



Une petite introduction à Google Android

1. Le contexte de développement



Claude Vilvens / Christophe Charlet

2023



Agenda

1. Un framework pseudo-Java pour mobiles ([p.4](#))
2. Le modèle en couches d'Android ([p.8](#))
3. La machine virtuelle Dalvik ([p.16](#))
4. Le fichier de distribution apk ([p.23](#))
5. Les composants applicatifs Android ([p.25](#))
6. Le cycle de vie d'une activité Android ([p.31](#))
7. Une application Android de base ([p.34](#))
8. L'installation de l'environnement Android ([p.36](#))

1. Un framework pseudo-Java pour mobiles (1/5)

Android est un framework combinant un système d'exploitation pour mobile et un SDK permettant de développer des applications pour tout mobile géré par cet OS.

On retire tout ce qui n'a aucune raison d'être mais on ajoute d'autres choses: Alarme/SMS, ...

- ◆ utilise une version modifiée du kernel **Linux** (≠ d'une distribution de Linux);
- ◆ développé dès 2007 par Android Inc. puis racheté par Google, actuellement sous le contrôle de la **OHA** (**O**pen **H**andset **A**lliance) [consortium d'entreprises : Bouygues, Alcatel, Dell, Acer, Samsung, HTC, Sony Ericsson, Toshiba, Intel, Vodafone, etc];
- ◆ (Historiquement) configuration matérielle : **HTC** Dream like = écran tactile, Wifi et Bluetooth, trackball, clavier coulissant Qwerty, carte SD de 1Gb;
- ◆ logiciel : **Android utilise un Java propriétaire**, non compatible avec J2ME et J2SE, et des librairies natives, construites à nouveau avec des **librairies C non standards** (la librairie **Bionic**).



1. Un framework pseudo-Java pour mobiles (2/5)



Le personnage nommé **Bugdroid** est le petit robot vert utilisé par Google pour présenter Android - il est resté le symbole d'Android, à l'instar de Duke pour le Java standard :



Les différentes versions d'Android sont désignées de trois manières :

◆ par le numéro de version de la plate-forme : 2.3, 3.2, 4.3, 6.0, 9.0, 10, 11, 12, 13... 14 (hier)

◆ par un nom de code : Cupcakes, Donuts, Ice Cream Sandwiches etc ;



+ un même nom peut être conservé pour une nouvelle version, auquel cas on complète le nom de la version précédente par "MR1", "MR2", etc;



◆ par un niveau d'APIs (il s'agit donc en fait de la version du SDK) : 8, 15, 18, 23, ... 31-32 (Android 12), 33 (Android 13)

1. Un framework pseudo-Java pour mobiles (2/5)bis

Il faut donc choisir avant le développement proprement dit:

- Le développement, uniquement pour Android ou pour Iphone (Swift) ?
 - Pour Android, quelle version ? API ?
 - Pour Iphone, il faut une licence et avant le développement (complètement différent d'Android), il faut commencer par créer un storyboard (succession des différents écrans avec un code généré en arrière plan).
- Plateforme de développement par exemple : **Android Studio** ou 
- Utilisation d'un émulateur **AVD (Android Virtual Device)** 

Frameworks mobiles multiplateformes (exemples):

- React Native
- Flutter
- Ionic
- Xamarin
- Unity



1. Un framework pseudo-Java pour mobiles (3/5)



Android 10 :
Google arrête
les friandises
pour clarifier
les noms des
versions



1. Un framework pseudo-Java pour mobiles (4/5)

Android 4	Level 20 Android 4.4W ³	KITKAT_WATCH	KitKat	99.64%	2013	
	Level 19 Android 4.4	KITKAT				
	▪ Google Play services beyond v21.33.56 (the last version despite what the blog states) does not support Android versions below API level 19.					
	Level 18 Android 4.3	JELLY_BEAN_MR2	Jelly Bean	99.68%	2012	
	Level 17 Android 4.2	JELLY_BEAN_MR1		99.71%		
	Level 16 Android 4.1	JELLY_BEAN		99.76%		
	▪ Google Play services beyond v14.8.39 (the last version despite what the blog states) does not support Android versions below API level 16.					
	Level 15 Android 4.0.3 – 4.0.4	ICE_CREAM_SANDWICH_MR1	Ice Cream Sandwich	99.79%	2011	
	Level 14 Android 4.0.1 – 4.0.2	ICE_CREAM_SANDWICH				
	▪ Jetpack/AndroidX libraries require a <code>minSdk</code> of 14 or higher.					
Android 3	Level 13 Android 3.2	HONEYCOMB_MR2	Honeycomb	No data		
	Level 12 Android 3.1	HONEYCOMB_MR1				
	Level 11 Android 3.0	HONEYCOMB				
Android 2	Level 10 Android 2.3.3 – 2.3.7	GINGERBREAD_MR1	Gingerbread			2010
	Level 9 Android 2.3.0 – 2.3.2	GINGERBREAD				
	Level 8 Android 2.2	FROYO	Froyo		2009	
	Level 7 Android 2.1	ECLAIR_MR1	Eclair			
	Level 6 Android 2.0.1	ECLAIR_0_1				
	Level 5 Android 2.0	ECLAIR				
Android 1	Level 4 Android 1.6	DONUT	Donut			2008
	Level 3 Android 1.5	CUPCAKE	Cupcake			
	Level 2 Android 1.1	BASE_1_1	Petit Four			
	Level 1 Android 1.0	BASE	None			

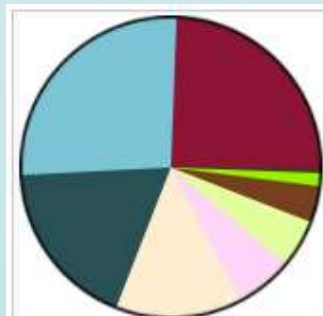
Un framework pseudo-Java pour mobiles

Version	SDK / API level	Version code	Codename	Cumulative usage ¹	Year
Android 14 BETA	Level 34	UPSIDE_DOWN_CAKE	Upside Down Cake	—	TBD
Android 13	Level 33	TIRAMISU	Tiramisu ²	30.33%	2022
	▪ targetSdk will need to be 33+ for new apps and app updates by August 2023.				
Android 12	Level 32 Android 12L	S_V2	Snow Cone ²	50.91%	2021
	Level 31 Android 12	S			
	▪ targetSdk must be 31+ for new apps and app updates.				
Android 11	Level 30	R	Red Velvet Cake ²	70.89%	2020
Android 10	Level 29	Q	Quince Tart ²	80.16%	2019
Android 9	Level 28	P	Pie	87.71%	2018
Android 8	Level 27 Android 8.1	O_MR1	Oreo	90.49%	2017
	Level 26 Android 8.0	O		93.7%	
Android 7	Level 25 Android 7.1	N_MR1	Nougat	94.31%	2016
	Level 24 Android 7.0	N		96.19%	
Android 6	Level 23	M	Marshmallow	97.83%	2015
Android 5	Level 22 Android 5.1	LOLLIPOP_MR1	Lollipop	98.81%	2015
	Level 21 Android 5.0	LOLLIPOP, L		99.34%	2014
	▪ Jetpack Compose requires a minSdk of 21 or higher. ▪ Google Play services will not support Android versions below API level 21 starting in August 2023.				
Android 4	Level 20 Android 4.4W ³	KITKAT_WATCH	KitKat	99.64%	

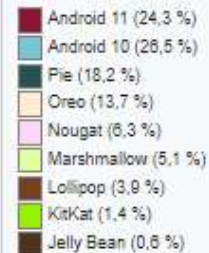
Un framework pseudo-Java pour mobiles



- Android 10 (API level 29)
- Android 11 (API level 30)
- Android 12 (API level 31-2)
- Android 13 (API level 33)



Graphique illustrant la part de chaque version d'Android sur le Google Play Store.

