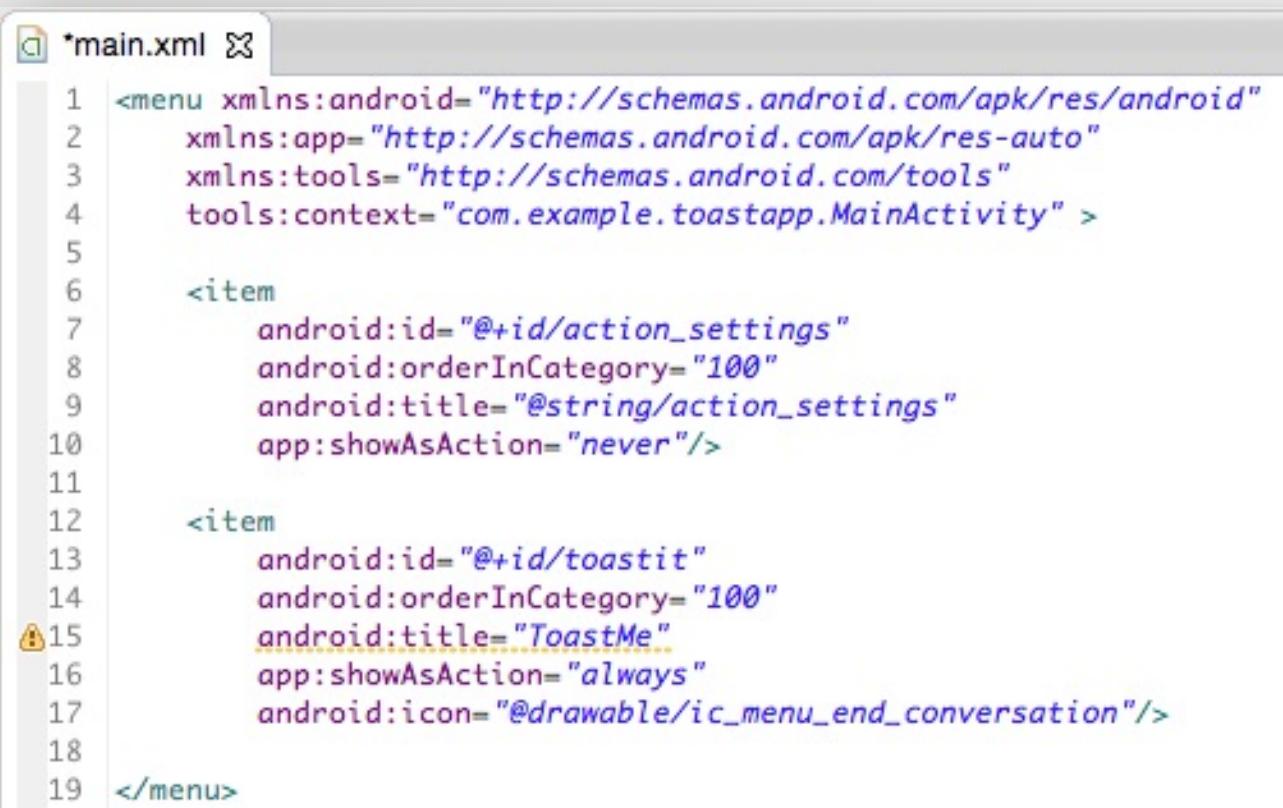
```
ActionBar actionBar = getSupportActionBar();
actionBar.show();
```

Différentes méthodes peuvent alors lui être appliquées. On pourra ainsi masquer le titre de l'application qui se trouve à côté du logo avec la méthode setDisplayShowTitleEnabled(false) qui prend un booléen en paramètre.

Documentation : <https://developer.android.com/guide/topics/ui/actionbar.html>

ActionBar

Ces ActionBar utilisent les fichiers de ressource qui se trouvent dans le dossier « menu ». Ces fichiers au format XML permettent de décrire les éléments qui vont composer l'ActionBar.



```
*main.xml
1 <menu xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:app="http://schemas.android.com/apk/res-auto"
3     xmlns:tools="http://schemas.android.com/tools"
4     tools:context="com.example.toastapp.MainActivity" >
5
6     <item
7         android:id="@+id/action_settings"
8         android:orderInCategory="100"
9         android:title="@string/action_settings"
10        app:showAsAction="never"/>
11
12     <item
13         android:id="@+id/toastit"
14         android:orderInCategory="100"
15         android:title="ToastMe"
16         app:showAsAction="always"
17         android:icon="@drawable/ic_menu_end_conversation"/>
18
19 </menu>
```

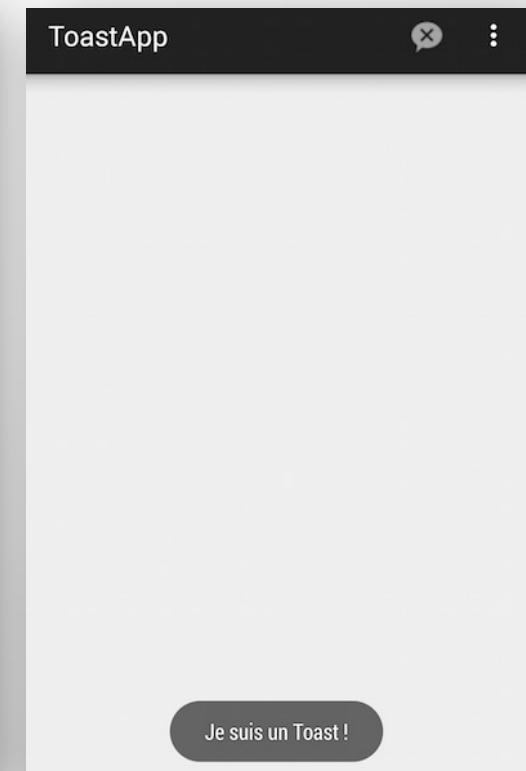
Différents attributs permettent de préciser les comportements :

- **Never** : Jamais dans la barre d'action
- **Always** : toujours présente
- **Withtext** : Affiche le texte de l'élément
- **ifRoom** : Affiche l'élément s'il y a de la place

ActionBar

Afin de pouvoir utiliser et préciser le comportement de ces menus, il faut surcharger deux méthodes au sein de notre activité. La méthode « `onCreateOptionsMenu` » permettant de prendre en compte l'ActionBar au sein de l'activité et « `onOptionsItemSelected` » permettant de définir l'action à mettre en œuvre lors d'un clic sur un des boutons, grâce à leurs identifiants (**qui sont des entiers**).

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    // Inflate the menu; this adds items to the action bar if it is present.  
    getMenuInflater().inflate(R.menu.main, menu);  
    return true;  
}  
  
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    // Handle action bar item clicks here. The action bar will  
    // automatically handle clicks on the Home/Up button, so long  
    // as you specify a parent activity in AndroidManifest.xml.  
    int id = item.getItemId();  
    if (id == R.id.action_settings) {  
        return true;  
    }  
    if (id == R.id.toastit) {  
        Toast.makeText(this, "Je suis un Toast !", Toast.LENGTH_LONG).show();  
    }  
  
    return super.onOptionsItemSelected(item);  
}
```



Plan – Chapitre 3

- 3. Intention & récepteur de diffusion
 - 3.1 Intentions
 - 3.1.1 Lancement d'activité
 - 3.1.2 Lancement d'activité explicite
 - 3.1.3 Lancement d'activité implicite
 - 3.1.4 Filtre d'intention
 - 3.1.5 Passage de valeurs
 - 3.1.6 Récupération de valeur
 - Pause
 - 3.2 Récepteur de diffusion
 - 3.2.1 Broadcast receiver..
 - 3.2.2 Enregistrement d'un récepteur de diffusion
 - 3.2.3 Création de récepteurs de diffusion

Chapitre N° 3

Intentions, récepteurs de diffusion

3.1 – Intentions

Intents

Les Intents sont des dispositifs d'envois de messages asynchrones entre les applications ou les composants, comme les activités, les services... Ils peuvent être utilisés pour envoyer des informations ou demander l'accès à certaines fonctionnalités.

D'une manière générale on peut les utiliser pour :

- Démarrer explicitement une activité ou un service en utilisant le nom de classe
- Lancer une activité ou un service dans le but d'effectuer une action concernant un ensemble de données
- Annoncer un événement survenu.

Ces objets sont du type « android.content.Intent ». Ces derniers sont couramment utilisés pour démarrer de nouvelle activité explicitement(en ciblant une classe à charger) ou implicitement (initié une action concernant un ensemble de données).

On peut également les utiliser pour diffuser des messages au système entier. Ainsi, les applications pourront écouter et réagir à ces messages via un récepteur de diffusion.

Lancer une activité

On peut ainsi utiliser les intents pour lancer une activité, ou un service. On utilise la **méthode startActivity()** qui prend un Intent en paramètre.

```
Intent myIntent = new Intent(this, SecondActivity.class);
startActivity(myIntent);
```

La méthode startActivity() détermine et démarre l'activité correspondant le mieux à l'intention « myIntent ».

La seconde Activité, ici nommée « SecondActivity » est quant à elle caractérisée de **sous-activité** pour la différencier de la première activité appelante.

Lancement explicite d'activité

D'une certaine manière, pour lier les écrans entre eux il est nécessaire de préciser explicitement quelle activité il faut ouvrir.

Pour démarrer explicitement une activité il faut initialiser un Intent et le passer à la méthode « `startActivity()` » comme vu précédemment. L'objet Intent lui est à initialiser en lui passant un context et l'activité cible à lancer.

```
Intent myIntent = new Intent(this, SecondActivity.class);
startActivity(myIntent);
```

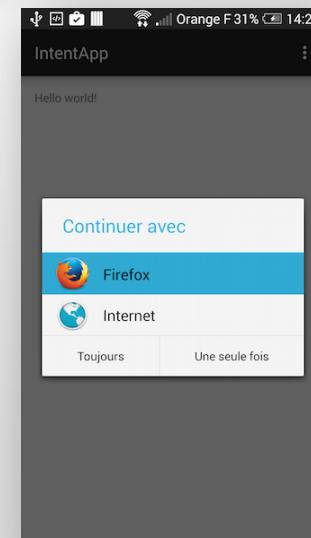
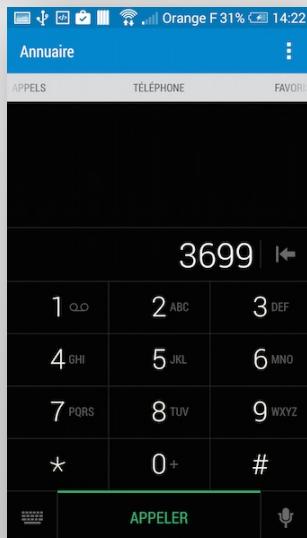
Une fois cet appel effectué, la nouvelle Activité, `SecondActivity`, est donc créée, lancée et mise dans la Back Stack, au sommet. Pour revenir à l'activité appelante, il faut donc soit invoquer la méthode « `finish` » de la seconde activité, ou appuyer sur le bouton retour de la plateforme. **L'appui sur ce bouton ou l'appel à la méthode `finish` supprimera les sous-activités en cours.** Bien entendu, il est possible d'initialiser autant d'activité que l'on veut de cette manière et dans l'activité que l'on souhaite.

Lancement implicite d'activité

Il y a une autre manière de lancer des actions, à savoir le lancement implicite. Ces dispositifs sont utilisés pour permettre aux composants d'applications anonymes de répondre à des demandes d'actions. Globalement, grâce à ce mécanisme il est possible de demander au système de lancer une activité ayant la possibilité d'effectuer une action sans savoir au préalable de quelles application ou activité il s'agit.

```
Intent telIntent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:36-99"));
startActivity(telIntent);

Intent webIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.developer.android.com"));
startActivity(webIntent);
```



Lancement implicite d'activité

Pour mettre en œuvre ce type d'intention, l'objet Intent prend en paramètre :

- L'action à effectuer
- L'URI des données sur lesquelles effectuer l'action (optionnel)

Si plusieurs activités ou applications peuvent répondre à l'action demandée, le système proposera alors une liste des applications pouvant l'exécuter à l'utilisateur comme le montre la diapositive précédente. Sinon, si une seule activité, application ou composant est disponible pour effectuer cette action, alors le système la démarrera automatiquement.

(<https://developer.android.com/reference/android/content/Intent.html>)

Il est possible qu'aucune application ne pouvant exécuter l'action ne soit disponible sur la plateforme. Pour pallier à cela, il est possible de tester si l'intent aura une réponse favorable en activité pouvant exécuter son action en appelant la méthode « resolveActivity » qui prend en paramètre un gestionnaire de package (package manager).

Pour les plus curieux :

<https://developer.android.com/reference/android/content/pm/PackageManager.html#resolveActivity%28android.content.Intent,%20int%29>

Lancement implicite - Exemples

ACTION_VIEW *content://contacts/people/1* – affiche des informations concernant le contact ayant comme identifiant '1'

ACTION_DIAL *content://contacts/people/1* – affiche le dialer avec le numéro de la personne ayant comme identifiant '1'

ACTION_DIAL *tel:123* – affiche le dialer avec le numéro prêt à être appelé '123'

ACTION_EDIT *content://contacts/people/1* – affiche l'édition d'information concernant la personne ayant l'identifiant '1'

```
Intent telIntent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:36-99"));
startActivity(telIntent);

Intent webIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.developer.android.com"));
startActivity(webIntent);
```

Filtres d'intentions

Tout comme il est possible de lancer des activités implicitement, il est possible de mettre en œuvre les récepteurs et les filtres qui permettront à votre application de répondre aux intention diffusées. Un filtre d'intention permettra ainsi à votre application répondre au besoin de visualiser un lien web, lire une vidéo, etc..

Ainsi les filtres d'intention vont permettre de déclarer qu'un composant particulier est capable d'effectuer l'action demandée sur un type de données. L'enregistrement d'un composant comme gestionnaire d'intention se fait par **l'ajout de la balise « intent-filter » dans le manifeste, au sein du nœud du composant (activité ou service)**.

Si notre composant peut gérer deux types différents d'intent, il y aura alors deux filtres.

Filtres d'intentions

```
<activity
    android:name="com.example.batterychecker.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

Trois balises sont à prendre en compte au sein de la balise « intent-filter », à savoir:

- **La balise action** précise le nom de l'action à laquelle répondre.
- **La balise category** précise dans quelle circonstance une réponse doit être donnée à l'action. Il peut y en avoir plusieurs, personnalisée ou natives (ALTERNATIVE, SELECTED_ALTERNATIVE, BROWSABLE, DEFAULT, HOME, LAUNCHER).
- **La balise data** permet elle de préciser le type de données sur lesquelles votre composant peut agir (host, mimetype, path, port, scheme)..