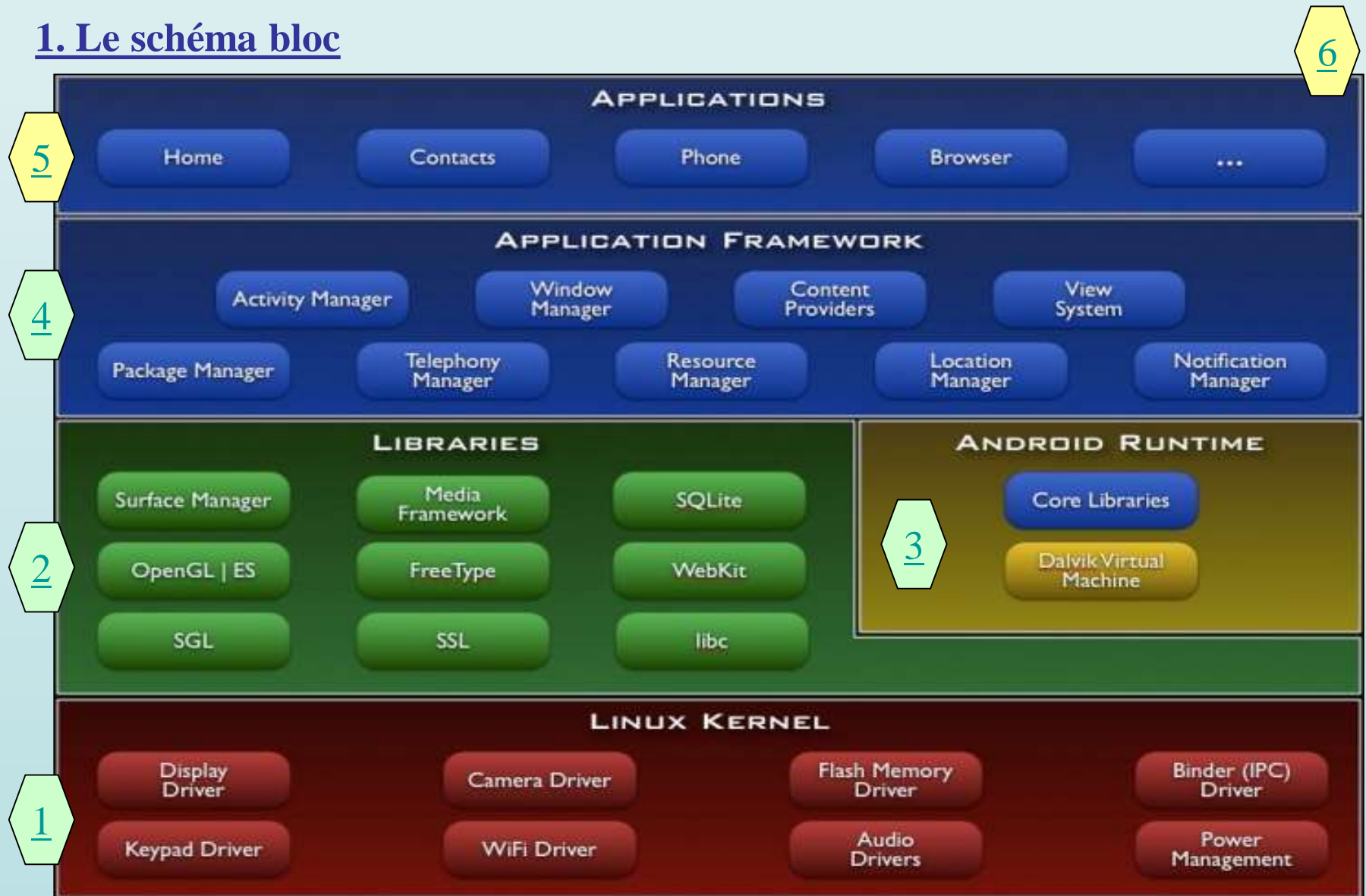


Version ↕	Nom de code ↕	Date de sortie ↕	Version API ↕	% ↕
4.1.x	Jelly Bean	9 juillet 2012	16	0,2 %
4.2.x		13 novembre 2012	17	0,3 %
4.3		24 juillet 2013	18	0,1 %
4.4	KitKat	31 octobre 2013	19	1,4 %
5.0	Lollipop	3 novembre 2014	21	0,7 %
5.1		9 mars 2015	22	3,2 %
6.0	Marshmallow	8 octobre 2015	23	5,1 %
7.0	Nougat	22 août 2016	24	3,4 %
7.1		4 octobre 2016	25	2,9 %
8.0	Oreo	21 août 2017	26	4 %
8.1		6 décembre 2017	27	9,7 %
9	Pie	1 ^{er} décembre 2018	28	18,2 %
10	Android 10	3 septembre 2019	29	26,5 %
11	Android 11	8 septembre 2020	30	24,3 %

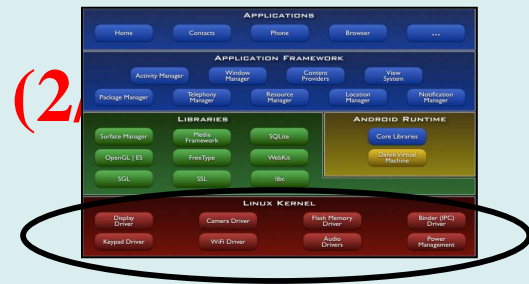
Android 12	Snow Cone	12	31	October 4, 2021
Android 12L	Snow Cone v2	12.1 ^[a]	32	March 7, 2022
Android 13	Tiramisu	13	33	August 15, 2022
Android 14	Upside Down Cake ^[27]	14 ^[b]	34	Q3 2023
Android 15	Vanilla Ice Cream ^[29]	15	TBA	Q3 2024

2. Le modèle en couches d'Android (1/8)

1. Le schéma bloc



2. Le modèle en couches d'Android (2)



2. La couche Linux Kernel

= l'interface d'accès au hardware pour les couches supérieures du modèle.

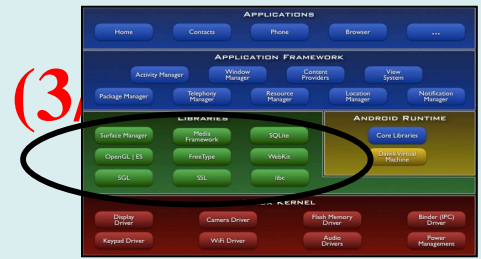
Android utilise une version modifiée du kernel Linux (un 4.x , 5.x ,.. 6.2) qui prend en charge les drivers, les accès aux ressources, l'alimentation électrique : mais certains éléments du kernel Linux traditionnel ont été enlevés et d'autres ont été ajoutés pour permettre son utilisation dans le monde particulier des mobiles ("patches").

"patches"

Prendre le moins de place possible et économiser les ressources afin d'optimiser les performances

- ♦ **Alarm** : pour le réveil de l'appareil quand il est en veille;
- ♦ **Ashmem** : gestion du partage mémoire, nécessaire sur un mobile où l'espace est réduit = une mémoire partagée avec plusieurs ressources
- ♦ **Binder** : module prenant en charge de manière sécurisée les **IPC** permettant une communication entre les différentes applications qui tournent dans des processus différents (= ne pas laisser les développeurs d'applications créer leurs propres IPC car potentiellement dangereux).

2. Le modèle en couches d'Android (3)



3. La couche Librairies

= bibliothèques C/C++ utilisables depuis la couche supérieure (l'application framework).

♦ **Bionic LibC** : une espèce de *glibc revue à la baisse*, pour tenir compte de l'environnement réduit d'un mobile;

♦ **Webkit** : une bibliothèque de fonctions permettant aux logiciels d'afficher les éléments d'une page web - on parle encore de "*moteur de rendu (HTTP)*"; c'est un dérivé du moteur KHTML du projet libre KDE (ensemble de technologies dédiées à un environnement de bureau et de développement d'application dans un contexte multi-plateformes);

♦ **SQLite** : un SGBD "léger" particulièrement adapté au monde des mobiles: Room
= sauvegarde locale (Lecture/Ecriture avec instructions SQL)



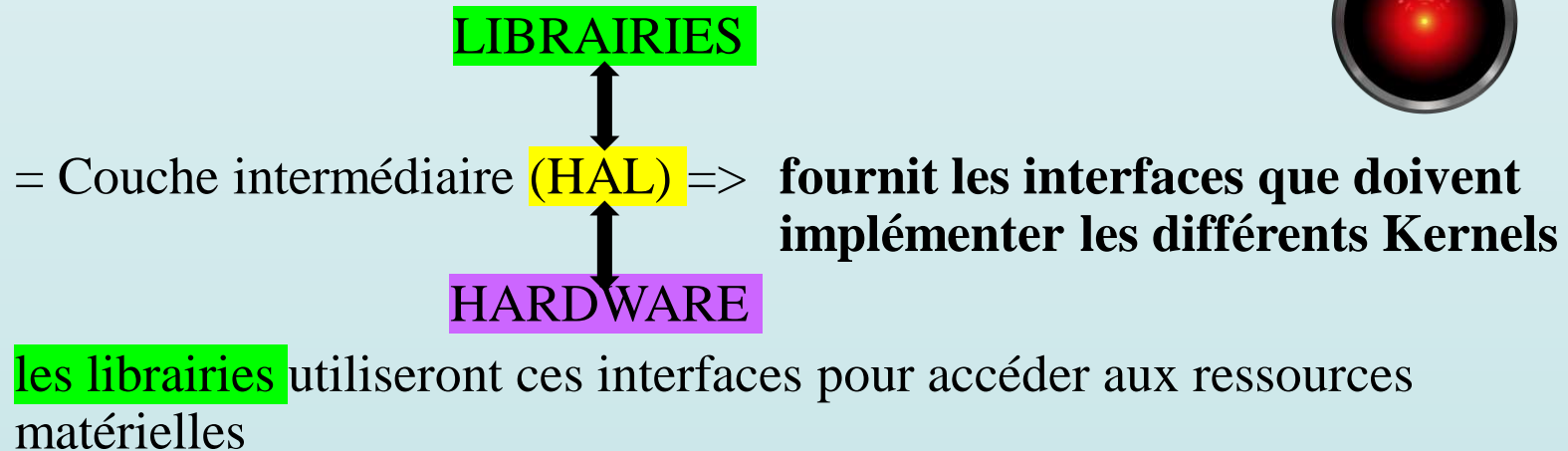
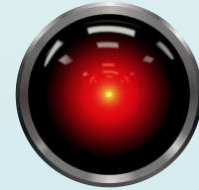
♦ **Surface Flinger** : pour les rendus graphiques en 2d et 3D => partie la plus amusante pour les infographistes.

Le rendu graphique est sous forme d'un fichier XML (+ java Beans => appli Android)

Pour que l'implémentation de ces bibliothèques soit gérable => HAL

2. Le modèle en couches d'Android (3/8 bis)

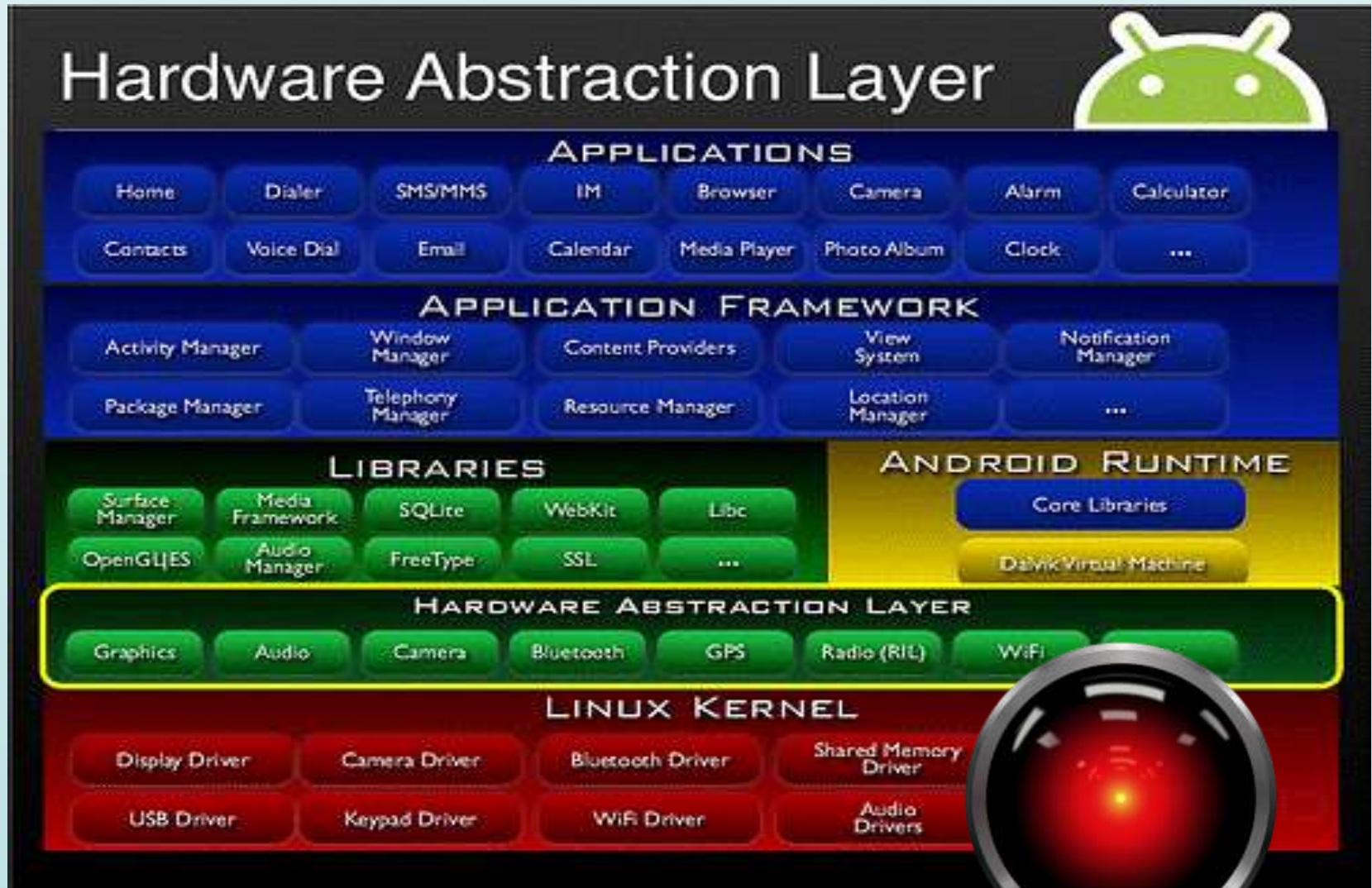
- **Hardware Abstraction Libraries :**



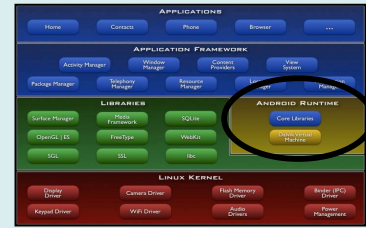
Tous les hardwares de base doivent proposer les mêmes fonctions C. Ce n'est pas aux développeurs d'application de modifier leurs librairies tout simplement parce que la couche hardware du dessous ne possède pas les mêmes APIs.

Tous ceux qui conçoivent un Kernel doivent fournir les différentes APIs comme par exemples pour: le bluetooth, le GPS, ...

2. Le modèle en couches d'Android (4/8)



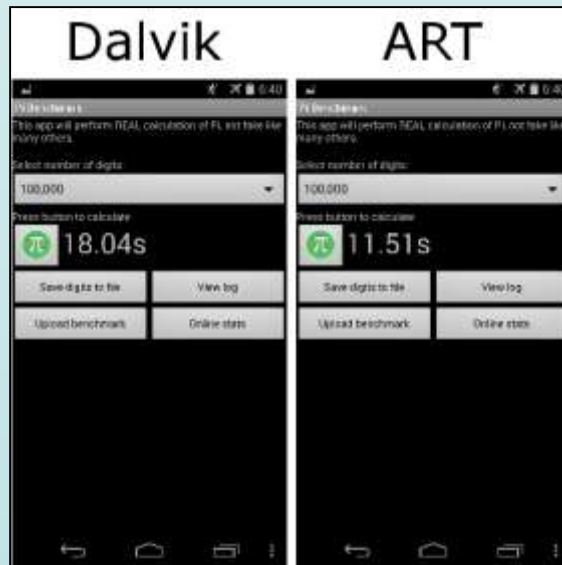
2. Le modèle en couches d'Android (5/8)



4. La couche Java ("Android Runtime")

= la couche développement "abstrait"

- ◆ les **Cores Librairies** : ce sont les libraires Java de base, de type J2SE 1.5 mais **propriétaires** et **différentes** (par exemple, pas de javax.swing, utilisation de fichiers XML);
- ◆ la **machine virtuelle Dalvik** ou **ART**: différente d'une JVM standard



Par exemple:

- Il n'existe pas de JPanels. Ces derniers sont remplacés par des TextView.
- Voir plus loin

Bon à savoir pour le développeur : si une application Java nécessite du code en C/C++, elle pourra le faire par JNI (Java Native Interface) sur du code développé avec le Native Development Kit (**NDK**);

2. Le modèle en cou

En fait, on programme des Activités (activity).

Une activité n'est pas une application

Une application contient en général, plusieurs activités.

Activity manager: permet le passage d'une activité à l'autre

5. La couche Application Framework

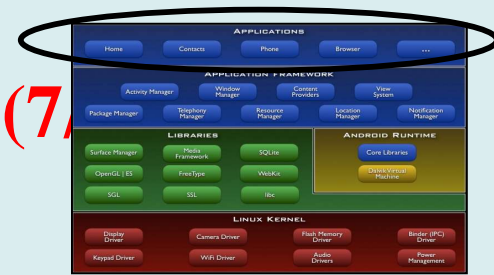
= toolkit que les applications utiliseront
qui tournent en arrière-plan et qui sont

1) les Core Platform Services = services communs à toutes les applications sur le mobile :

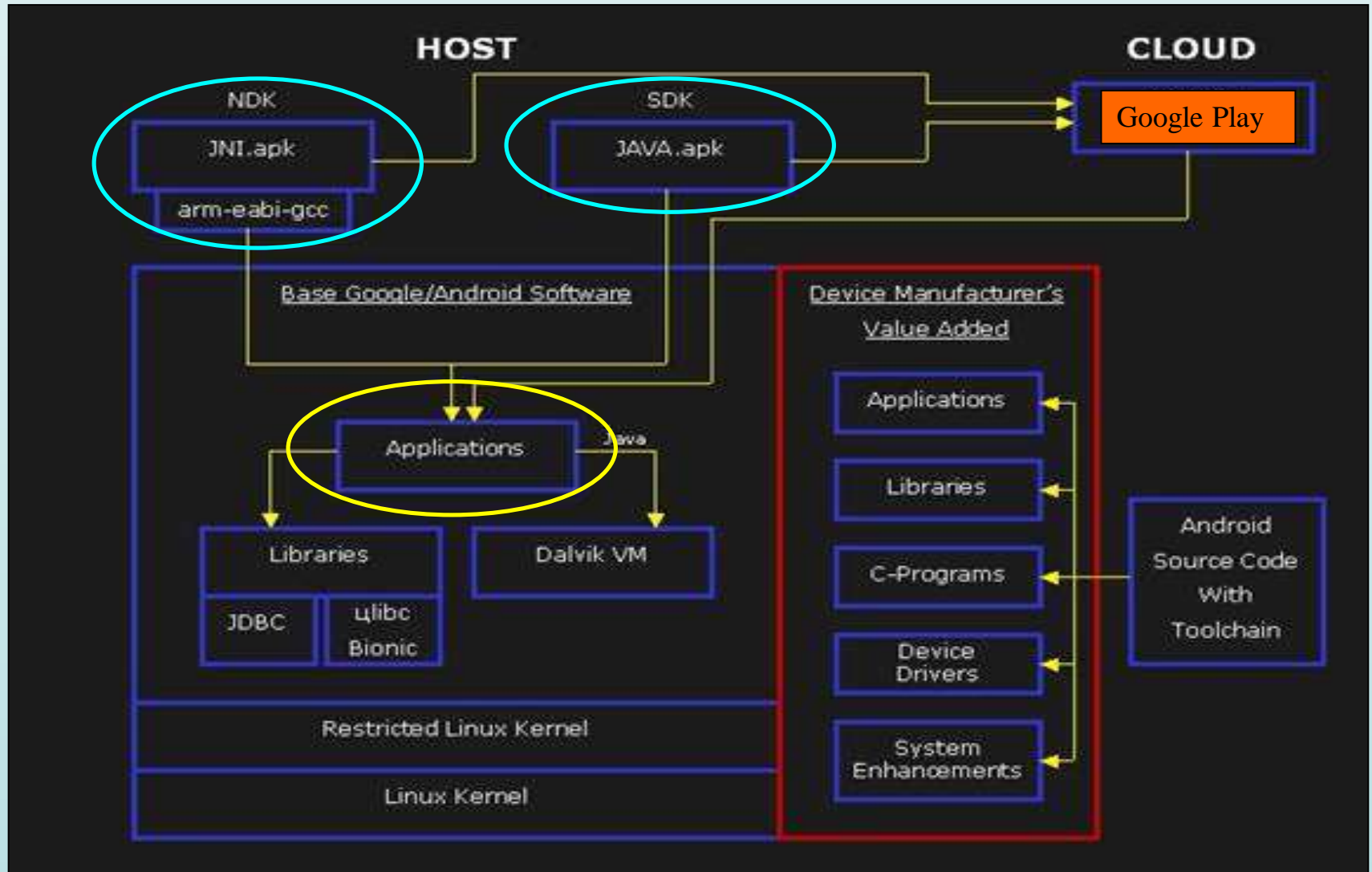
- ♦ **Activity Manager** : gestion du cycle de vie des applications et de la pile de navigation (*pour revenir à l'application précédente quand une application se termine*);
- ♦ **Package Manager** : chargement des fichiers **.apk** (voir plus loin, équivalent du **.jar**)
- ♦ **Window Manager** : au-dessus du Surface Flinger, il gère les recouvrements des fenêtres;
- ♦ **Resource Manager** : partage des ressources;
- ♦ **Content Provider** : gestion de partage de données entre applications (provenant par exemple d'une base de données, de fichiers son/image, ...);
- ♦ **View System** : tous les composants graphiques classiques, comme les listes, grilles, zones d'édition, etc. Définition des éléments graphiques dans des fichiers XML auxquels il faut associer des objets: boutons, list, ...

2) les Hardware Services = APIs vers le matériel = Telephony Service, Location Service (pour le GPS), Bluetooth Service, Wifi Service, USB Service