Documentation Intent :

<https://developer.android.com/reference/android/content/Intent.html>

Intent - Passage de valeur (1)

Grâce aux Intent, il est possible de transférer des données entre les différentes activités avec ce que l'on appelle les « extras ». Ce qui est appelé extra est en fait un couple de clés/valeur. Cette clé étant toujours une chaîne de caractère. La valeur de la clé quant à elle peut être de plusieurs types (primitifs, String, Parcelable). Ils sont stockés dans l'intention en tant qu'objet Bundle.

Pour attacher ces extras au sein de l'Intent on utilisera ainsi la méthode « putExtra » surchargée.



The screenshot shows a Java code editor with two tabs: 'MainActivity.java' and 'SecondActivity.java'. The 'MainActivity.java' tab is active, displaying the following code:

```
13    @Override
14    protected void onCreate(Bundle savedInstanceState) {
15        super.onCreate(savedInstanceState);
16        setContentView(R.layout.activity_main);
17
18        Intent myIntent = new Intent(this, SecondActivity.class);
19        myIntent.putExtra("key", "mon message");
20        startActivity(myIntent);
21
22    }
```

Intent - Passage de valeur (1)



The screenshot shows an IDE interface with two tabs: "MainActivity.java" and "SecondActivity.java". The "MainActivity.java" tab is active, displaying the following Java code:

```
14    @Override
15    protected void onCreate(Bundle savedInstanceState) {
16        super.onCreate(savedInstanceState);
17        setContentView(R.layout.activity_second);
18
19        Bundle bundle = getIntent().getExtras();
20
21        if(bundle.containsKey("key")){
22            String message = bundle.getString("key");
23            Toast.makeText(this, message, Toast.LENGTH_LONG).show();
24        }
25        else{
26            Log.v("Message : ", "Message vide");
27        }
28    }
```

Intent - Passage de valeur (2)

Tout comme dans le cas d'un lancement d'activité explicite, il est possible de passer des données en faisant un appel implicite. L'exemple ci-dessous montre la préparation et l'envoi un mail à partir d'une activité en intégrant des données à l'Intent.

```
String to = "email@destinataire.fr";
String subject = "Sujet du mail";
String message = "Bonjour, je suis un message !";

Intent email = new Intent(Intent.ACTION_SEND);
email.putExtra(Intent.EXTRA_EMAIL, new String[]{to});
email.putExtra(Intent.EXTRA_SUBJECT, subject);
email.putExtra(Intent.EXTRA_TEXT, message);

email.setType("message/rfc822");

startActivity(email);

//startActivity(Intent.createChooser(email, "Choisir un client"));
```

Récupération de résultat

Toutes les activités enregistrées dans le manifeste peuvent être lancées en tant que sous-activités. Une activité qui a été démarrée avec la méthode `startActivity` est totalement indépendante de l'activité parente. Toutefois, si besoin, il est possible d'attendre un retour d'une sous-activité en utilisant la méthode « `startActivityForResult()` »

Une fois que la sous-activité a fini ses traitements, elle peut renvoyer un résultat en appelant la méthode `setResult(Result_OK, result)` qui prend deux paramètres à savoir le code retour et le résultat à renvoyer. Ensuite elle appelle la méthode `finish()`. L'activité appelante récupère ensuite ces résultats au sein de la méthode `onActivityResult(int requestCode, int resultCode, Intent data)`.

Exemple :

Considérons une application qui permette aux utilisateurs d'effectuer une recherche de salon de coiffure dans un rayon relativement proche d'une ville ou d'un code postal. La première interface permettrait à l'utilisateur de saisir une ville ou un code postal, et la seconde interface afficherait une liste des salons de coiffure disponible dans cette localité.

PS : Le maître d'ouvrage de l'application insiste pour que le choix de l'utilisateur s'affiche ensuite dans la première interface (pour des raisons personnelles)...

Récupération de résultat - Activité 1

Afin de transférer les informations à la seconde activité, qui en aura besoin, on prépare un Intent en lui injectant les valeurs saisies par l'utilisateur en « extras ».

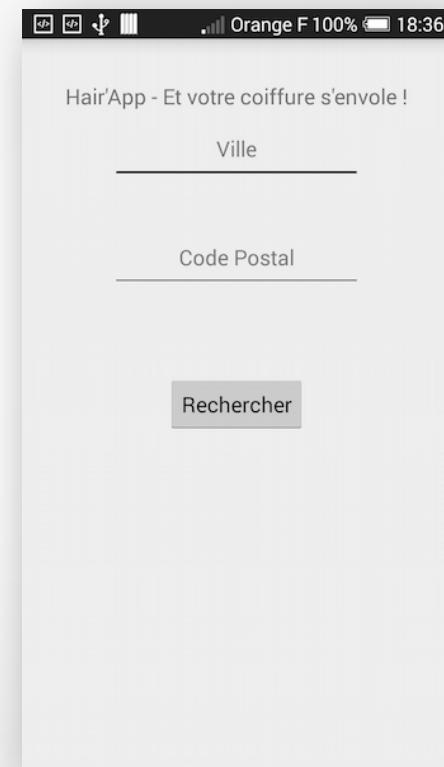
Le démarrage de la sous-activité dont on attend un retour se fait avec la méthode « **startActivityForResult** » qui prend un Intent et un code de requête en paramètre.

```
MainActivity.java
```

```
16
17 public class MainActivity extends Activity implements OnClickListener {
18
19     final int REQUEST_CODE = 1;
20     EditText ville;
21     EditText codePostal;
22
23
24     @Override
25     protected void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         setContentView(R.layout.activity_main);
28
29         ville = (EditText) findViewById(R.id.city);
30         codePostal = (EditText) findViewById(R.id.code);
31
32         Button validateBtn = (Button) findViewById(R.id.validButton);
33         validateBtn.setOnClickListener(this);
34
35
36
37     @Override
38     public void onClick(View v) {
39
40         if(ville.getText() != null && codePostal.getText() != null){
41
42             //On transmet à la seconde activité les informations saisies par l'utilisateur
43             Intent myIntent = new Intent(this, SecondActivity.class);
44             myIntent.putExtra("user_city", ville.getText().toString());
45             myIntent.putExtra("user_code", codePostal.getText().toString());
46
47             startActivityForResult(myIntent, REQUEST_CODE);
48         }
49     }
50 }
```

Récupération de résultat - Activité 1

```
MainActivity.java activity_main.xml
1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     tools:context="com.example.intentapp.MainActivity" >
6
7     <EditText
8         android:id="@+id/city"
9         android:layout_width="wrap_content"
10        android:layout_height="wrap_content"
11        android:layout_alignParentTop="true"
12        android:layout_centerHorizontal="true"
13        android:layout_marginTop="60dp"
14        android:ems="10"
15        android:gravity="center_vertical|center_horizontal"
16        android:hint="Ville" >
17    </EditText>
18
19    <EditText
20        android:id="@+id/code"
21        android:layout_width="wrap_content"
22        android:layout_height="wrap_content"
23        android:layout_centerHorizontal="true"
24        android:layout_below="@+id/city"
25        android:layout_marginTop="50dp"
26        android:gravity="center_vertical|center_horizontal"
27        android:ems="10"
28        android:hint="Code Postal" />
29
```

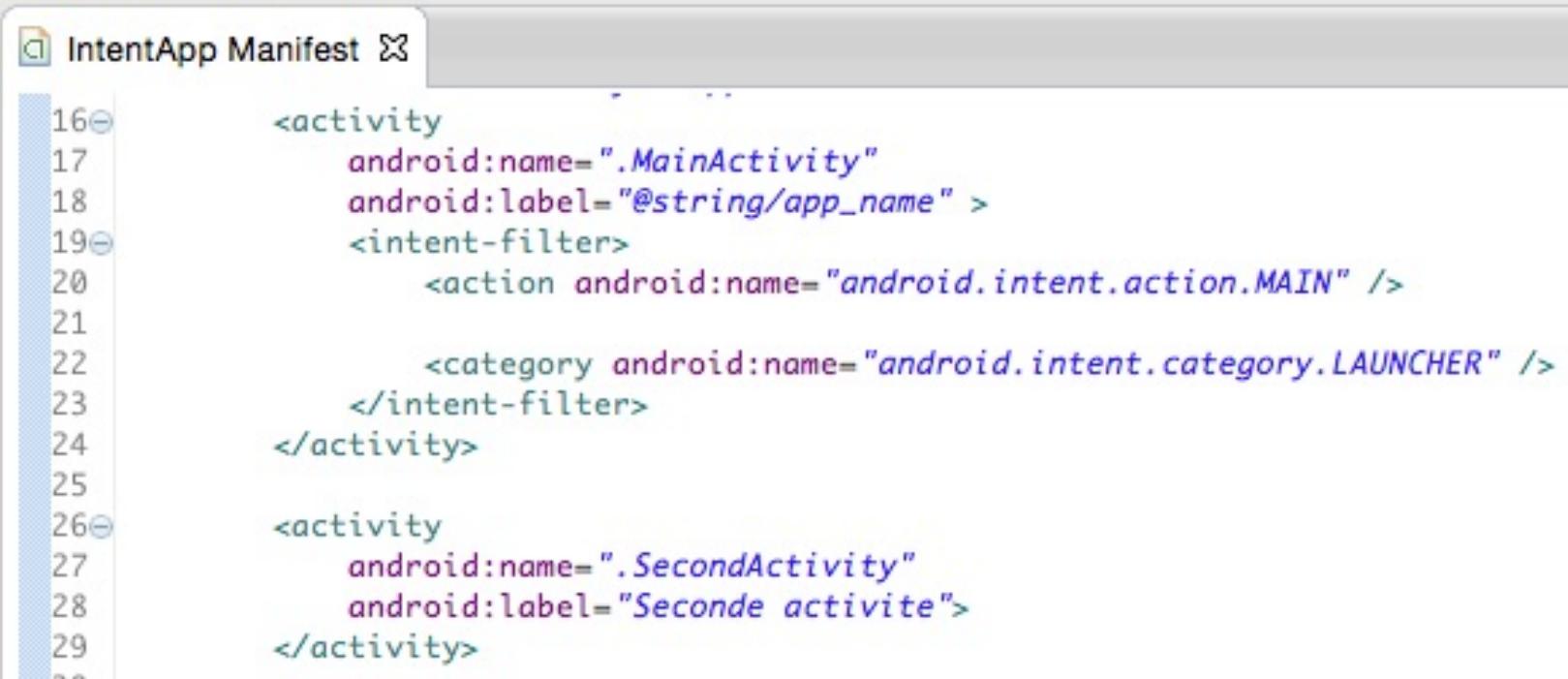


Layouts

```
30     <Button
31         android:id="@+id/validButton"
32         android:layout_width="wrap_content"
33         android:layout_height="wrap_content"
34         android:layout_centerHorizontal="true"
35         android:layout_centerVertical="true"
36         android:text="Rechercher" />
37
38     <TextView
39         android:id="@+id/textView1"
40         android:layout_width="wrap_content"
41         android:layout_height="wrap_content"
42         android:layout_alignParentTop="true"
43         android:layout_centerHorizontal="true"
44         android:layout_marginTop="20dp"
45         android:text="Hair'App - Et votre coiffure s'envole !"
46         android:textAppearance="?android:attr/textAppearanceMedium" />
47
48 </RelativeLayout>
```

Récupération de résultat - Manifeste

Chaque activité doit être déclarée au sein du manifeste !