2. Le modèle en couches d'Android (7)

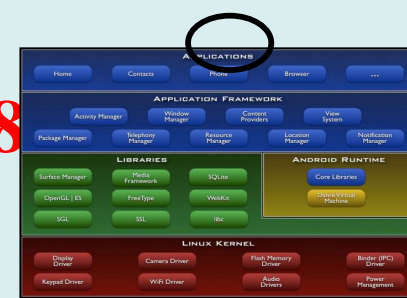


6. Le point de vue du développeur

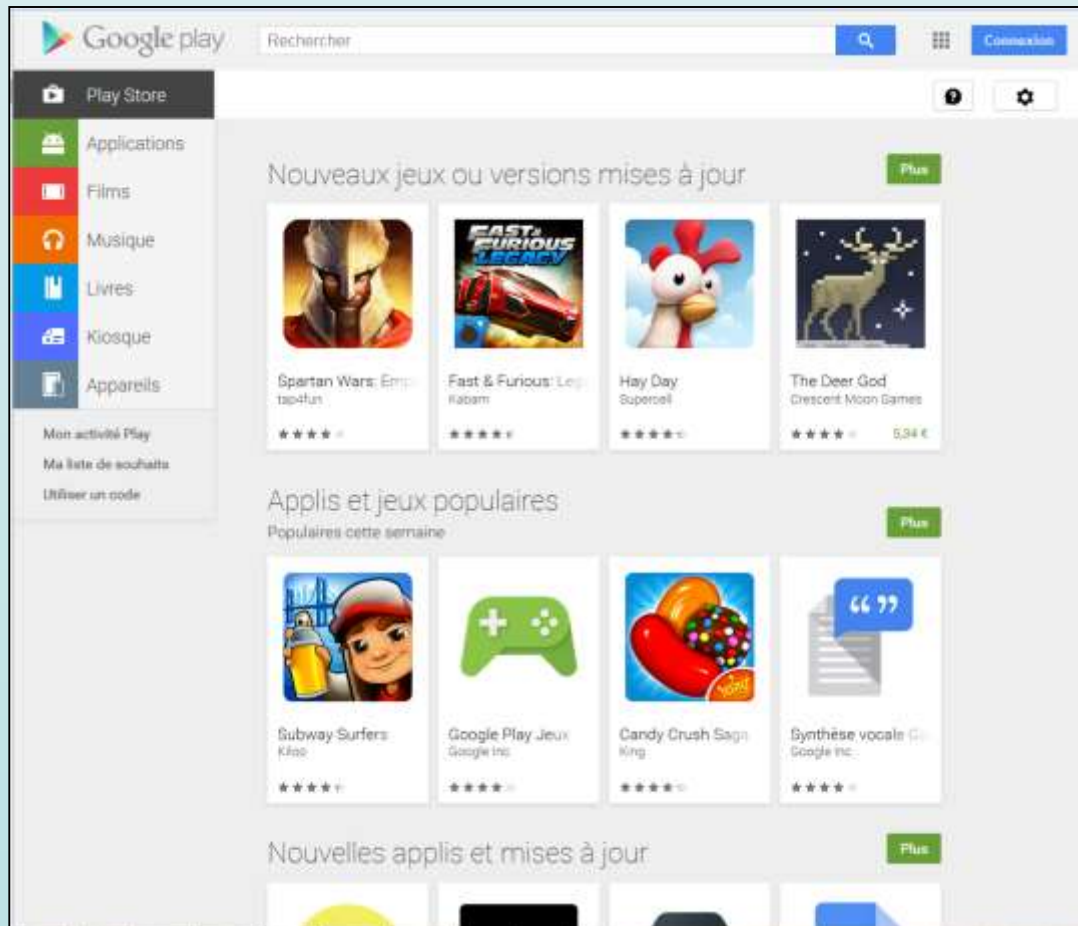


2. Le modèle en couches d'Android (8/8)

7. La couche annexe : Google Play



L'**Android Market** a été depuis rebaptisé **Google play** (<https://play.google.com/store>). on peut s'y fournir des applications gratuites ou payantes (avec une contribution de 30% laissée à Google) proposées par les développeurs Android :



3. La machine virtuelle Dalvik (1/9)

1. Une machine orientée registres

La machine virtuelle d'Android = **Dalvik** - tout à fait particulière : elle possède une **architecture "orientée registres"** [*register-based architecture*], à l'opposé des JVM classiques à **architecture "orientée pile"** [*stack machine*].

La différence entre les deux se marque particulièrement pour les instructions de haut niveau qui manipulent des données et qu'il faut convertir en langage machine:

- ♦ **sur une machine à pile**, le mécanisme automatisé de la pile permet de faire l'économie des adresses dans les instructions machines mais réclame un **nombre plus important d'instructions bas niveau** pour manipuler les données.
- ♦ **sur une machine à registres**, les instructions bas niveau **contiennent directement les adresses des registres** source et destination mais, par le fait même, sont **plus longues**.

☺ Dalvik a été développée notamment pour permettre aux **appareils peu puissants** de **faire tourner plusieurs applications simultanément**.

3. La machine virtuelle Dalvik (1/9)

Architecture "orientée pile" [*stack machine*].

Exemple:  (équivalent à $(a+b).c$)

- J'empile l'opérante (a), suivi de l'opérante (b),
- on arrive sur un opérateur (+) => on dépile, récupération de a et b et on effectue le (+).
- J'empile le résultat obtenu (a+b)
- J'empile l'opérante (c)
- Je vois l'opérateur (*) => on dépile, récupération de c et du résultat (a+b) et on effectue (*)
- Fin (la pile est vide si il n'y a pas d'erreur)

Pas besoin d'adressage (économie d'adresses mais exécution de beaucoup d'instructions)

3. La

Ressemble au « pool des chaînes de caractères constants » utilisé en Java. Même si une chaîne de caractères apparaît plusieurs fois dans le code, elle n'existera **qu'en un seul exemplaire**. Tout le monde pointe donc sur la même chaîne => l'opérateur == peut être appliqué sur les chaînes de caractères (c'est la seule exception, si non on utilise « **equal** » afin de comparer les références de deux objets).

2. Un bytecode

La JVM Dalvik

.class, mais exécute des applications converties en un **fichier .dex (Dalvik EXecutable)**.

Ce format dex est plus particulièrement adapté aux mobiles dotés d'un processeur de faible puissance et/ou d'une mémoire centrale limitée.

Par exemple, *les références multiples sont restructurées en des références uniques à un pool*, supprimant ainsi les redondances et allégeant donc l'ensemble produit.

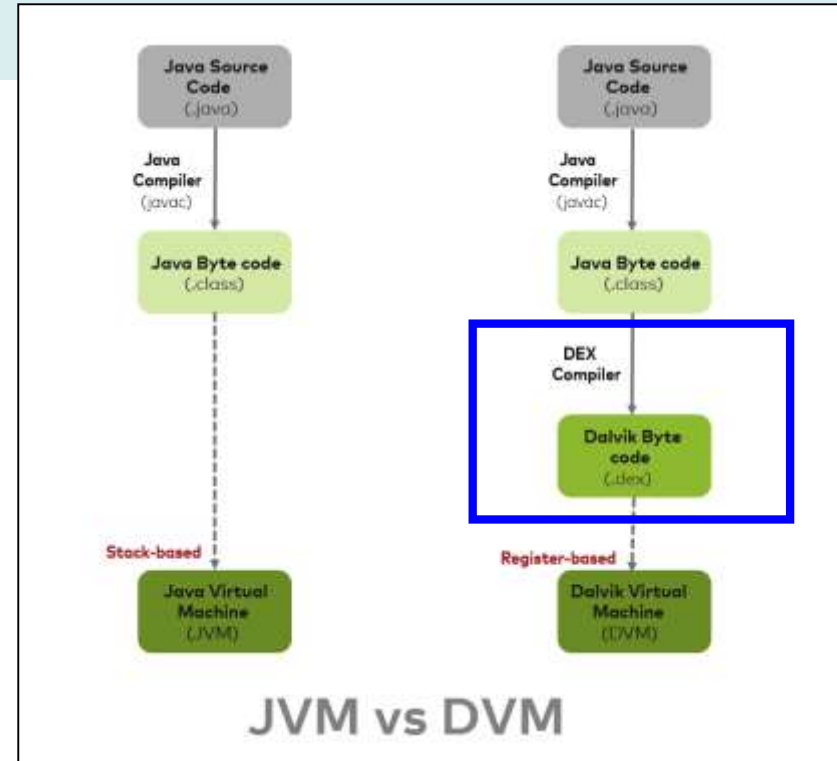
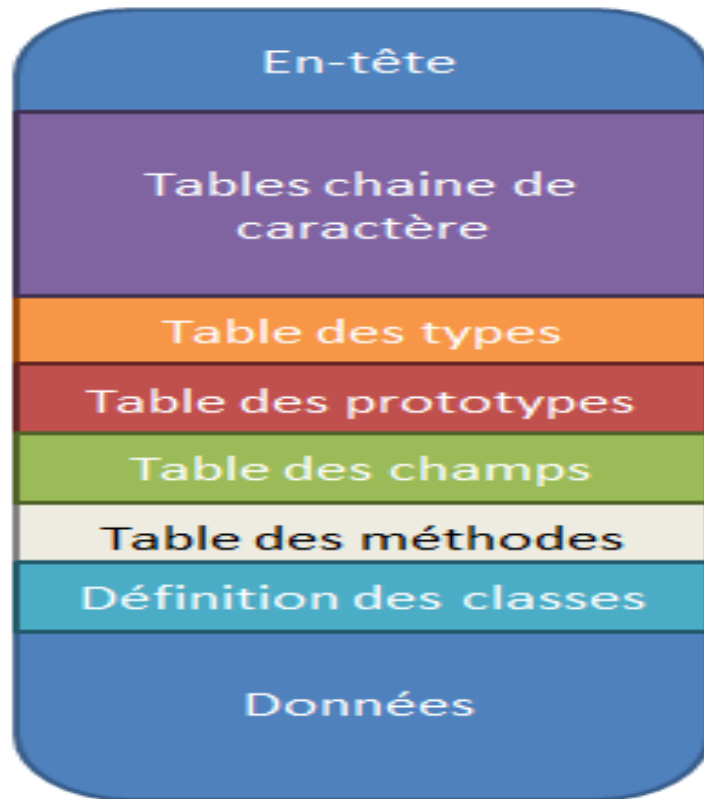
Le bytecode qui sera fabriqué par le compilateur Java classique est donc converti en un nouveau set d'instructions Dalvik (au moyen d'un outil appelé **dx**) : on parle encore d'"**exécutable Dalvik**".

Un tel exécutable peut encore être modifié lors de son installation sur le mobile visé, ceci dans un but d'optimisation (modification du byte order, éviction de classes non utilisées, etc).

Ce "bytecode Dalvik" est comme toujours traduit à la volée (et mis en cache), selon la méthode dite **Just-In-Time** (juste à temps), **à chaque lancement d'une application**.