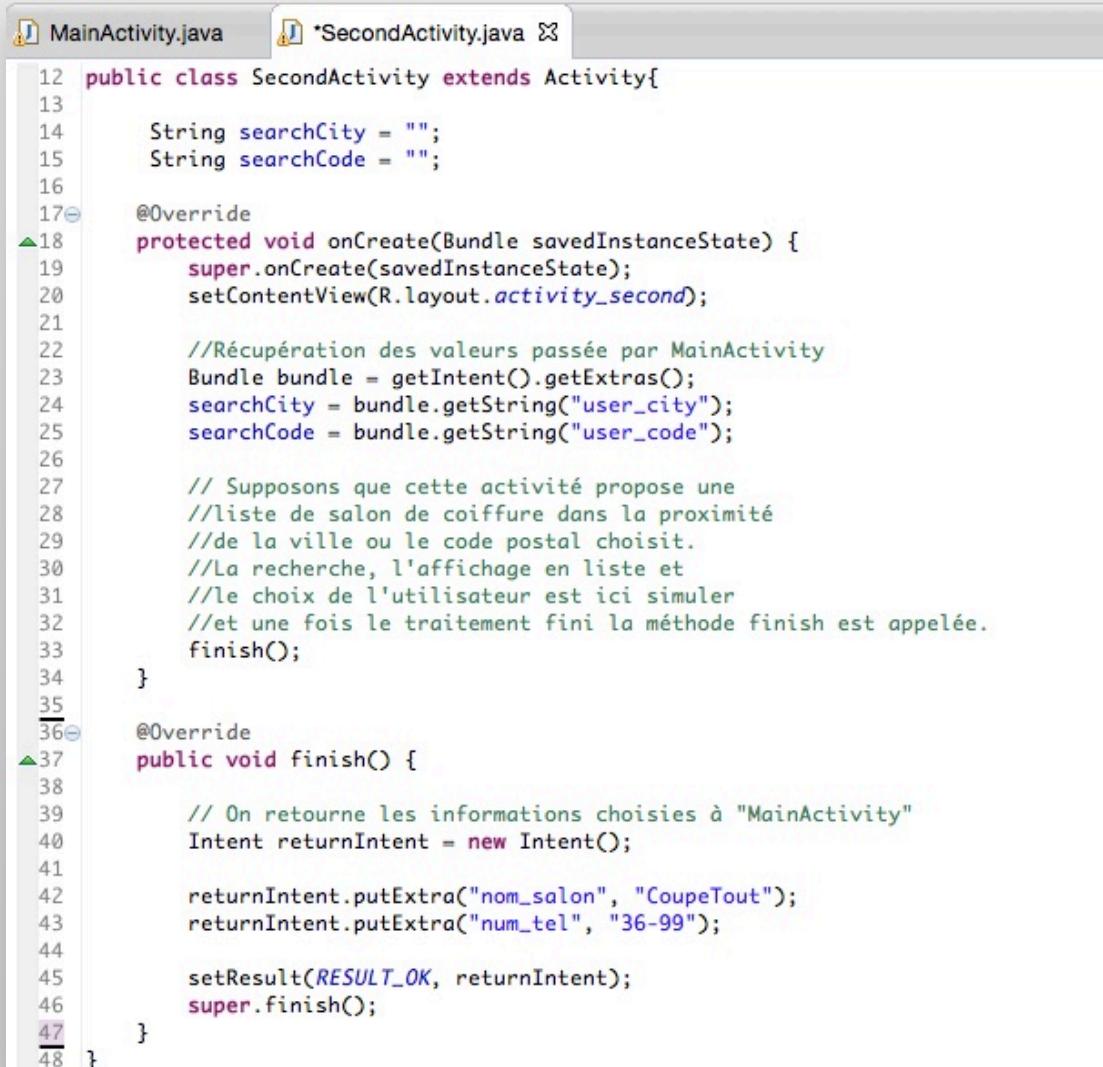


```
IntentApp Manifest

16<activity
17    android:name=".MainActivity"
18    android:label="@string/app_name" >
19    <intent-filter>
20        <action android:name="android.intent.action.MAIN" />
21
22        <category android:name="android.intent.category.LAUNCHER" />
23    </intent-filter>
24</activity>
25
26<activity
27    android:name=".SecondActivity"
28    android:label="Seconde activite">
29</activity>
```

Récupération de résultat – Activité 2

3.1.6



The screenshot shows an IDE interface with two tabs: "MainActivity.java" and "*SecondActivity.java". The code in "SecondActivity.java" is as follows:

```
12 public class SecondActivity extends Activity{
13
14     String searchCity = "";
15     String searchCode = "";
16
17     @Override
18     protected void onCreate(Bundle savedInstanceState) {
19         super.onCreate(savedInstanceState);
20         setContentView(R.layout.activity_second);
21
22         //Récupération des valeurs passée par MainActivity
23         Bundle bundle = getIntent().getExtras();
24         searchCity = bundle.getString("user_city");
25         searchCode = bundle.getString("user_code");
26
27         // Supposons que cette activité propose une
28         //liste de salon de coiffure dans la proximité
29         //de la ville ou le code postal choisit.
30         //La recherche, l'affichage en liste et
31         //le choix de l'utilisateur est ici simuler
32         //et une fois le traitement fini la méthode finish est appelée.
33         finish();
34     }
35
36     @Override
37     public void finish() {
38
39         // On retourne les informations choisies à "MainActivity"
40         Intent returnIntent = new Intent();
41
42         returnIntent.putExtra("nom_salon", "CoupeTout");
43         returnIntent.putExtra("num_tel", "36-99");
44
45         setResult(RESULT_OK, returnIntent);
46         super.finish();
47     }
48 }
```

Lorsque l'activité a fini son traitement, ou que l'utilisateur à appuyer sur le bouton retour, la méthode `finish` est appelé. C'est alors au sein de cette méthode que le retour des valeurs est préparé.

Tout comme dans la première activité, **on prépare un Intent porteur de données**. Le retour à l'activité appelante se fait ainsi grâce à l'utilisation de la méthode **setResult, avant l'appel de la méthode `super.finish()`**.

Récupération de résultat – Activité 1

La réception des données se fait alors dans l'activité appelante. Dès que celle-ci est remise en avant-plan, la méthode `onActivityResult` est appelée ce qui permet de traiter les données retournées par la sous-activité. Cette méthode prend en paramètre le code de requête qui a été utilisé pour lancer la sous-activité, le code de la sous-activité pour indiquer son résultat, et l'intention porteurs des données.

```
*MainActivity.java *SecondActivity.java
54
55@Override
56protected void onActivityResult(int requestCode, int resultCode, Intent data) {
57    super.onActivityResult(requestCode, resultCode, data);
58
59    if (resultCode == RESULT_OK) {
60        if (data.getStringExtra("nom_salon") != null && data.getStringExtra("num_tel") != null) {
61            Toast.makeText(this, data.getStringExtra("nom_salon"), Toast.LENGTH_SHORT).show();
62            Toast.makeText(this, data.getStringExtra("num_tel"), Toast.LENGTH_SHORT).show();
63        }
64    }
65    else if(resultCode == RESULT_CANCELED){
66        Toast.makeText(this, "Recherche interrompue", Toast.LENGTH_SHORT).show();
67    }
68}
69}
```

3.2 – Broadcast Receiver

Recevoir un évènement

Le Broadcast Receiver (récepteur de diffusion) est un mécanisme qui permet d'envoyer et recevoir des évènements grâce à des intentions. Pour qu'un récepteur puisse recevoir des diffusions, il faut au préalable qu'il ait été enregistré au sein du manifeste. Au sein de cette déclaration « de réception », il est nécessaire de préciser quels types d'actions le récepteur écoutera. C'est en réalité une implémentation du pattern observer qui est fait ici.

```
9 public class BootReceiver extends BroadcastReceiver{  
10  
11    @Override  
12    public void onReceive(Context context, Intent intent) {  
13        //  
14    }  
15 }
```

Pour déclarer un récepteur de diffusion, il faut étendre votre classe avec la classe BroadcastReceiver, vous obligeant ainsi à implémenter la méthode onReceive. Cette méthode prend en paramètre un context et un intent, elle est directement appelée dès que les événements correspondant aux actions que vous avez ciblés dans le manifeste surviennent. Mais que met-on dans ce manifeste ?

Récepteur de manifeste

```
<receiver  
    android:name="com.example.Nom_De_Mon_Receiver">  
    <intent-filter android:priority="100">  
        <action android:name="Action_A_Ecouter"/>  
    </intent-filter>  
</receiver>
```

Il est également possible de spécifier une balise **android:exported="true"** pour préciser si le récepteur peut être utilisé par des activités hors à notre application

```
<uses-permission android:name="android.permission.PERMISSION_ACTION"/>
```

Lorsque le récepteur est enregistré dans le manifeste comme dans ce cas, on parle de récepteur de manifeste.

!\\ Information importante : Les applications qui disposent de récepteur de manifeste n'ont pas besoin d'être en cours d'exécution pour capter les intention. Il est ainsi possible de lancer ces applications automatiquement et prévoir des comportements en fonction de certain événement qui se produisent sur le mobile sans que l'utilisateur ait à lancer ces dites applications !

Actions de diffusion natives

ACTION_BOOT_COMPLETED : récepteur déclenché quand la plateforme mobile a terminé sa séquence de démarrage. Nécessite de rajouter la permission RECEIVE_BOOT_COMPLETED

ACTION_SCREEN_OFF / ACTION_SCREEN_ON : Récepteur diffusé quand l'écran s'allume ou s'éteint

ACTION_CAMERA_BUTTON : Déclenchée quand un appui sur le bouton de l'appareil photo est effectué.

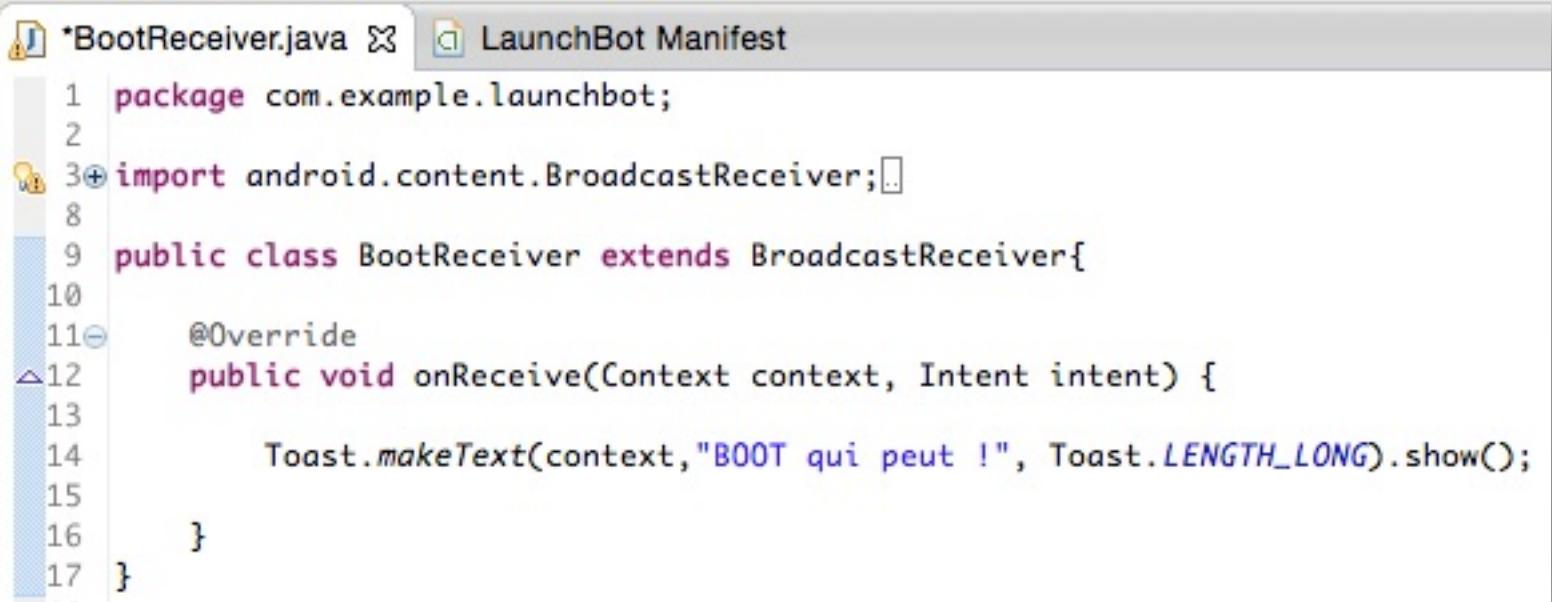
ACTION_DATE_CHANGED / ACTION_TIME_CHANGED : Diffusé quand l'heure ou la date sont changées manuellement.

ACTION_NEW_OUTGOING_CALL : Diffusé quand un appel nouvel appel sortant va effectué. Nécessite la permission PROCESS_OUTGOING_CALLS

La liste des actions natives est bien entendu disponible sur la documentation :
<https://developer.android.com/reference/android/content/Intent.html>

Exemple – Récepteur (Boot)

Prenons un exemple, nous allons créer un récepteur de diffusion qui se lance dès la fin de la séquence de démarrage d'une plateforme mobile. Commençons par créer notre récepteur « BootReceiver ».



The screenshot shows an IDE interface with two tabs at the top: '*BootReceiver.java' and 'LaunchBot Manifest'. The code editor displays the following Java code:

```
1 package com.example.launchbot;
2
3+import android.content.BroadcastReceiver;
4
5 public class BootReceiver extends BroadcastReceiver{
6
7     @Override
8     public void onReceive(Context context, Intent intent) {
9
10         Toast.makeText(context, "BOOT qui peut !", Toast.LENGTH_LONG).show();
11     }
12 }
```

Ici, le déclenchement du récepteur ne fait qu'afficher un Toast pour nous permettre de visualiser simplement ce déclenchement, mais on pourrait très bien lancer une activité particulière...

Exemple – Manifeste (Boot)

The screenshot shows an AndroidManifest.xml file in an IDE. The file defines a package named com.example.launchbot with version 1.0. It specifies a minimum SDK version of 8 and targets version 21. A permission for RECEIVE_BOOT_COMPLETED is requested. The application contains a Main Activity and a Boot Receiver. The Boot Receiver is highlighted with a red rectangle and has an intent filter for the BOOT_COMPLETED action.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.launchbot"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="21" />

    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        <receiver android:name=".BootReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
    </application>

```

Enregistrement au récepteur

Certains récepteurs de diffusion nécessitent que l'on s'enregistre, notamment ceux qui affectent l'interface utilisateur. Ainsi, ces récepteurs ne pourront répondre aux intentions que si l'activité (ou service, etc..) où ils se sont enregistrés a été lancé précédemment. Pour s'abonner à un récepteur, on utilise la méthode « **registerReceiver(receiver, filter)** » qui prend en paramètre un receiver et un filtre d'intention. Pour se désabonner, on utilisera la méthode **unregisterReceiver(receiver)** en lui passant le receiver à quitter.

Exemple : Supposons, que le constructeur de notre plateforme mobile ait oublié d'intégrer le niveau de batterie restante dans la barre de notification. Tout comme une jauge d'essence en panne sur une voiture, cela est particulièrement désappointant...

Intégrons donc au sein d'une petite application un récepteur de diffusion nous notifiant les changements d'état du niveau de batterie.



BatteryChecker - Manifeste

BatteryReceiver.java MainActivity.java BatteryChecker Manifest

```

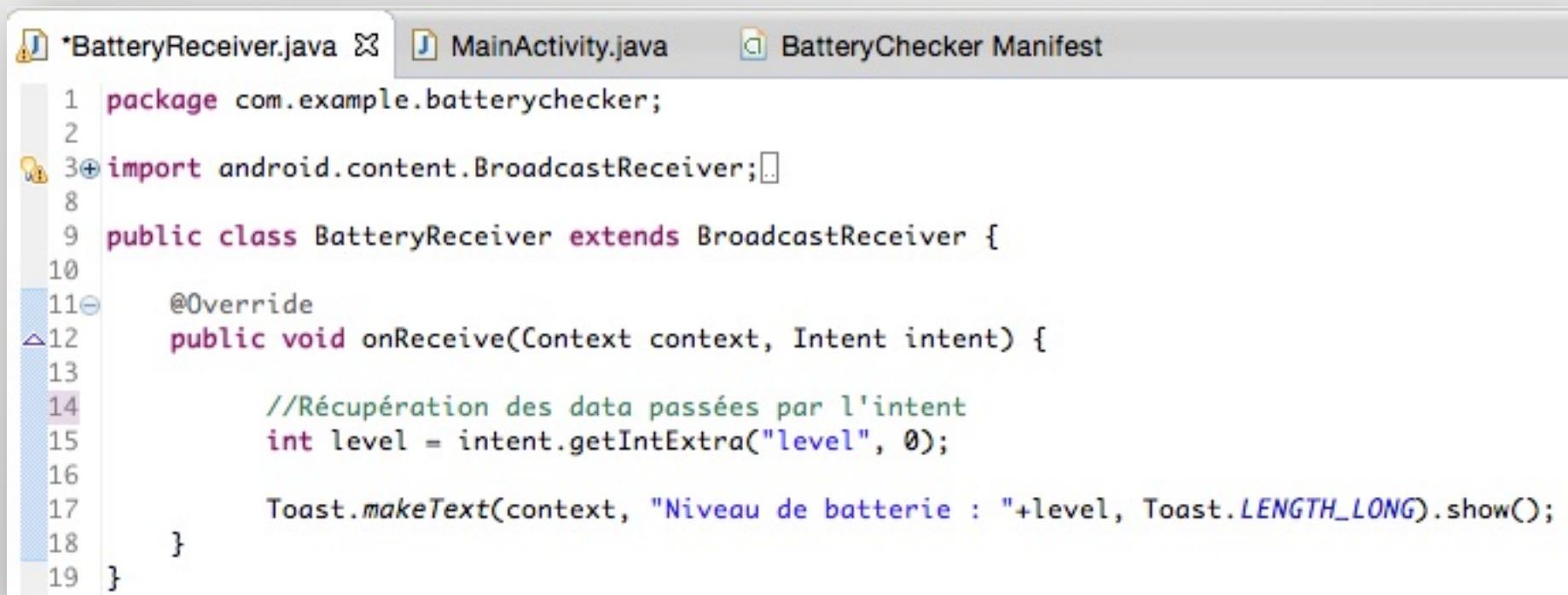
1  <?xml version="1.0" encoding="utf-8"?>
2<manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.batteryindicator"
4     android:versionCode="1"
5     android:versionName="1.0" >
6     <uses-sdk
7         android:minSdkVersion="8"
8         android:targetSdkVersion="21" />
9
10    <uses-permission android:name="android.permission.BATTERY_STATS"/>
11
12<application
13     android:allowBackup="true"
14     android:icon="@drawable/ic_launcher"
15     android:label="@string/app_name"
16     android:theme="@style/AppTheme" >
17     <activity
18         android:name="com.example.batterychecker.MainActivity"
19         android:label="@string/app_name" >
20         <intent-filter>
21             <action android:name="android.intent.action.MAIN" />
22
23             <category android:name="android.intent.category.LAUNCHER" />
24         </intent-filter>
25     </activity>
26
27     <receiver
28         android:name="com.example.batterychecker.BatteryReceiver">
29         <intent-filter android:priority="100">
30             <action android:name="android.intent.action.BATTERY_CHANGED" />
31         </intent-filter>
32     </receiver>
33 </application>
```

Cela nécessite la permission android.permission.BATTERY_STATS



Les actions à prendre en compte au sein du manifeste sont : android.intent.action.BATTERY_CHANGED

BatteryChecker - Receiver



```
1 package com.example.batterychecker;
2
3+import android.content.BroadcastReceiver;[]
4
5 public class BatteryReceiver extends BroadcastReceiver {
6
7     @Override
8     public void onReceive(Context context, Intent intent) {
9
10         //Récupération des data passées par l'intent
11         int level = intent.getIntExtra("level", 0);
12
13         Toast.makeText(context, "Niveau de batterie : "+level, Toast.LENGTH_LONG).show();
14     }
15 }
16
17 }
```

Au sein du récepteur de diffusion, nous pouvons ainsi récupérer les données contenues dans l'intention « **intent** ». Dans ce cas les données passées en « **extra** » concernent le niveau de batterie.

BatteryChecker - MainActivity

The screenshot shows the Android Studio interface. On the left, the code editor displays `MainActivity.java` with the following content:

```
1 package com.example.batterychecker;
2
3+import com.example.batteryindicator.R;□
11
12
13 public class MainActivity extends ActionBarActivity {
14
15@    @Override
16    protected void onCreate(Bundle savedInstanceState) {
17        super.onCreate(savedInstanceState);
18        setContentView(R.layout.activity_main);
19
20        //Instanciation d'un Battery Receiver
21        BatteryReceiver theReceiver = new BatteryReceiver();
22
23        //Instanciation d'un filtre d'intention
24        IntentFilter theIntentFilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
25
26        //enregistrement au receiver
27        registerReceiver(theReceiver, theIntentFilter);
28
29    }
```

On the right, the Android emulator window titled "Battery Checker" shows the application running. The status bar at the top indicates signal strength, battery level (Orange F 43%), and time (09:58). A notification bubble in the bottom right corner of the screen displays the text "Niveau de batterie : 43".

Au sein de l'activité, il est alors nécessaire de s'enregistrer au récepteur via la méthode `registerReceiver(receiver,intentFilter)`. Cette méthode prend en paramètre une instance notre récepteur de diffusion, à savoir `BatteryReceiver`, et le filtre d'intention qui se concentre sur le changement d'état de la batterie.

Créer vos intentions de diffusion

Il est par ailleurs possible de créer ses propres intentions et les diffuser. Ainsi, il faut préparer l'intention à diffuser, l'action et les données à intégrer.

La chaîne d'action de l'intention permet d'identifier un événement diffusé, elle doit être unique. L'intégration des données au sein de l'intention se fait comme précédemment avec la méthode « `putExtra` ». Une fois prêt, il ne reste plus qu'à diffuser avec la méthode « `sendBroadcast(intent)` ».

```
String MY_BROADCAST = "com.example.android.BROADCAST_TEST";  
  
Intent broadcastToSend = new Intent(MY_BROADCAST);  
sendBroadcast(broadcastToSend);
```

Plan – Chapitre 3

- 4. Tâche asynchrone, service, Webview, notification
 - 4.1 Tâches asynchrones (AsyncTask)
 - 4.2 Service
 - Pause
 - 4.3 WebView
 - 4.4 Notification

Chapitre N°4

Tâche asynchrone, service,
Webview, notification

4.1 – Tâche asynchrone - AsyncTask

AsyncTask

Une application mobile ne disposant pas d'une interface fluide et réactive ne sera pas véritablement bien perçue par les utilisateurs finaux. Les utilisateurs remarquent les délais dans les saisies et les pauses de l'interface dès qu'ils dépassent 200 millisecondes !

Les traitements relativement lourds doivent donc être traités au sein de tâches parallèles, des threads, en arrière-plan. Par exemple, les accès fichiers, accès réseaux, transactions sur les bdd, calculs complexes..

Android intègre ainsi une classe AsyncTask permettant de définir une opération qui sera exécutée en arrière plan. Une fois terminés, les résultats seront alors postés dans le thread principal de l'utilisateur. Ce mécanisme est particulièrement adapté pour les traitements d'arrière-plan courts, dont la progression et les résultats doivent être ensuite affichés sur l'interface utilisateur. Malgré la durée relativement courte des traitements, les effectuer au sein du thread principal de l'interface occuperait trop de ressources et figerait les éléments graphiques.

Attention, ces tâches ne persistent pas entre les redémarrages de l'activité. Autrement dit, si l'orientation de la plateforme mobile change, la tâche asynchrone sera annulée.

AsyncTask – Implémentation

```

LoadAsyncActivity.java ✘

4
5
6 public class LoadAsyncActivity extends AsyncTask<String, Integer, String>{
7
8     @Override
9     protected String doInBackground(String... params) {
10
11         //Thread permettant d'exécuter
12         //les traitements lourds en
13         //tâche de fond
14         return null;
15     }
16
17     @Override
18     protected void onProgressUpdate(Integer... progress){
19
20         //Mise à jour de la progression
21         //du thread en cours d'exécution.
22         //cela permet par exemple de mettre
23         //à jour la barre d'avancement pour notifier
24         //l'état du traitement à l'utilisateur.
25         //Appelé par publishProgress
26
27     }
28
29     @Override
30     protected void onPostExecute(String result){
31
32         //Une fois le traitement lourd
33         //terminé, on effectue la mise
34         //à jour de l'interface utilisateur
35         //avec les résultats retournés
36
37     }
38 }
```

Pour utiliser ce mécanisme, il faut hériter de la classe `AsynTask`. Nous pouvons ensuite redéfinir ses méthodes :

- **`doInBackground()`** qui exécutera son traitement dans un thread supplémentaire.
- **`onProgressUpdate()`** qui permettra de mettre à jour la progression de la tâche
- **`onPostExecute()`** qui mettra à jour l'interface utilisateur avec les résultats du traitement une fois la méthode `doInBackground` terminée.
- **Éventuellement, `onPreExecute()`** qui servira à préparer le traitement (variables, interface, etc..). Attention cette méthode est exécutée dans le thread principal.

AsyncTask – Implémentation

Prenons un exemple récurrent, l'avancement d'une barre de progression en fonction d'un traitement conséquent (parsing d'un flux rss, téléchargement d'un fichier, etc..).

Au sein de notre activité, créons une classe interne qui hérite donc de la classe AsyncTask. Il est ainsi nécessaire d'implémenter la méthode doInBackground() dans laquelle nous allons simuler un traitement lourd avec une boucle.

```
//----- Classe interne Progress AsyncTask -----//  
  
public class Progress AsyncTask <Void, Integer, Void> {  
  
    //L'état de la progression  
    int theProgress;  
  
    //On prépare l'exécution de la tâche  
    @Override  
    protected void onPreExecute() {  
        theProgress = 0;  
    }  
  
    //Dès que le système le peut, il lance la tâche en fond  
    //Ici un traitement simuler par une boucle 0 à 100  
    // faisant une pause de 50 ms à chaque itération  
    @Override  
    protected Void doInBackground(Void... params) {  
  
        while(theProgress<100){  
            theProgress++;  
            publishProgress(theProgress); //Publication de la progression  
  
            try {  
                Thread.sleep(50);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
        return null;  
    }  
}
```

AsyncTask – Implémentation

```
@Override  
protected void onProgressUpdate(Integer... values) {  
    progressBar.setProgress(values[0]); //Mise à jour graphique de la progression  
}  
  
//Une fois le traitement terminé la méthode onPostExecute est appelée  
//on mets alors à jour l'interface utilisateur avec le résultat  
@Override  
protected void onPostExecute(Void result) {  
    super.onPostExecute(result);  
    Toast.makeText(MainActivity.this, "Traitement Terminé", Toast.LENGTH_LONG).show();  
}
```

Lorsque nous appelons la méthode **publishProgress(int progress)** dans l'exécution de notre tâche, c'est **onProgressUpdate(Integer... values)** qui est exécutée par le système.

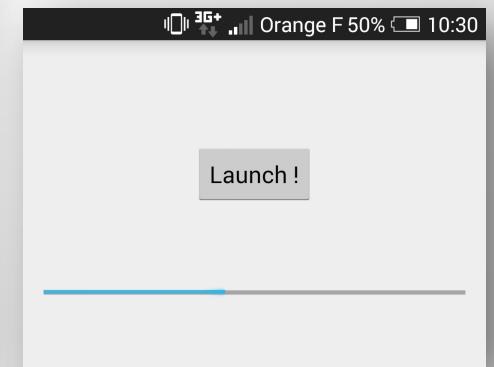
Cela nous permet ainsi de mettre à jour l'avancement de la progressBar avec le paramètre passé.

Une fois le traitement « lourd » terminé, la méthode **onPostExecute** est appelée et nous pouvons alors afficher les résultats sur l'interface utilisateur (ici un Toast...).

AsyncTask – Implémentation

```
public class MainActivity extends Activity {  
  
    ProgressBar progressBar;  
    Button launchButton;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        //On instancie les éléments graphiques  
        launchButton = (Button)findViewById(R.id.launchBtn);  
        progressBar = (ProgressBar)findViewById(R.id.progressBar1);  
        progressBar.setProgress(0);  
  