3. La machine virtuelle Dalvik (3/9)

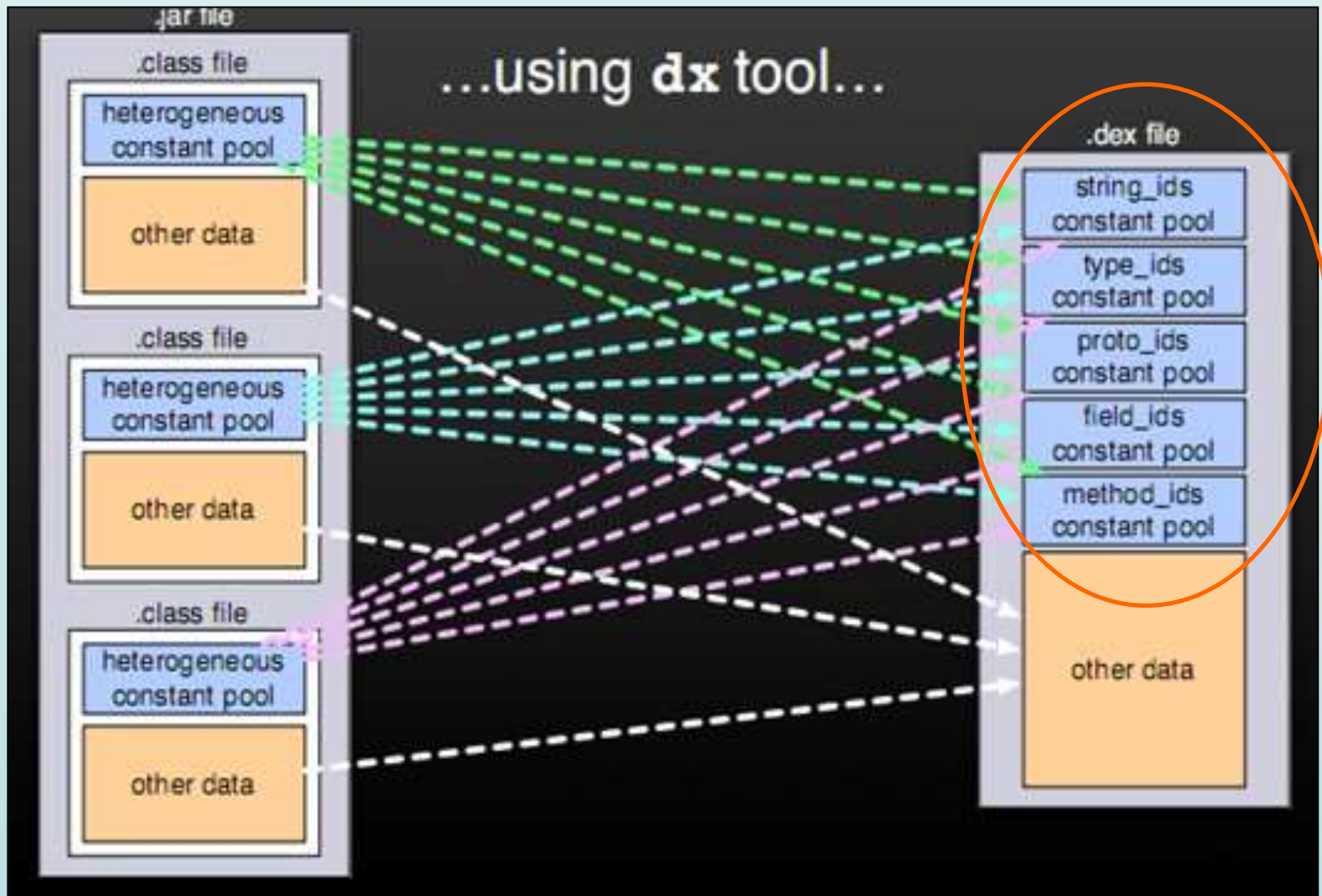
La structure d'un fichier dex :



A comparer avec l'adressage indexé (ASM), l'héritage virtuel (C++) ;-) !

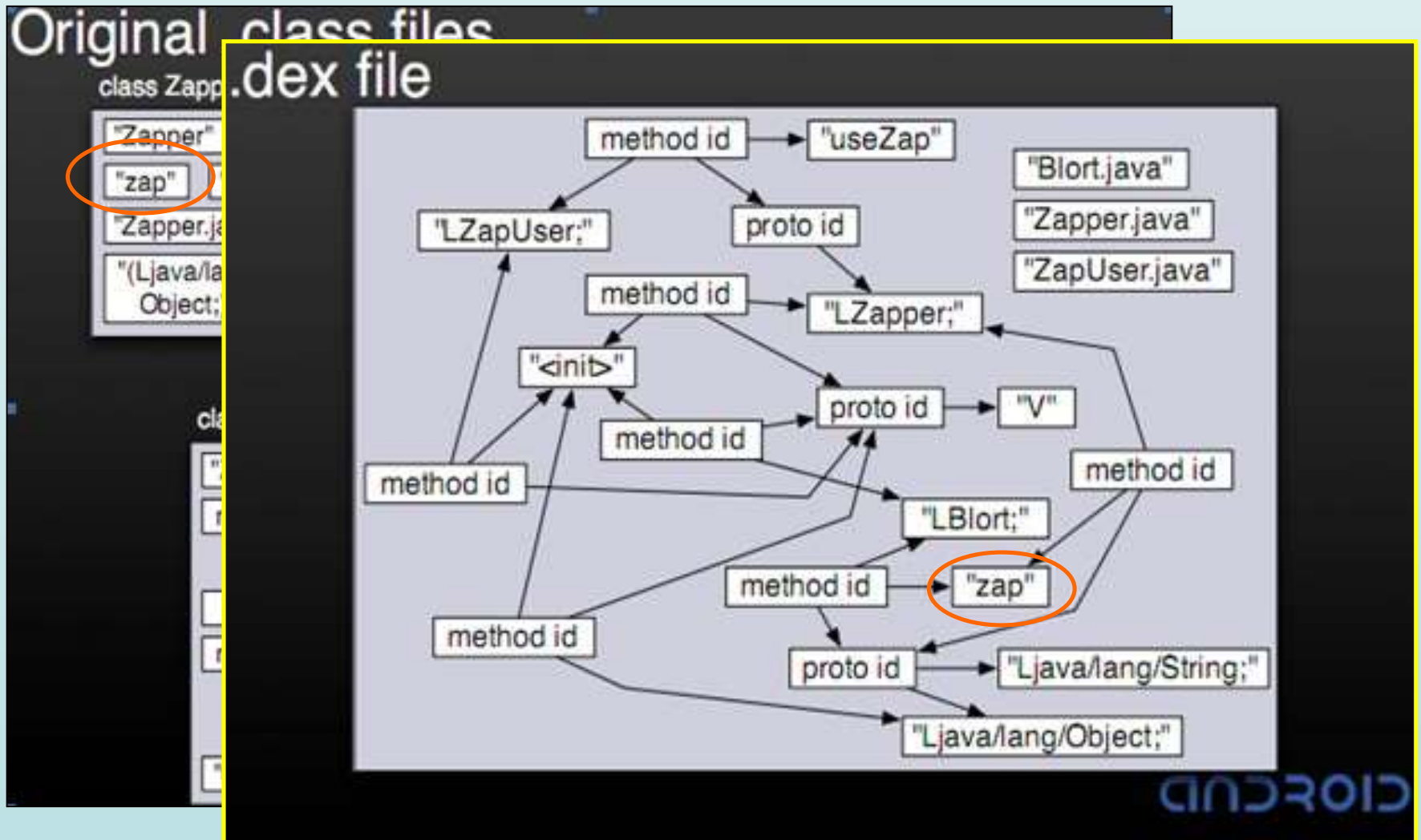
3. La machine virtuelle Dalvik (4/9)

Le processus de transformation :



3. La machine virtuelle Dalvik (5/9)

différence entre un fichier .jar contenant des fichiers .class et un fichier .dex dans un .apk



3. La machine virtuelle Dalvik (6/9)

3. Plusieurs machines virtuelles

Une instance de la machine virtuelle Dalvik est créée pour chaque processus différent tournant sur l'appareil.

Ce système a été choisi afin d'éviter une erreur générale dans le cas du plantage d'une des machines virtuelles.

Mais par ce fait, il importait d'avoir une machine virtuelle beaucoup moins imposante qu'une JVM standard : le fichier dex répond à ce besoin.

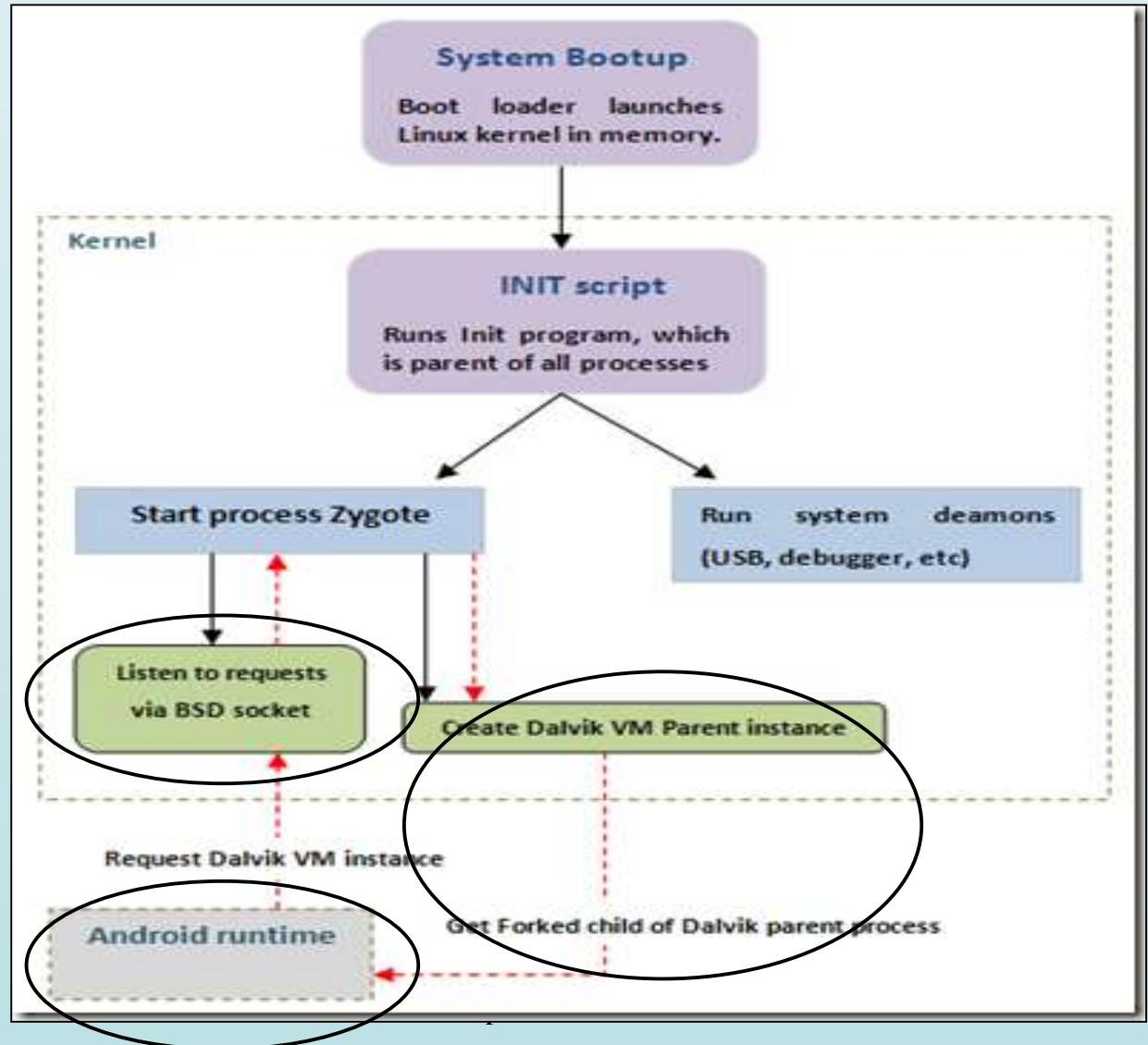
Plus précisément, une fois le boot du système terminé, le système lance le programme INIT, comme sur un O.S. Linux classique, qui lui-même démarre les différents processus nécessaires au bon fonctionnement de l'O.S., dont le processus "Zygote", qui se met en attente de requêtes après avoir lancé la machine virtuelle "principale" :

Permet d'éviter de bloquer l'appareil.