        // À l'appui du bouton, je lance ma tâche asynchrone avec la méthode execute  
        launchButton.setOnClickListener(new Button.OnClickListener(){  
            @Override  
            public void onClick(View v) {  
                new Progress AsyncTask().execute();  
            }  
        });  
    }  
}
```

Dans notre activité,
nous pouvons lancer
notre tâche
asynchrone dès que
nous le souhaitons,
avec la méthode
execute()



4.2 – Service

Service

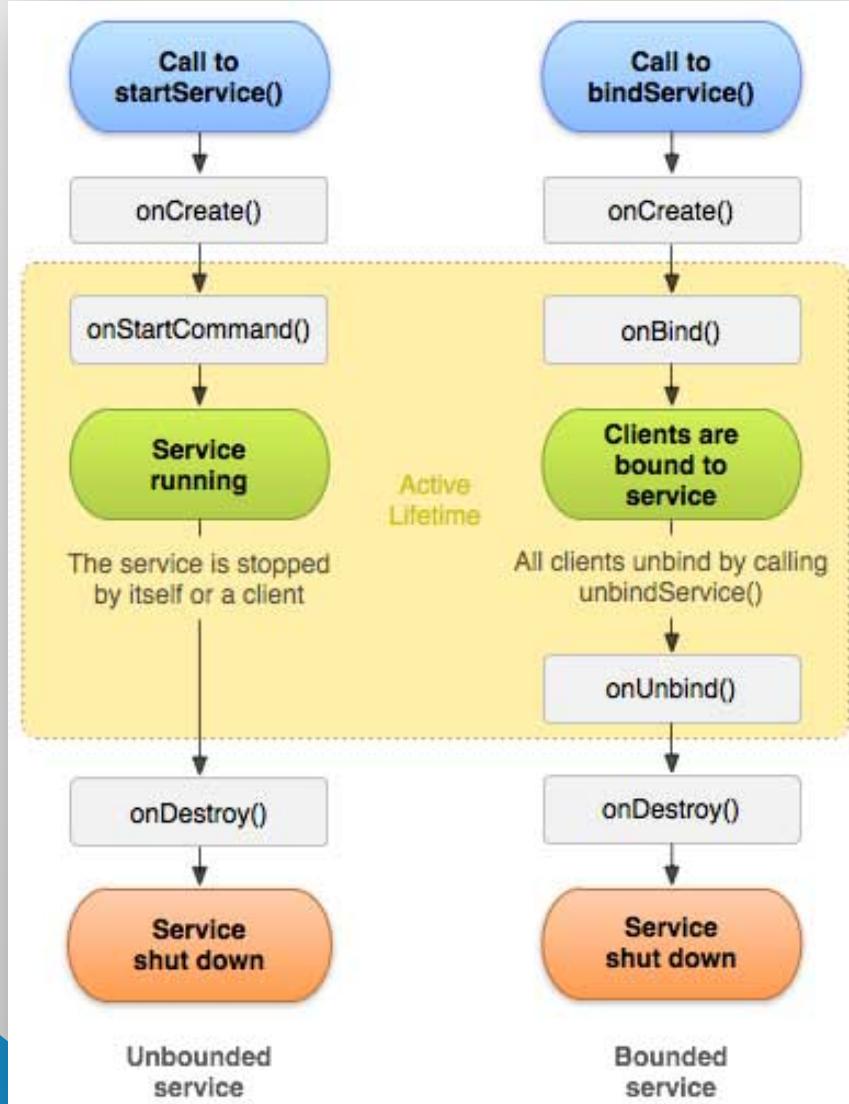
Les services permettent d'effectuer des traitements sur des données, de déclencher des notifications, de mettre à jour certaines informations, sans aucune intervention de l'utilisateur. Il n'y a ici aucune interface utilisateur contrairement aux activités. Ces services sont conçus pour fonctionner sans limites de temps, mais on peut tout de même les arrêter selon les besoins. Soit en fonction de leur cycle de vie prévu par le développeur, soit un arrêt effectué par le système dans le cas d'un besoin de ressources. D'une manière générale on trouve deux types de services:

- **Services locaux (unboundservice)** : L'activité qui a lancé le service et le service lui-même font partie de la même application.
- **Service distant (boundservice)** : Les services sont lancés par un composant qui appartient à une autre application.

Pour créer un service, il faut créer une classe qui hérite de « Service ». Il est ainsi ensuite possible de redéfinir les méthodes de cette dernière, comme dans le fonctionnement des activités. Justement, tout comme les activités, les services sont régis par un cycle de vie.

Nous ne verrons qu'une introduction aux services dans ce cours, pour les plus curieux :
<https://developer.android.com/reference/android/app/Service.html>

Cycle de vie d'un service



Plusieurs méthodes seront ainsi à implémenter dans notre classe service permettant de gérer cette dernière en fonction du cycle de vie:

onCreate : Permettra d'initialiser le service

onStartCommand : Exécute le traitement prévu par le service. Cette méthode est **appelée par le système à la suite d'onCreate** lorsque l'on appelle la méthode `startService` dans une activité. Elle retourne un entier permettant de préciser le comportement du service s'il est tué par le système (redémarrage, etc..)

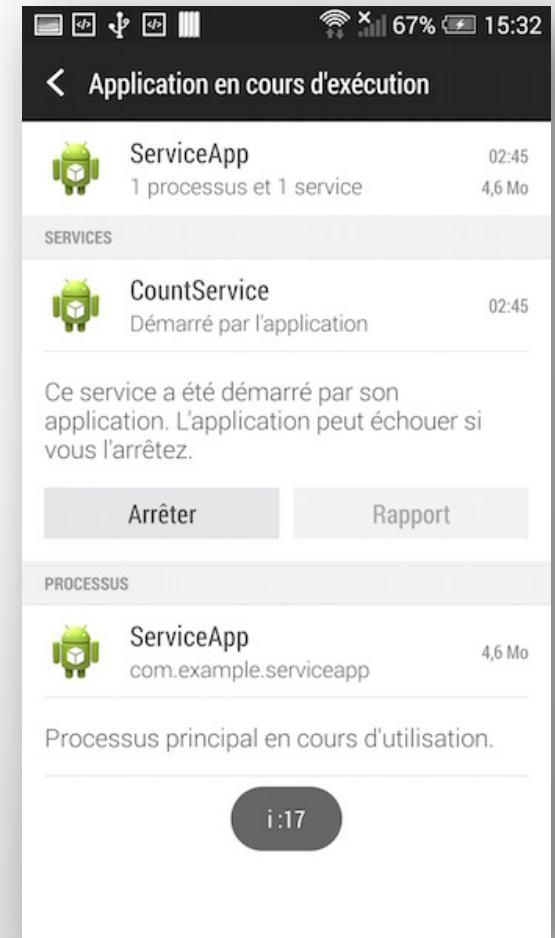
onBind : permettra de lier un service à une activité. Utilisé quand un service a besoin de mettre à jour une activité.

Exemple

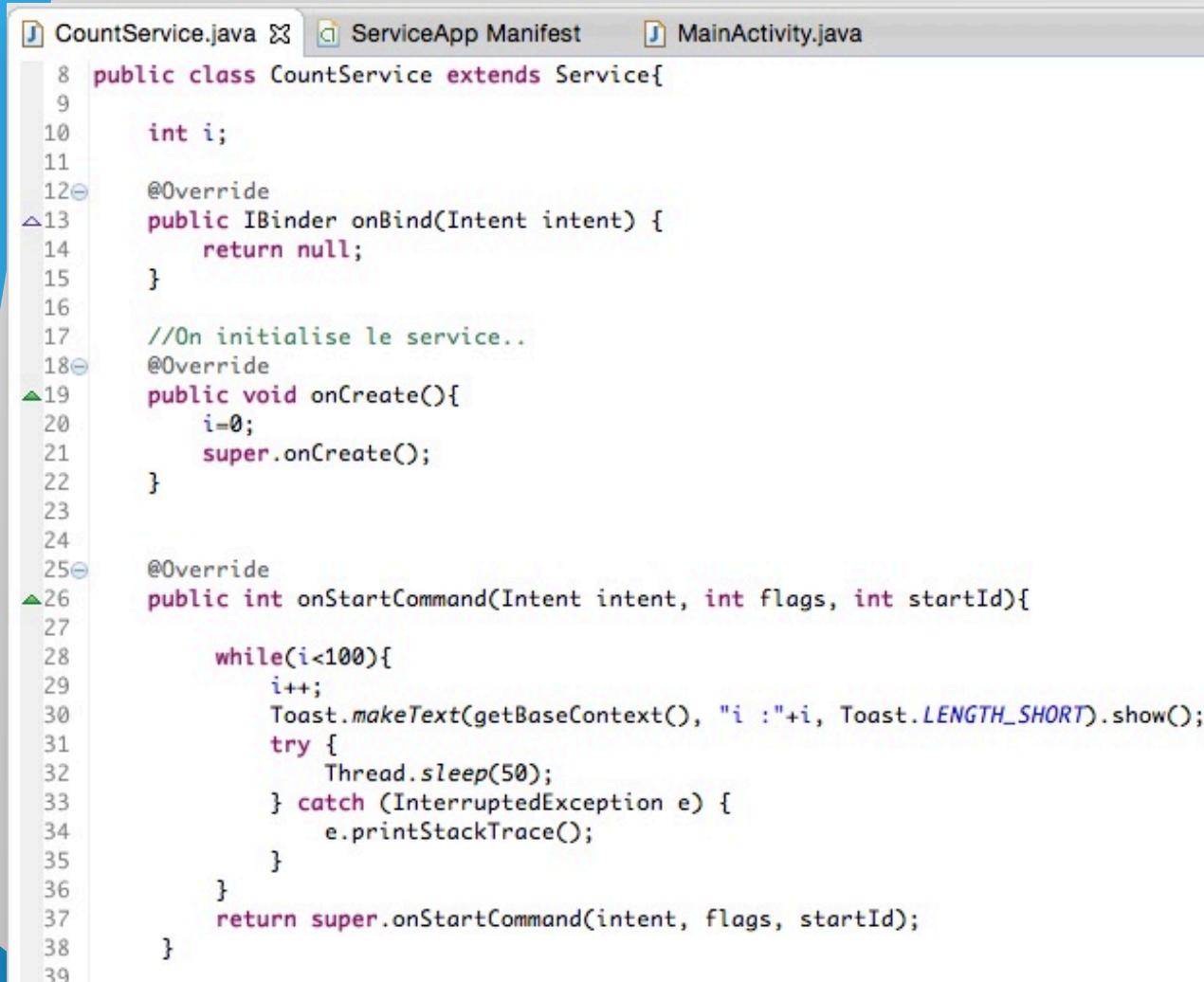
Prenons un exemple, inutile, mais qui permet d'illustrer ce mécanisme. Il s'agit ici de mettre en œuvre un service qui traite une simple boucle jusqu'à 100 pour simuler un traitement.

Il faut alors :

- Créer notre classe qui hérite de service
- Implémenter notre boucle
- Autoriser son exécution dans le manifeste
- Démarrer notre service dans une activité.



Exemple - CountService



The screenshot shows an IDE interface with three tabs at the top: 'CountService.java', 'ServiceApp Manifest', and 'MainActivity.java'. The 'CountService.java' tab is active, displaying the following Java code:

```
8 public class CountService extends Service{
9
10    int i;
11
12    @Override
13    public IBinder onBind(Intent intent) {
14        return null;
15    }
16
17    //On initialise le service..
18    @Override
19    public void onCreate(){
20        i=0;
21        super.onCreate();
22    }
23
24
25    @Override
26    public int onStartCommand(Intent intent, int flags, int startId){
27
28        while(i<100){
29            i++;
30            Toast.makeText(getApplicationContext(), "i :" + i, Toast.LENGTH_SHORT).show();
31            try {
32                Thread.sleep(50);
33            } catch (InterruptedException e) {
34                e.printStackTrace();
35            }
36        }
37        return super.onStartCommand(intent, flags, startId);
38    }
39}
```

Une fois le service lancé, le système appelle donc la méthode `onCreate` qui permet d'initialiser le service.

Ensuite, la méthode `onStartCommand` est appelée et exécute le traitement prévu à savoir notre boucle.

Exemple

```

15
16    @Override
17    protected void onCreate(Bundle savedInstanceState) {
18        super.onCreate(savedInstanceState);
19        setContentView(R.layout.activity_main);
20
21        Button startBtn = (Button) findViewById(R.id.startServBtn);
22
23        startBtn.setOnClickListener(new OnClickListener(){
24
25            @Override
26            public void onClick(View v) {
27                //Démarrage du service
28                startService(new Intent(MainActivity.this, CountService.class));
29            }
30        });
31    }
}

```

Le démarrage du service se fait grâce à la méthode startService qui prend un Intent en paramètre.

Il est nécessaire de déclarer le service au sein du nœud application avec une balise service dans le manifeste. On peut également préciser l'attribut **permission**. Les applications tierces devront alors inclure une balise **uses-permission** pour pouvoir accéder à votre service.

```

<service
    android:enabled="true"
    android:name=".CountService"/>

```

4.3 – Webview

Webview

Les WebView sont des composants permettant d'intégrer des pages web HTML et les outils basiques d'un navigateur (js, css, zoom...) au sein d'une application. Pour faire cela, on utilise un composant WebView sur lequel on peut appliquer divers paramètres.

```
WebView webview = (WebView) findViewById(R.id.webView);
webview.loadUrl("http://docs.oracle.com/javase/7/docs/api/");
```

Comme le montre l'exemple ci-dessus, une fois le composant lié à l'activité, il ne reste plus qu'à charger une URL avec la méthode loadUrl en lui passant en paramètre l'URL souhaitée. Attention cela nécessite d'avoir la permission d'utiliser l'accès internet précisé dans le manifeste..

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Webview – Javascript / Paramètres

Il est par ailleurs possible de paramétriser cette webview en fonction des besoins, et de charger directement du code HTML à partir de l'activité.

```
WebView webview = (WebView) findViewById(R.id.webView);
webview.loadUrl("http://docs.oracle.com/javase/7/docs/api/");

//récupération des paramètres avec la méthode getSettings
WebSettings params = webview.getSettings();

//activation du javascript
params.setJavaScriptEnabled(true);

//ajout des boutons de zoom
params.setBuiltInZoomControls(true);

String myPage = "<html><body><h1>Ma page dans ma webview</h1></body></html>";
// chargement de code html
webview.loadData(myPage, "text/html", "UTF-8");
```

Webview – Interface Javascript

The screenshot shows the Android Studio interface with two tabs open: `MainActivity.java` and `WebAppInterface.java`. The `MainActivity.java` tab is active, displaying the following Java code:

```
11 public class MainActivity extends ActionBarActivity {
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17
18
19         WebView webview = (WebView) findViewById(R.id.webView);
20
21         //récupération des paramètres avec la méthode getSettings
22         WebSettings params = webview.getSettings();
23
24         //activation du javascript
25         params.setJavaScriptEnabled(true);
26
27         //ajout des boutons de zoom
28         params.setBuiltInZoomControls(true);
29
30         //Ajout d'une interace javascript à la webview
31         webview.addJavascriptInterface(new WebAppInterface(this), "Android");
32
33         String myPage = "<input type=\"button\" value=\"Toast Me\""
34             + " onClick=\"showAndroidToast('Javascript Toast!')\" />" +
35             "<script type=\"text/javascript\">" +
36             "function showAndroidToast(toast) {" +
37             "Android.showToast(toast);"
38             + "}</script>";
39
40         webview.loadData(myPage, "text/html", "UTF-8");
41
42     }
}
```

Afin de pouvoir exécuter du code natif au sein de notre JavaScript, il est par ailleurs possible de créer une interface JS dans laquelle définir nos méthodes appelées.

Dans cet exemple un bouton HTML permet d'appeler la **fonction showAndroidToast()** en **JavaScript**. Cette fonction appelant notre **code natif Android.showToast()** au sein de l'interface JS **WebAppInterface**.

Webview – Interface Javascript

4.3.2

The image shows a split-screen view. On the left, the Android Studio IDE displays the Java code for a class named `WebAppInterface.java`. The code defines a class that registers itself as a JavaScript interface and provides a method to display a toast message. On the right, a screenshot of an Android device or emulator shows a custom browser application titled "BrowsApp". The browser has a URL input field containing "Saisissez votre url ici" and a "Go" button. Below the input field is a button labeled "Toast Me". A toast message is visible at the bottom of the screen with the text "Javascript Toast!". The top status bar of the device shows signal strength, battery level (Orange F 100%), and the time (20:11).

```
1 package com.example.browsapp;
2
3+import android.content.Context;
4
5 public class WebAppInterface {
6
7     Context mContext;
8
9     // Instanciation de l'interface javascript et enregistrement du contexte //
10    WebAppInterface(Context c) {
11        mContext = c;
12    }
13
14
15    //Affichage d'un toast sur la web page
16    @JavascriptInterface
17    public void showToast(String toast) {
18        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
19    }
20}
21}
```

WebView – WebViewClient



Il est donc possible d'afficher une page HTML dans un composant WebView. Cependant, pour afficher certaines URL notre activité et son widget ne permettront pas cette opération.

Il faut alors créer une classe qui hérite de WebViewClient pouvant prendre en charge tout type d'URL au sein du widget.

Dans cette dernière il est nécessaire de redéfinir la méthode **shouldOverrideUrlLoading(WebView view, String url)** prenant en paramètre notre widget webview et l'URL à charger. Le retour de la méthode permet de préciser si on prend en charge l'URL passée (true) ou non (false).

Aussi, si notre WebViewClient personnalisée n'existe pas, l'Activity Manager demandera à l'utilisateur de choisir une application disponible pour traiter l'URL.

Créons notre propre navigateur !

Webview – WebViewClient

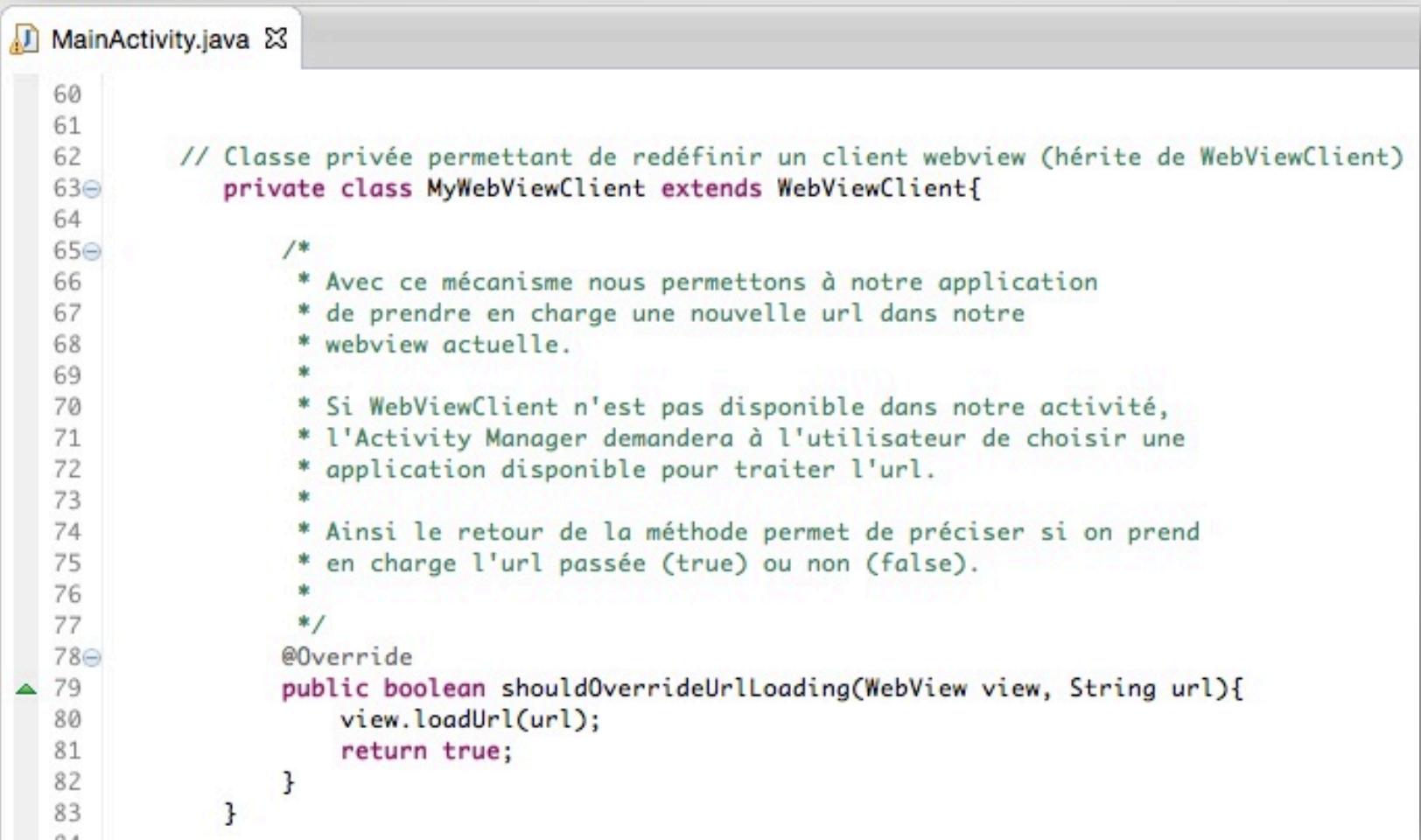
Comme dans l'exemple précédent, nous initialisons notre webview au sein de la méthode onCreate. Une fois les paramètres appliqués il suffit de passé une instance de notre WebViewClient personnalisé en paramètre de la méthode setWebViewClient()

```
MainActivity.java ✘
19 public class MainActivity extends ActionBarActivity implements OnClickListener {
20
21     WebView webview;
22     EditText url;
23     Button goBtn;
24
25     @Override
26     protected void onCreate(Bundle savedInstanceState) {
27         super.onCreate(savedInstanceState);
28         setContentView(R.layout.activity_main);
29
30         url = (EditText) findViewById(R.id.urlText);
31         goBtn = (Button) findViewById(R.id.validButton);
32         goBtn.setOnClickListener(this);
33         webview = (WebView) findViewById(R.id.webView);
34
35         //récupération des paramètres avec la méthode getSettings
36         WebSettings params = webview.getSettings();
37         //activation du javascript
38         params.setJavaScriptEnabled(true);
39         //ajout des boutons de zoom
40         params.setBuiltInZoomControls(true);
41         //Ajout d'une interface javascript à la webview
42         webview.addJavascriptInterface(new WebAppInterface(this), "Android");
43
44         //Attribution d'un client webview personnalisé
45         webview.setWebViewClient(new MyWebViewClient());
46     }
47
48     @Override
49     public void onClick(View v) {
50         //Au clic on charge l'url
51         switch(v.getId()){
52             case R.id.validButton:
53                 webview.loadUrl(url.getText().toString());
54         }
55     }
}
```

Webview – WebViewClient

4.3.3

Créons notre classe héritant de WebViewClient pouvant prendre compte les URL



The screenshot shows a Java code editor window titled "MainActivity.java". The code defines a private inner class "MyWebViewClient" that extends "WebViewClient". This class overrides the "shouldOverrideUrlLoading" method to return true, causing the web view to load the provided URL. The code is annotated with comments explaining the purpose of the class and the overridden method.

```
60
61
62 // Classe privée permettant de redéfinir un client webview (hérite de WebViewClient)
63 private class MyWebViewClient extends WebViewClient{
64
65     /*
66      * Avec ce mécanisme nous permettons à notre application
67      * de prendre en charge une nouvelle url dans notre
68      * webview actuelle.
69      *
70      * Si WebViewClient n'est pas disponible dans notre activité,
71      * l'Activity Manager demandera à l'utilisateur de choisir une
72      * application disponible pour traiter l'url.
73      *
74      * Ainsi le retour de la méthode permet de préciser si on prend
75      * en charge l'url passée (true) ou non (false).
76      */
77
78     @Override
79     public boolean shouldOverrideUrlLoading(WebView view, String url){
80         view.loadUrl(url);
81         return true;
82     }
83 }
```

Webview – WebViewClient (retour)

```
MainActivity.java
```

```
95
96    /*
97     * Activation de l'historique et du bouton retour.
98     * Lors d'un appui sur le bouton retour, l'utilisateur
99     * reviens au site précédent qu'il a consulté,
100    * si il y en a un, grâce aux méthodes
101    * canGoBack() et goBack() de la webview
102    */
103
104    @Override
105    public boolean onKeyDown(int keyCode, KeyEvent event) {
106        // Vérification de l'évènement Key égal au bouton retour
107        // et si il y a un historique
108        if ((keyCode == KeyEvent.KEYCODE_BACK) && webView.canGoBack()) {
109            webView.goBack();
110            return true;
111        }
112
113        // Si il n'y a pas d'historique le bouton est utilisé comme par défaut
114        return super.onKeyDown(keyCode, event);
    }
```

Ainsi, il faut vérifier si la valeur de l'événement Key correspond au bouton retour et si un historique existe grâce à la méthode canGoBack(). Si c'est le cas, nous pouvons appliquer la méthode goBack sur notre WebView. Si ce n'est pas le cas, le bouton retour de la plateforme mobile est utilisé avec son comportement par défaut.

Un navigateur sur lequel on ne peut pas revenir sur les pages précédentes n'est pas réellement un navigateur... Pour activer l'historique et la possibilité de revenir à la page précédente, redéfinissons la méthode onKeyDown permettant de prendre en compte le retour à la page précédente grâce à la touche retour de notre plateforme.

4.4 – Notifications

Notification

Pour pouvoir passer un message à l'utilisateur et l'alerter sur certains évènements, mise à jour de données alors qu'aucune activité de ne soit en premier plan, il existe un système de notification. On les retrouve par exemple pour signaler à l'utilisateur un appel manqué, un nouveau SMS, un nouveau mail, etc..

On peut les utiliser pour transmettre une information à l'utilisateur, mais aussi le ramener à une activité particulière s'il effectue un clic sur cette notification. Depuis les versions récentes d'Android, à savoir la version 4.1, l'utilisateur peut dorénavant interagir directement sur certaines actions à partir de la notification, mais aussi avoir plus de détails sur cette dernière.

Pour mettre en œuvre ce mécanisme, il est nécessaire d'utiliser le gestionnaire de notification, ce dernier permettant de récupérer une instance du service de notification.

Il ne reste plus ensuite qu'à construire la dite notification avec les aspects visuels souhaités grâce au Notification Builder (titre, contenu, image, vibration du téléphone, allumage de la diode, etc..).

Notification - Exemple

Notifions.. Dans l'exemple présenté ici, nous allons préparer et afficher une notification. Considérons que la méthode ci-dessous se lance dès que vous le souhaitez, après une action, un traitement particulier, un appui sur un bouton, etc.. La première étape consiste à récupérer le service de notification inclus sur notre plateforme.