3. La machine virtuelle Dalvik (7/9)

Chaque fois que le système créera une nouvelle application,
il enverra une requête
vers le processus Zygote
demandant la création
d'un sous-processus de
la JVM principale :

→ c'est ce sous-processus faisant fonctionner une nouvelle instance de JVM qui sera attribuée à l'application.



3. La machine virtuelle Dalvik (8/9)

Of Ahead Time, a
silly reordering of
Ahead Of Time

4. Une nouvelle machine virtuelle

La nouvelle machine virtuelle ART, ou **Android Run Time** ne fonctionne plus en **Just-In-Time**, mais en **Ahead-Of-Time** ("avant le temps").

A la différence de Dalvik, le code fonctionnant avec la machine virtuelle est traduit une seule fois **lors de l'installation des applications**. ART utilise un outil **dex2oat** pour réaliser cette compilation. **L'allocation mémoire est aussi optimisée** (nouveau garbage collector).

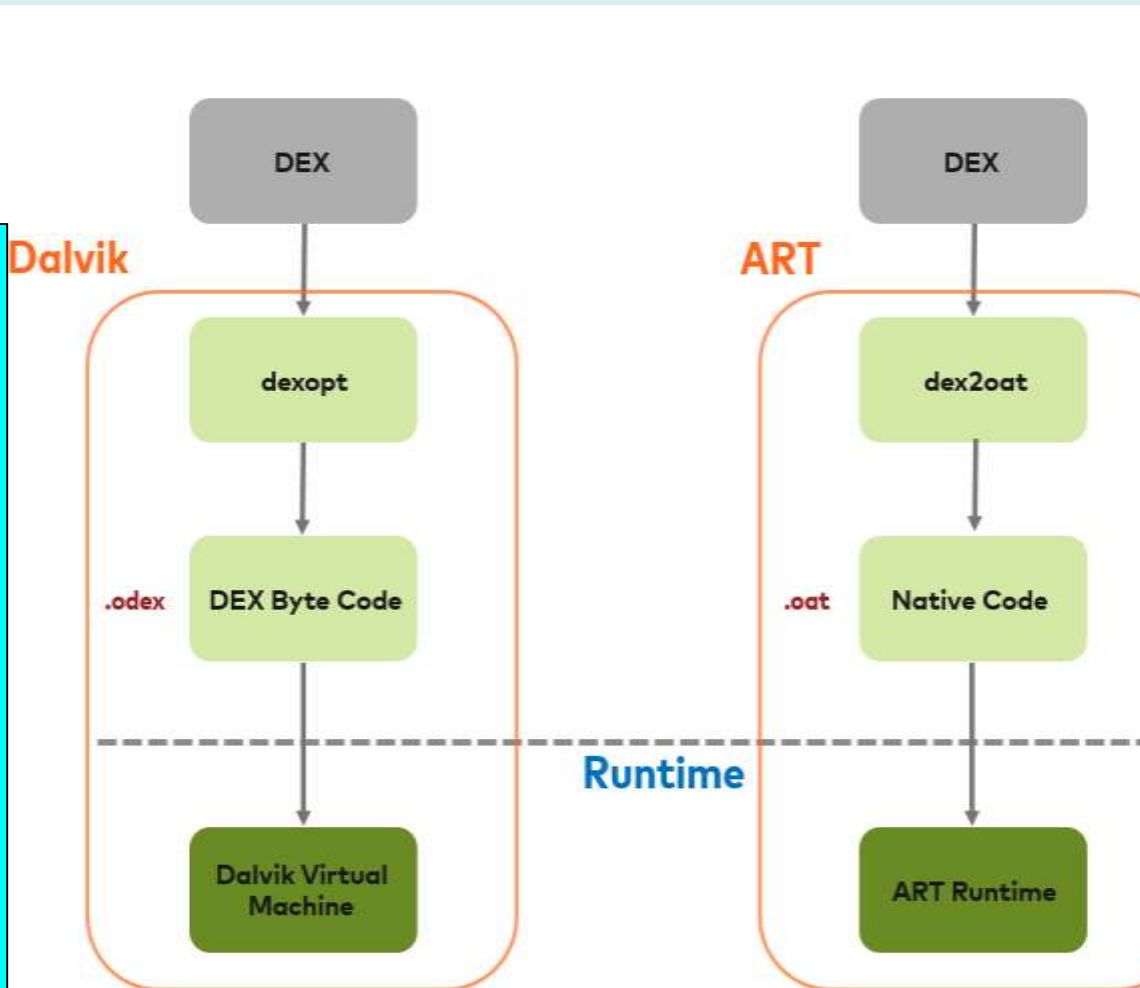
ART était déjà intégré à 4.4 KitKat, mais pas activée par défaut. La version intégrée à Android 5.0 Lollipop l'utilise, avec des modifications concernant la machine virtuelle.

Intérêt ☺ : Les applications sont plus rapides car le code déjà compilé est stocké dans la mémoire et le système n'a donc plus à compiler le code à chaque lancement → les temps de chargement sont réduits.

Inconvénients ☹ : Les temps d'installation seront plus longs et les applications prennent un peu plus de place → problème pour les smartphones bas de gamme. ART et Dalvik sont compatibles : les anciens fichiers dex sont donc correctement exécutés (ouf !).

3. La machine virtuelle Dalvik (9/9)

Just In Time (JIT)
With the Dalvik JIT compiler, **each time when the app is run**, it dynamically **translates** a part of the Dalvik bytecode into machine code. As the execution progresses, more **bytecode is compiled and cached**. Since JIT compiles only a part of the code, it has a **smaller memory footprint** and uses less physical space on the device.



Dalvik vs ART

Ahead Of Time (AOT)
ART is equipped with an Ahead-of-Time compiler. **During the app's installation phase**, it statically **translates** the DEX bytecode into machine code and stores in the device's storage. This is a one-time event which happens when the app is installed on the device. With no need for JIT compilation, the code executes **much faster**. As ART runs app machine code directly (native execution), it doesn't hit the CPU as hard as just-in-time code compiling on Dalvik. Because of less CPU usage results in less battery drain.

4. Le fichier de distribution apk (1/2)

Empaquetage final d'une application Android = un fichier **.apk** (Android PacKage), qui est *une variante du format des fichiers .jar* [on peut donc l'ouvrir avec les outils classiques (7-Zip, WinZip, WinRar, etc)]. Le type MIME est **application/vnd.android.package-archive**.

Ce fichier APK contient

- ◆ l'habituel répertoire META-INF;
- ◆ le fichier **.dex**;
- ◆ un fichier **xml** décrivant l'application (**AndroidManifest.xml**);
- ◆ les fichiers de ressources **.res** et **.arsc** : en fait, res est le répertoire qui contient les ressources proprement dites sous forme de **fichiers XML** tandis que le fichier .arsc (Android ReSourCe) est en fait "la mise à plat" de la table des ressources, c'est-à-dire la forme binaire de ces ressources appelée souvent la "**ressource compilée**" (voir la **classe R.java** décrivant les ressources).

5. Les composants applicatifs Android (1/4)

1. Composants et threads

Une application Android est écrite en Java, mais il ne s'agit pas d'un Java standard.

♦ une telle application n'a **pas de fonction main()** qui servirait d'unique point d'entrée; en fait, c'est une collection de composants qui peuvent être utilisés par d'autres applications ...

♦ une **application** [*task*] peut utiliser ses propres composants ou se servir des composants d'autres applications (si celles-ci le permettent) : elle démarre simplement la partie de code dont elle a besoin.

Comme déjà précisé, celle-ci fonctionne au sein d'un processus Linux, avec **son ID unique** et **sa propre machine virtuelle**, ce qui assure l'isolation du code. Elle possède seule (du moins à priori) les permissions pour en utiliser les composantes et ressources.

Par défaut, **tout se passe dans un seul thread**, ce qui implique que toute **activité** doit s'abstenir d'effectuer des tâches bloquantes – mais rien n'empêche de lancer des threads (classe Thread) ou d'utiliser des **intents** (voir plus loin) *envoyés d'un composant à un autre*.

5. Les composants applicatifs Android (2/4)

Les composants possibles:

- ◆ une activité [*activity*] : objet instance de la classe **Activity** dont on peut lancer l'exécution pour réaliser une tâche quelconque **utilisant un GUI** (il possède une fenêtre par défaut); *c'est l'association de ces activités qui permet de définir une interface graphique complet pour l'utilisateur de l'application*;
- ◆ un service : objet instance de la classe **Service** dont on peut lancer l'exécution en *tâche de fond* et qui ne possède donc **pas d'interface graphique** (exemple typique: une musique de fond); il est évidemment possible de démarrer un service mais aussi de se connecter à lui si il est déjà en cours d'exécution;
- ◆ un récepteur des annonces broadcast [*broadcast receiver*] : les annonces peuvent provenir du système (batterie faible, fuseau horaire modifié, etc) ou d'une autre application (exemple typique : fin d'un téléchargement);
- ◆ un fournisseur de contenu [*content provider*] : un tel objet rend des ressources de l'application disponibles aux autres applications (celles-ci utiliseront un "content resolver", instance de **ContentResolver**) pour viser le "content provider" en question et dialoguer avec lui).

5. Les composants applicatifs Android (3/4)

2. La communication par intents

Les trois premiers types de composants sont activés au moyen d'un "**intent**" : il s'agit d'un **message asynchrone spécifiant une action et contenant l'URI du composant à manipuler ou de l'action à initier** (ceci permet aux composants de communiquer).