```
protected void launchNotification(){

    //Récupération du gestionnaire de notification à l'aide de getSystemService
    //Le getSystemService prend en paramètre le nom du service que l'on veux récupérer
    final NotificationManager notifManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

    final Intent notifIntent = new Intent(this, MainActivity.class);

    /*PendingIntent appelé lors d'un clic utilisateur sur la notification.
     Différents FLAG permettent de préciser le comportement de l'intent
     Ici, FLAG_ONE_SHOT signifie que cet intent ne sera utilisé qu'une seule fois.
     Il est nécessaire de faire un PendingIntent pour donner la possibilité au gestionnaire
     de notification de lancer un intent particulier au sein de notre activité/application.
     Sinon, il n'aurait pas les droits nécessaires.
    */
    final PendingIntent notificationPendingIntent = PendingIntent.getActivity(this,
        1, notifIntent,
        PendingIntent.FLAG_ONE_SHOT);
}
```

Notification - Exemple

```

//Instanciation d'un builder de notification
Notification.Builder notifBuilder = new Notification.Builder(this);

//Heure de lancement de la notification (ici l'heure en cours)
notifBuilder.setWhen(System.currentTimeMillis());
//Titre de la notification
notifBuilder.setContentTitle(getResources().getString(R.string.titre_notification));
//Contenu de la notification
notifBuilder.setContentText(getResources().getString(R.string.contenu_notification));
//Titre du lancement de la notification
notifBuilder.setTicker(getResources().getString(R.string.titre_lancement_notif));
//Icon de la notification
notifBuilder.setSmallIcon(R.drawable.phone_icon);
//Vibration à la notification
//Vibration(ms), Attente (ms), vibration(ms)...
long pattern[] = {500,500,500,500,500,500};
notifBuilder.setVibrate(pattern);
//Clignotement de la led
notifBuilder.setLights(Color.RED, 1, 0);

//Ajout de l'intent à lancer lors d'un clic sur la notification
notifBuilder.setContentIntent(notificationPendingIntent);

//Lancement de la notification (dépend de l'heure fixée au dessus..)
notifManager.notify(notifId, notifBuilder.build());
}

```

FLAG



FLAG_CANCEL_CURRENT

L'intent précédent est détruit avant d'en recréer un nouveau

FLAG_ONE_SHOT

L'intent ne sera lancé qu'une seule fois

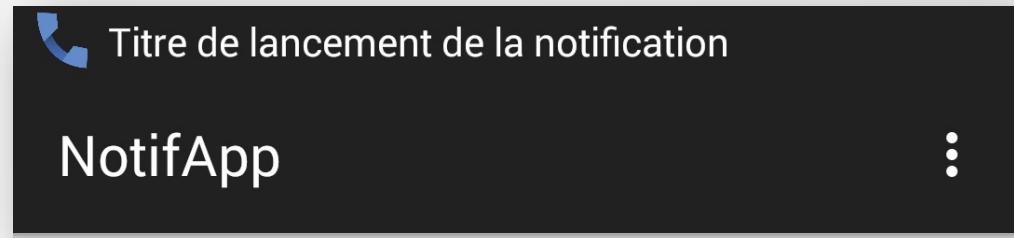
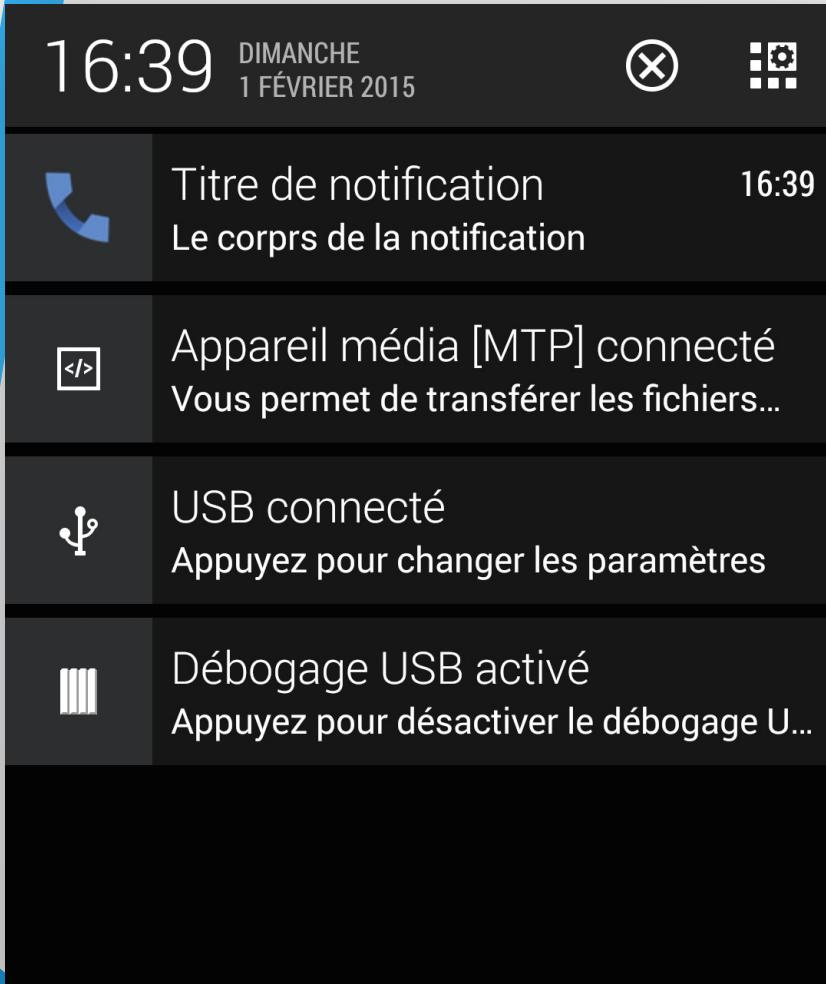
FLAG_UPDATE_CURRENT

L'intent existant est mis à jour

FLAG_NO_CREATE

Si pas d'intent existant, aucune création

Notification - Exemple



Il est possible de redéfinir votre propre vue utilisateur de notification, grâce à la classe `RemoteViews`.

Depuis la version Android 4.1, les notifications sont personnalisables avec une plage de texte et une image de contenu plus grande.

Pour les plus curieux :

<https://developer.android.com/guide/topics/ui/notifiers/notifications.html>



**Fin du chapitre
Architecture
&
Développement**

Chapitre N°5

Notions de sécurité

Notion de sécurité

- 1 - Globalités
- 2 - Permissions
- 3 - Stockage et cryptage
- 4 - Validation des entrées
- 5 - ASLR
- 6 – Pour aller plus loin

Globalités

Un certain nombre de fonctionnalités sont déjà intégrées au sein du système Android afin de réduire l'impact et la fréquence des problèmes de sécurité qui peuvent survenir :

- Application Android SandBox : Isolation des données et de l'exécution des autres applications
- Framework applicatif robuste : Cryptographie, Permissions, IPC (InterProcess Communication).
- Gestion sécuritaire de la mémoire : ASLR (Address space layout randomization)
- Système de fichiers chiffrés
- Autorisation utilisateur pour restreindre l'accès aux fonctions du système et aux données.

Néanmoins, il est nécessaire d'avoir une certaine vigilance dans la construction des applications Android.

Permissions

Les permissions sont des mécanismes applicatifs qui permettent de restreindre l'accès aux composants.

Elles permettent notamment d'empêcher des applications mal intentionnées de corrompre des données, d'accéder aux informations sensibles mais également de faire un usage excessif ou non autorisé des ressources matérielles et des canaux de communication externes.



Permissions (normales)

Il est recommandé d'utiliser au minimum les demandes de permissions afin de rendre les applications moins vulnérables aux utilisations abusives et aux attaquants.

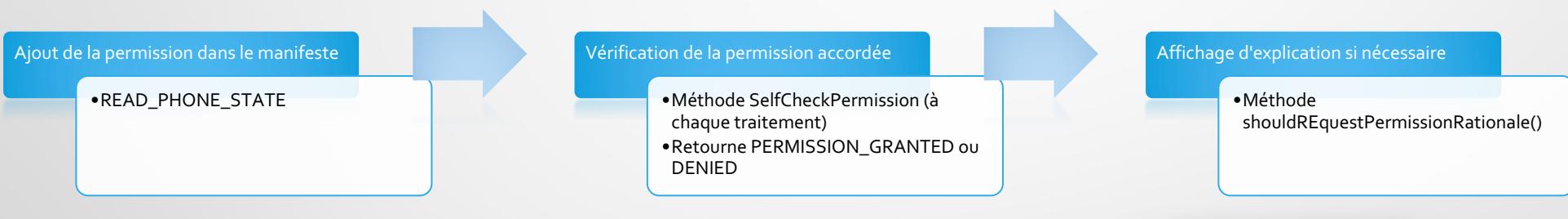
Les permissions sont accordées (ou refusées) lors de l'installation d'un package, une fois cette étape passée, l'utilisateur ne sera plus invité à accorder ces dernières... De même, lors de la définition de vos permissions, il est important de bien définir le niveau de protection (<https://developer.android.com/guide/topics/manifest/permission-element.html>)

```
<permission  
    android:name="myapp.app.permission.MAPS_RECEIVE"  
    android:protectionLevel="signature" />  
  
<uses-permission android:name="myapp.app.permission.MAPS_RECEIVE" />  
<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="com.google.android.providers.gsf.permission.READ_GSERVICES" />  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Permissions sensibles

Depuis la version 6 (API 23), il existe deux catégories de permissions :

- Permissions normales : Déclaration dans le manifeste car elle ne concerne pas la vie privée de l'utilisateur ni ses données personnelles.
- Permissions sensibles : Demande effectuée à l'utilisation de l'application (Runtime) via une popup de confirmation, car elles concernent la vie privée de l'utilisateur (ex : carnet de contact). Obligatoire depuis l'API 23 (targetSdkVersion).



Permissions sensibles - Exemple

```
14 public class MainActivity extends AppCompatActivity {
15
16     public static final int MY_PERMISSIONS_REQUEST_READ_PHONE_STATE = 1;
17     TelephonyManager telephoneManager;
18
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.activity_main);
23
24         //Required READ_PHONE_STATE permission in manifest //!
25         telephoneManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
26
27         /**
28          * getPhoneType() | 0 = PHONE_TYPE_NONE, 1 = PHONE_TYPE_GSM, 2 = PHONE_TYPE_CDMA, 3 = PHONE_TYPE_SIP
29          */
30         int phoneType = telephoneManager.getPhoneType();
31         Log.v(MainActivity.this.getClass().getName(), msg: "phoneType = " + phoneType);
32
33         /**
34          * getDataState()
35          */
36         int dataState = telephoneManager.getDataState();
37         Log.v(MainActivity.this.getClass().getName(), msg: "dataState = " + dataState);
38
39         /**
40          * getSimOperator() and getNetworkOperator()
41          */
42         String operator = telephoneManager.getSimOperator();
43         Log.v(MainActivity.this.getClass().getName(), msg: "Operator = " + operator);
44
45         String networkOperator = telephoneManager.getNetworkOperator();
46         int mcc = 0, mnc = 0;
47         if (networkOperator != null) {
48             mcc = Integer.parseInt(networkOperator.substring(0, 3));
49             mnc = Integer.parseInt(networkOperator.substring(3));
50         }
51         Log.v(MainActivity.this.getClass().getName(), msg: "NetworkOperator = " + networkOperator);
52
53         //Verified permission demand state
54         if (ContextCompat.checkSelfPermission(context: MainActivity.this, Manifest.permission.READ_PHONE_STATE) != PackageManager.PERMISSION_GRANTED) {
55
56             // Should we show an explanation?
57             if (ActivityCompat.shouldShowRequestPermissionRationale(activity: MainActivity.this, Manifest.permission.READ_PHONE_STATE)) {
58                 // Show an explanation to the user *asynchronously* — don't block
59                 // this thread waiting for the user's response! After the user
60                 // sees the explanation, try again to request the permission.
61
62             } else {
63                 // No explanation needed, we can request the permission.
64                 ActivityCompat.requestPermissions(activity: MainActivity.this, new String[]{Manifest.permission.READ_PHONE_STATE}, MY_PERMISSIONS_REQUEST_READ_PHONE_STATE);
65                 // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
66                 // app-defined int constant. The callback method gets the
67                 // result of the request.
68             }
69         } else{ //We have already permission, we can get informations
70             getPermittedInfos();
71         }
72     }
73 }
```

Permissions sensibles - Exemple

```
77     /**
78      * On return validation by user
79      * @param requestCode
80      * @param permissions
81      * @param grantResults
82      */
83     @Override
84     public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {
85         switch (requestCode) {
86             case MY_PERMISSIONS_REQUEST_READ_PHONE_STATE: {
87                 // If request is cancelled, the result arrays are empty.
88                 if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
89                     // permission was granted, yay! Do the
90                     // contacts-related task you need to do.
91                     getPermittedInfos();
92                 } else {
93
94                     Log.v(MainActivity.this.getClass().getName(), msg: "ERR-PERM : Perm Non Accepted");
95                     // permission denied, boo! Disable the
96                     // functionality that depends on this permission.
97                 }
98             }
99             return;
100        }
101    }
102
103    // other 'case' lines to check for other
104    // permissions this app might request
105 }
```

Permissions sensibles - Exemple

```
108     /**
109      * Get Phone information
110     */
111    protected void getPermittedInfos(){
112
113        String idPhone = this.telephoneManager.getDeviceId();
114        Log.v(MainActivity.this.getClass().getName(), msg: "idPhone = "+ idPhone);
115
116        String countryISO = telephoneManager.getSimCountryIso();
117        Log.v(MainActivity.this.getClass().getName(), msg: "countryISO = "+ countryISO);
118
119        String operatorName = telephoneManager.getSimOperatorName();
120        Log.v(MainActivity.this.getClass().getName(), msg: "operatorName = "+ operatorName);
121
122        int simState = telephoneManager.getSimState();
123        Log.v(MainActivity.this.getClass().getName(), msg: "simState = "+ simState);
124
125        String voicemailNumer = telephoneManager.getVoiceMailNumber();
126        Log.v(MainActivity.this.getClass().getName(), msg: "voicemailNumer = "+ voicemailNumer);
127
128        String voicemailAlphaTag = telephoneManager.getVoiceMailAlphaTag();
129        Log.v(MainActivity.this.getClass().getName(), msg: "voicemailAlphaTag = "+ voicemailAlphaTag);
130
131        // Getting connected network iso country code
132        String networkCountry = telephoneManager.getNetworkCountryIso();
133        Log.v(MainActivity.this.getClass().getName(), msg: "networkCountry = "+ networkCountry);
134
135        // Getting the connected network operator ID
136        String networkOperatorId = telephoneManager.getNetworkOperator();
137        Log.v(MainActivity.this.getClass().getName(), msg: "networkOperatorId = "+ networkOperatorId);
138
139        // Getting the connected network operator name
140        String networkName = telephoneManager.getNetworkOperatorName();
141        Log.v(MainActivity.this.getClass().getName(), msg: "networkName = "+ networkName);
142
143        int networkType = telephoneManager.getNetworkType();
144        Log.v(MainActivity.this.getClass().getName(), msg: "networkType = "+ networkType);
145
146
147    }
```

Cryptage et Stockage externe

Quand bien même il est possible de limiter l'accès aux données du stockage (interne), pour garantir une sécurité maximale il est possible de chiffrer les fichiers locaux depuis Android 5.0. Les applications n'ont pas accès aux clés de chiffrement.

<https://source.android.com/security/encryption/>

Le stockage externe quant à lui est accessible en lecture et en écriture d'une manière globale. Il est vivement déconseillé de stocker des données sensibles sur le stockage externe. En effet, les cartes SD peuvent être enlevées du téléphone pour être utilisées sur d'autres machines.



Stockage - Interne

Il existe trois manières d'enregistrer des données sur les plateformes Android : interne, externe et via un fournisseur de contenu.

Concernant le stockage interne, chaque application dispose de son répertoire. Cependant chacune d'entre elle peut partager ses fichiers grâce aux permissions (MODE_WORLD_READABLE et MODE_WORLD_WRITABLE). Cette utilisation est dépréciée depuis l'API 17.

Par défaut le mode de création des fichiers est PRIVATE. Ainsi les fichiers ne sont accessibles que par l'application qui les a créer.

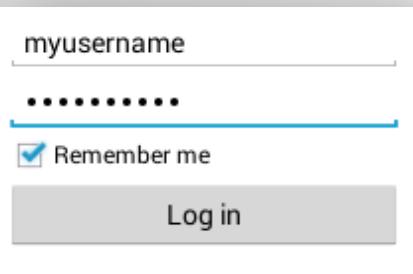
<https://developer.android.com/guide/topics/manifest/permission-element.html>

Stockage - Exemple

Prenons un exemple classique, la mémorisation des informations de connexion d'un utilisateur.

L'enregistrement des données de connexion dans les préférences va créer un dossier « shared_prefs » au sein du package de l'application. Ce dernier contiendra un fichier « userdetails.xml » contenant les informations de connexion de l'utilisateur en clair...

Un paramétrage des données en MODE_WORLD_READABLE pourrait être catastrophique...



```
public void savePreferences(String login, String password){  
  
    SharedPreferences sharedPref = this.getSharedPreferences(Context.MODE_PRIVATE);  
    SharedPreferences.Editor editor = sharedPref.edit(),  
    editor.putString("login", login);  
    editor.putString("password", password);  
    editor.putBoolean("remember", true);  
    editor.commit();  
}
```

Validation des entrées

Android n'échappe pas aux règles qui s'appliquent sur tous développements d'applications, notamment concernant les données en entrées. Qu'elles soient saisies par l'utilisateur, récupérées par le réseau ou par une autre application, les données doivent être vérifiées pour limiter au maximum les risques.

Une mauvaise analyse et validation des données expose les applications aux problèmes classiques tels que les dépassemens de tampon mémoire (Buffer Overflow) ou les injections SQL.

Même si le système a été conçu avec des technologies qui permettent de limiter les risques (ASLR) cela n'est pas suffisant !

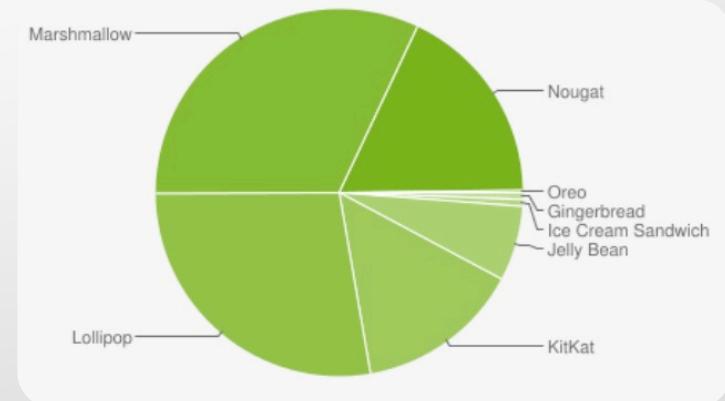
Première implémentation sur Android 4.4, non fiable. Déploiement à partir d'Android 5.0, théoriquement plus fiable.

ASLR

L'ASLR (Address Space Layer Randomization) est un mécanisme permettant de rendre aléatoires les adresses d'allocation mémoire des modules exécutable pour chaque exécution (Tas, Pile, Libs). L'objectif étant principalement d'éviter les attaques de type « Buffer Overflow ».

2015 : Exploitation de la faille **Stagefright** par J. Drake, 95 % du parc de terminaux sous Android exposé...

La bibliothèque **Stagefright** permet de lire les fichiers multimédias comme les vidéos et les fichiers audio. Son exploitation donner un accès root sur la machine avec un seul fichier vidéo... (Testé sur Android 2.3, 4.0, 5.0, 5.1)



Répartition des versions Android sur les terminaux mondiaux (2017)

Source : <http://www.numerama.com/tech/132165-les-versions-dandroid-les-plus-utilisees.html>

ASLR

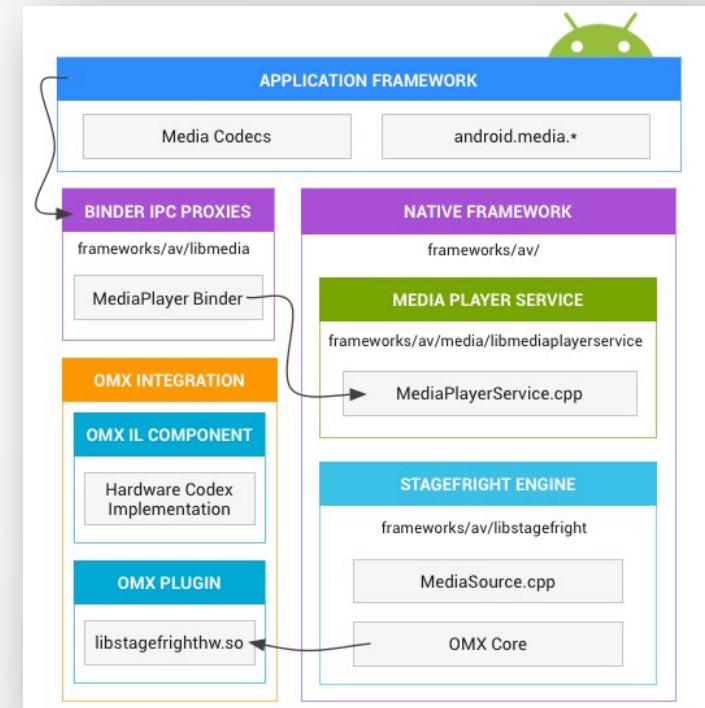
L'ASLR complique donc la tâche des personnes malveillantes, mais il est tout de même possible de le contourner.

Moyen d'attaque :

- MMS (Fichier traité à la réception .. !)
<https://www.youtube.com/watch?v=PxQc5gOHnKs>
- Application de chat (Whatsapp), browser..

Contre-mesure :

- Désactivation du chargement auto des MMS
- Désactivation du chargement auto des vidéos
- Stagefright Detector App
<https://blog.zimperium.com/stagefright-vulnerability-details-stagefright-detector-tool-released/>



http://www.fandroid.com/android/applications/securite-applications/301840_stagefright-details-de-lattaque

Metaphor - Stagefright Exploitation

<https://www.youtube.com/watch?v=l507kD0zG6k>

Metaphor technical explanation (Stagefright exploit) :

<https://github.com/NorthBit/Public/raw/master/NorthBit-Metaphor.pdf>

<https://www.youtube.com/watch?v=71YP65UANP0>

Un début...

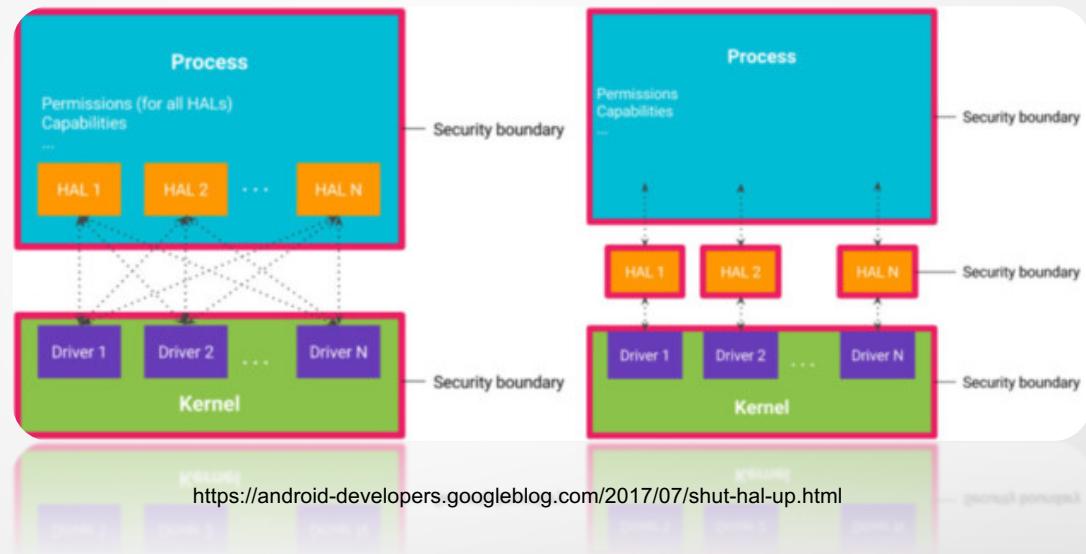
Android 8.0 amène plus de sécurité... Notamment avec le projet Treble, MAJ plus rapides, Système plus sécurisé (Fragmentation du serveur média), Play Protect, Boot 2.0 vérifié

So, what's new?

- Google Play Protect
- Verified Boot 2.0
- Improved Isolation
- Key Attestation
- Fine-Grained Install Sources
- FBE Encryption Enhancements
- Treble
- FIDO U2F Security Key Support
- Verify Apps API
- Lockscreen Enhancements
- Isolated Webview
- And more...
- reCAPTCHA API
- Updated Permissions Model
- Seccomp by-default

#io17

http://www.frandroid.com/android/mises-a-jour-android/454167_3-nouveautés-qui-comptent-sur-android-8-0-oreo



Mais... La nouvelle fonctionnalité « Autofill » permettant le remplissage automatique des champs de saisie n'est pas sécurisée → Stockage des données non partitionnées.

Le remplissage de tous les champs présents sur la page se fait en même temps.. Y compris les champs invisibles. Une demande du login/mdp peut ainsi permettre une récupération des autres données, comme le N° CB enregistré.

Contre-mesure : Désactivation de la fonctionnalité.

Pour aller plus loin (TP)

La sécurisation des applications Android nécessite une recherche et une remise en question constante. Voici quelques thèmes à approfondir pour aller plus loin dans cette quête de sécurité. Cette liste étant évidemment non exhaustive :

- Webview (Faille XSS)
- Race Condition
- Désassemblage/Décompilation
- Appels et paiements non autorisés
- Chargement dynamique de code



Documentation officielle concernant la sécurité Android :

<https://source.android.com/security/index.html>

<https://developer.android.com/training/articles/security-tips.html>

Outils, références

- **Documentation officielle** : <https://developer.android.com>
- **SDK** : <https://developer.android.com/sdk>
- **Android Studio** :
<https://developer.android.com/sdk/installing/studio.html>
- **Services Google** :
<https://developer.android.com/google/index.html>
- **Security Tips** :
<https://developer.android.com/training/articles/security-tips.html>
- **CodeLabs** :
<https://codelabs.developers.google.com/>

Bibliographie

Une liste, non exhaustive, d'ouvrage vous est présentée ici. Il est fortement conseillé de se tenir à jour des évolutions du système Android, relativement rapides et fréquentes, soit par l'intermédiaire d'ouvrages comme ci-après et/ou grâce à la documentation officielle en ligne, mentionnée sur la diapositive (outils & références).

- **Android 4 – Développement d'applications avancées.** Reto Meier – Ed. Pearson – 2012. ISBN-13 : 978-2744025440 (1)
- **Android 4 – Les fondamentaux du développement d'applications Java.** Nazim Benbourahla – Ed. ENI – 2012. ISBN-13 : 978-2746075603 (2)
- **Google Android 4 efficace : Utilisation avancée des smartphones et tablettes Android.** Arnaud Faque. Ed. Eyrolles – 2013. ISBN-13 : 978-2212137217 (3)
- **Android 7 – Les fondamentaux du développement d'applications Java.** Nazim Benbourahla – Ed. ENI – 2017. ISBN-13 : 978-2409005947 (4)
- **Android Security : Attacks and defenses.** Anmol Misra, Abhishek Dubey – Auerbach Publications . ISBN-13 : 978-1439896464 (5)
- **Android Security Cookbook.** Keith Makan, Scott Alexander-Brown – Packt Publishing Limited. ISBN-13 : 978-1782167167 (6)