C'est le rôle d'un objet implémentant la classe abstraite **Context** de

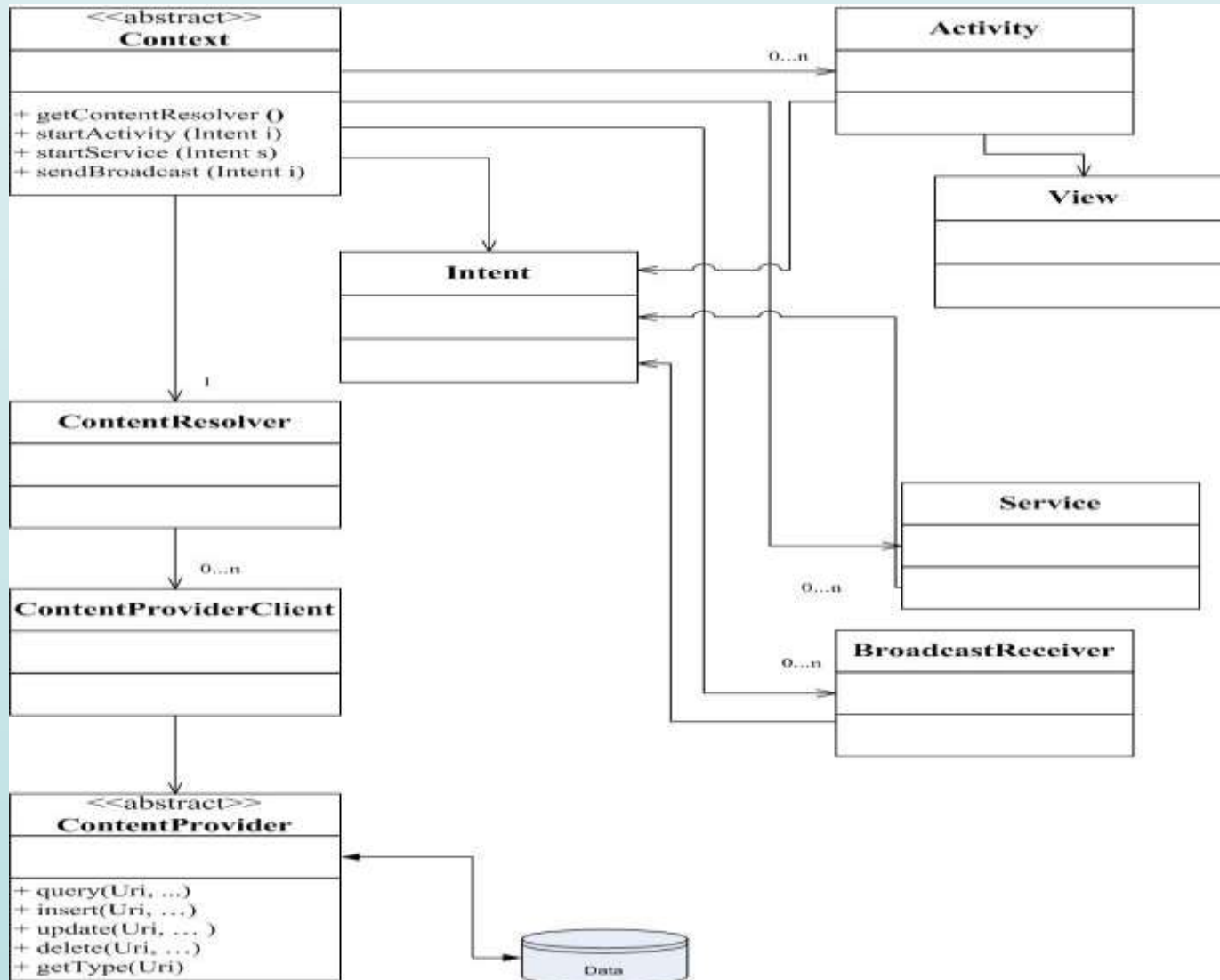
♦ provoquer les démarrages des activités et services, avec par exemple :

```
public abstract void startActivity (Intent intent)
public abstract ComponentName startService (Intent service)
public abstract boolean stopService (Intent service)
public abstract boolean bindService (Intent service, ServiceConnection conn, int flags)
public abstract void sendBroadcast (Intent intent)
```

♦ permettre l'accès aux ressources de l'application, avec par exemple :

```
public abstract ContentResolver getContentResolver ()
    (qui lui-même dispose de la méthode
    public final ContentProviderClient acquireContentProviderClient (Uri uri) )
public abstract Resources getResources ()
```

5. Les composants applicatifs Android (4/4)



6. Le cycle de vie d'une activité Android (1/4)

Un objet "**Activity**", instance d'une classe dérivée de la classe `android.app.Activity` (qui implémente `Context`) est donc un objet dont on peut lancer l'exécution pour réaliser une tâche quelconque et utilisant un GUI = en fait une instance d'une classe dérivée de `android.view.View` (dont les classes dérivées sont appelées "**widgets**")

```
package my.apps.learning;
import android.app.Activity;
import android.os.Bundle;

public class hello extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Un objet de type **Bundle** permet de sauvegarder l'état de l'activité lorsque celle-ci est détruite par le système et uniquement par le système pour être recréée ensuite à l'état identique.

L'utilisateur qui ferme volontairement une activité, par exemple en pressant la touche retour de l'appareil, ne provoquera pas la sauvegarde de cet objet de type Bundle.

Interaction uniquement avec 1 seule activité à la fois (pas de chevauchement d'activités)

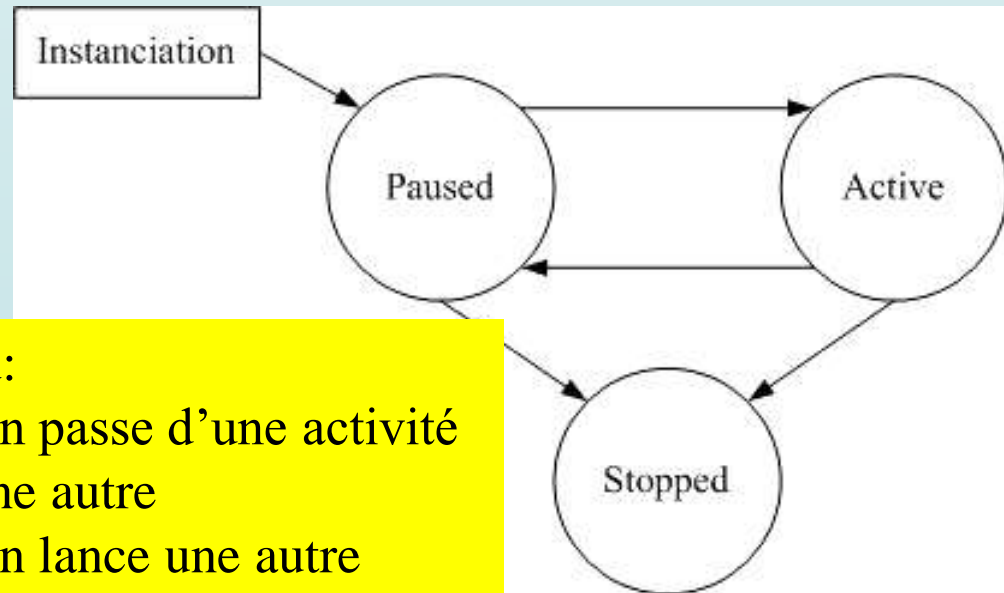
6. Le cycle de vie d'une activité Android (2/4)

Une **application** peut consister en une seule activité ou en un groupe d'activités, l'une d'entre elles étant la principale. On passe d'une activité à une autre via des espèces de messages asynchrones (INTENT) chacune pouvant passer la main à une autre.

Le cycle de vie d'une activité est classique dans le monde des mobiles : il rappelle classiquement, par exemple, celui d'une MIDlet de J2ME :

Paused:

- si on passe d'une activité à une autre
- si on lance une autre application



Une application peut être constituée de plusieurs activités, mais **l'utilisateur n'interagit qu'avec une seule d'entre elles à un instant donné**. A priori (même si ce n'est pas indispensable), une application possède au moins une activité.

6. Le cycle de vie d'une activité Android (3/4)

Le passage d'un état à l'autre **est notifié** à l'application par l'appel aux méthodes (protected) :

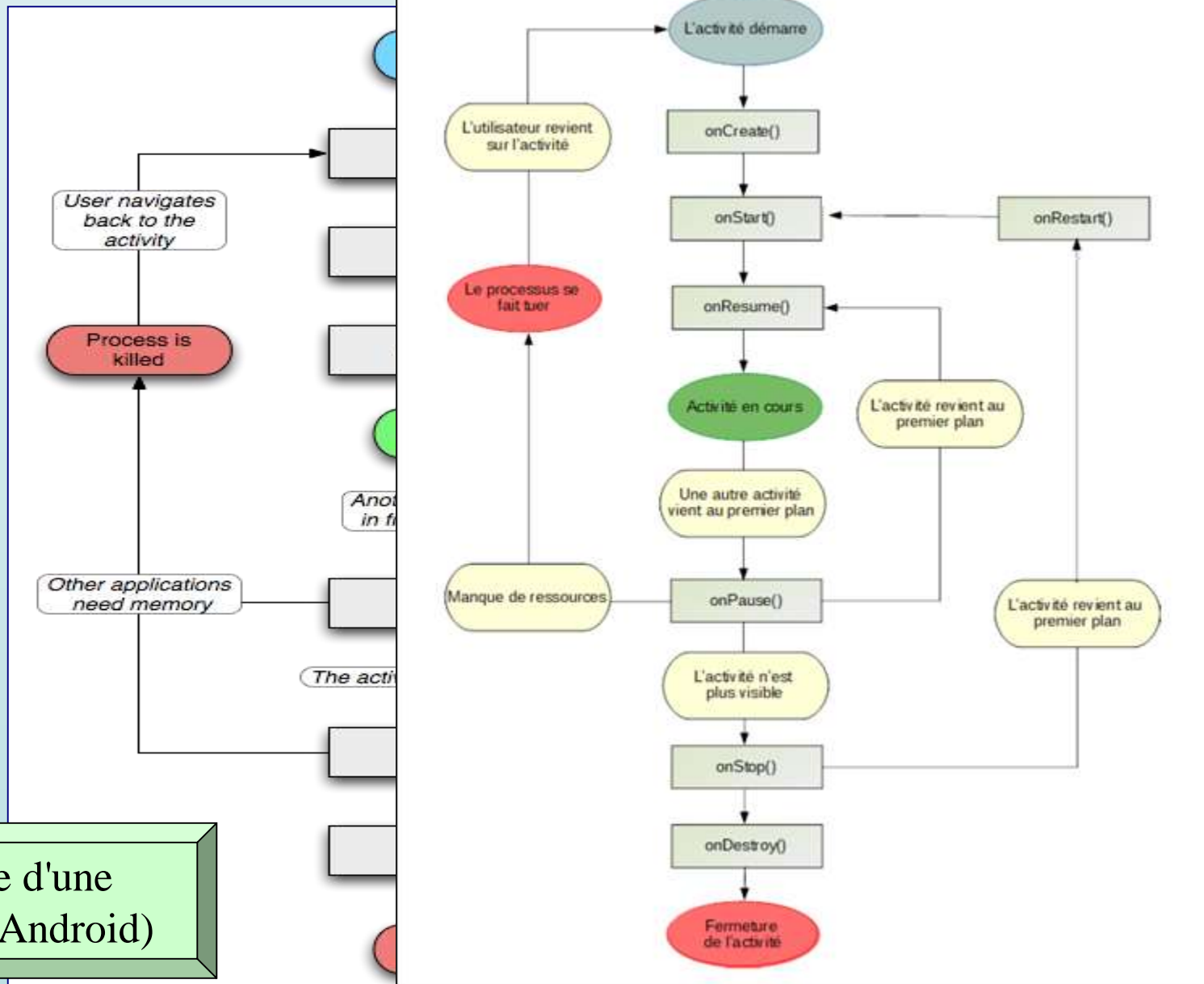
cycle de vie de base		Obligatoire pas les autres
<u>void onCreate(Bundle savedInstanceState);</u>	<u>void onDestroy();</u>	
cycle de vie visible		
<u>void onStart();</u>	<u>void onStop();</u>	
cycle de vie à l'arrière-plan		
<u>void onPause();</u>	<u>void onRestart();</u>	<u>void onResume();</u>

Si une activité est à l'arrêt (temporaire ou définitif), le processus qui lui donne son contexte peut être détruit en cas de besoin de mémoire pour d'autres applications

Un **service** diffère d'une activité par le fait qu'il peut attendre que l'on fasse appel à lui :

- ♦ il peut être démarré [**startService**] en cas de besoin, puis arrêté (par lui-même ou par un autre composant);
- ♦ il peut se mettre en attente d'une connexion établie par un autre composant [**bindService**] qui l'utilisera en utilisant les méthodes d'un interface qu'il a défini et rendu public.

6. Le cycle de vie d'une activité Android (4/4)



cycle de vie d'une
activité (doc. Android)

7. Une application Android de base (1/3)

Une classe basique dérivée d'**Activity** va *typiquement redéfinir deux méthodes* :

1) protected void **onCreate** (Bundle savedInstanceState)

Cette méthode est appelée quand l'activité est démarrée.

Le paramètre instance de android.os.**Bundle** est une implémentation de l'interface android.os.**Parcelable** = les classes qui peuvent être sauvées ou restaurées à partir d'un objet android.os.**Parcel** = ce qui est transmis par IPC dans le contexte de la gestion des processus sur le mobile.

**** Classiquement, la méthode onCreate() appellera notamment :

public void **setContentView** (View view)

[utilise une classe android.view.**View** qui désigne simplement une zone rectangulaire susceptible de contenir des graphiques et d'interagir avec l'utilisateur]

ou

public void **setContentView** (int **layoutResID**)

[fait référence à une vue considérée comme une ressource définie dans **R.java**]

2) protected void **onPause** ()

Appelée quand l'activité passe à l'arrière-plan sans être détruite pour la cause.



Patience ! On va y venir

```
/* AUTO-GENERATED FILE. DO NOT  
MODIFY....
```

```
*/
```

```
package my.apps.learning;
```

```
public final class R {
```

```
    public static final class attr {  
    }
```

```
    public static final class drawable {  
        public static final int icon=0x7f020000;  
    }
```

```
    public static final class layout {  
        public static final int main=0x7f030000;  
    }
```

```
    public static final class string {  
        public static final int app_name=0x7f040000;  
    }  
}
```

```
@Override
```

```
public void onCreate(Bundle savedInstanceState)  
{
```

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

```
}
```

R.java

oid de base (2/3)

Hello.java

main.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    >  
    <TextView  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:text="Hello World, hello"  
    />  
</LinearLayout>
```

.java → class → dex

res → arsc

7. Une application Android de base (3/3)

Pourquoi décrire les ressources avec des fichiers XML ?

L'idée est que toutes les propriétés des composants visuels (les "**widgets**") sont certes accessibles dans le code Java mais peuvent également être fixées par des attributs dans le fichier XML du layout.

Les **avantages** de cette manière de faire sont :

♦ **une séparation de la conception visuelle et du code de la logique de l'application** car il suffit alors qu'une équipe travaille sur l'aspect visuel de l'application (sur les fichiers XML) alors qu'une autre équipe s'occupe des traitements de l'application (le code "métier") :

- ceci permet de rendre le code métier plus bref car moins surchargé;
- cette séparation permet des changements plus rapides et aisés dans les choix des différents paramètres de l'interface.

♦ et donc **une réduction des risques qu'une modification du code source de la vue perturbe l'application** : un changement de l'aspect visuel n'implique pas de changer quoi que ce soit dans le code métier car il suffit de modifier le fichier XML.

8. L'installation de l'environnement Android (1/18)

On suppose le JDK "standard" installé et il nous faut donc installer un **SDK Android**.

Il peut être obtenu sur <http://developer.android.com/sdk/index.html>.



SDK Setup.exe

The screenshot shows the Android Studio website. The main heading is "Android Studio" with the subtitle "The Official IDE for Android". Below this, there is a description of the IDE's capabilities and a large green button labeled "DOWNLOAD ANDROID STUDIO 2.1 FOR WINDOWS (126.2 MB)". To the left, there is a sidebar with links to "Back to Developers", "DOWNLOAD", "FEATURES", and "USER GUIDE". Below the main content, there are links to "Read the docs" and "See the release notes". At the bottom, there are links to "Features", "Latest", "Resources", "Videos", and "Download Options".

2019: redirigé vers ➔

NDK

8. L'installation de l'environnement Android (2/18)

Develop > Tools > **Download Android Studio and SDK Tools** Developer Console

Download ^

Installing the SDK

Adding SDK Packages

Android Studio v

Workflow v

Tools Help v

Build System v

Performance Tools v

Testing Tools v

Support Library v

Data Binding Library

All Android Studio Packages

Select a specific Android Studio package for your platform. Also see the [Android Studio release notes](#).

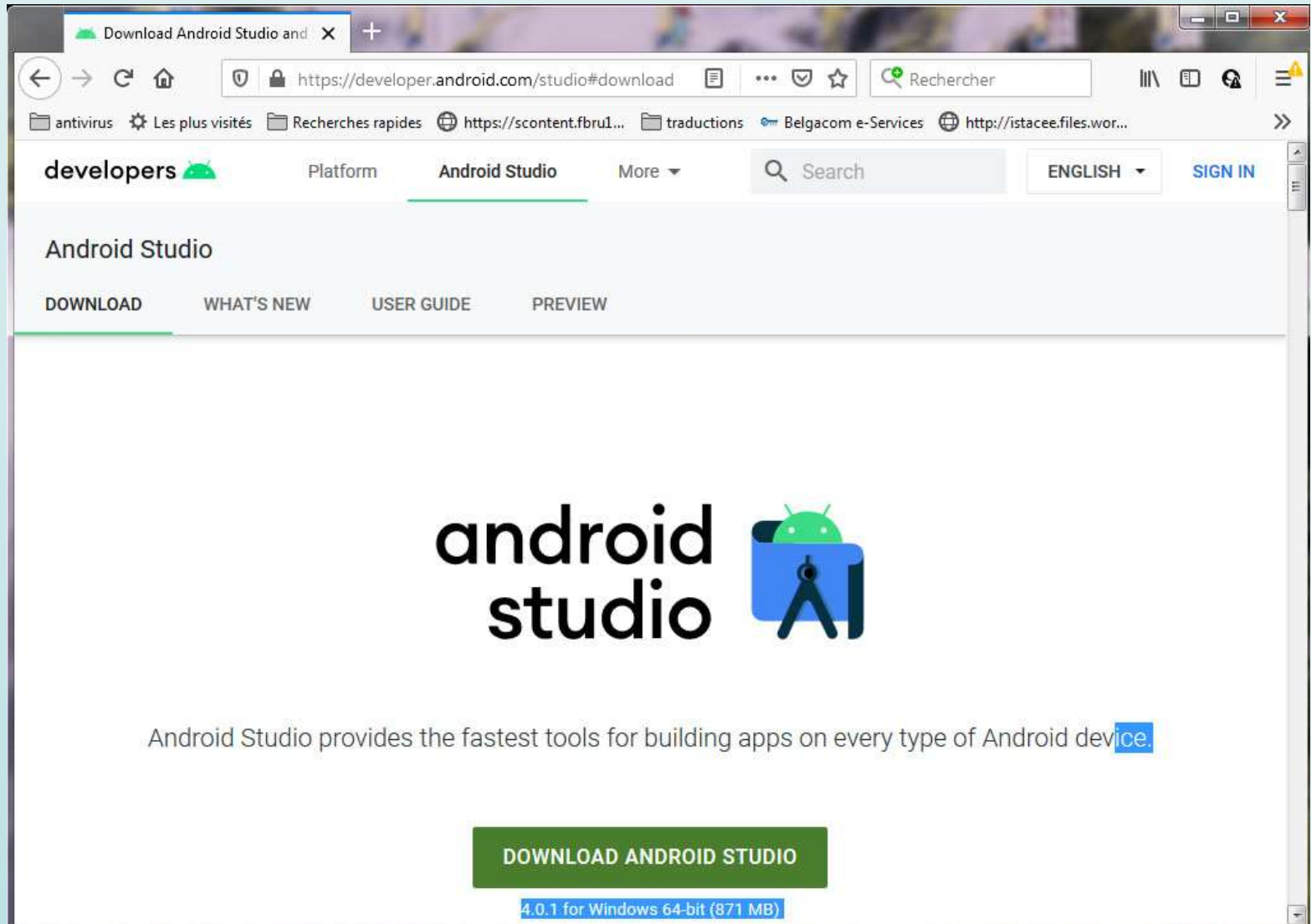
Platform	Package	Size	SHA-1 Checksum
Windows	android-studio-bundle-141.2178183-windows.exe (Recommended)	1136982712 bytes	c7d39c529dd434489da9d086ff689d34dc791526
	android-studio-ide-141.2178183-windows.exe (No SDK tools included)	321810248 bytes	b5d1aaa000729c03a3cf980add79d1b93121c56d
	android-studio-ide-141.2178183-windows.zip	344424713 bytes	3134f226b5f3c3f74d4fc2d9cff03a4458f01d69
Mac OS X	android-studio-ide-141.2178183-mac.dmg	368335367 bytes	75b67eb15a34a152a40e7189484ab0ebc375b877
Linux	android-studio-ide-141.2178183-linux.zip	352010593 bytes	cf780413f8c8223eb348bd27c19a9c04b75eae2

8. L'installation de l'environnement Android (3/18)

The screenshot shows the Android Studio User Guide page for 'Install Android Studio'. A Windows file dialog is overlaid on the page, asking to register the file 'android-studio-bundle-143.3101438-windows.exe'. The dialog text is in French: 'Ouverture de android-studio-bundle-143.3101438-windows.exe', 'Vous avez choisi d'ouvrir:', 'android-studio-bundle-143.3101438-windows.exe', 'qui est un fichier de type : Binary File (1,2 Go)', 'à partir de : https://dl.google.com', and 'Voulez-vous enregistrer ce fichier?'. The buttons are 'Enregistrer le fichier' and 'Annuler'.

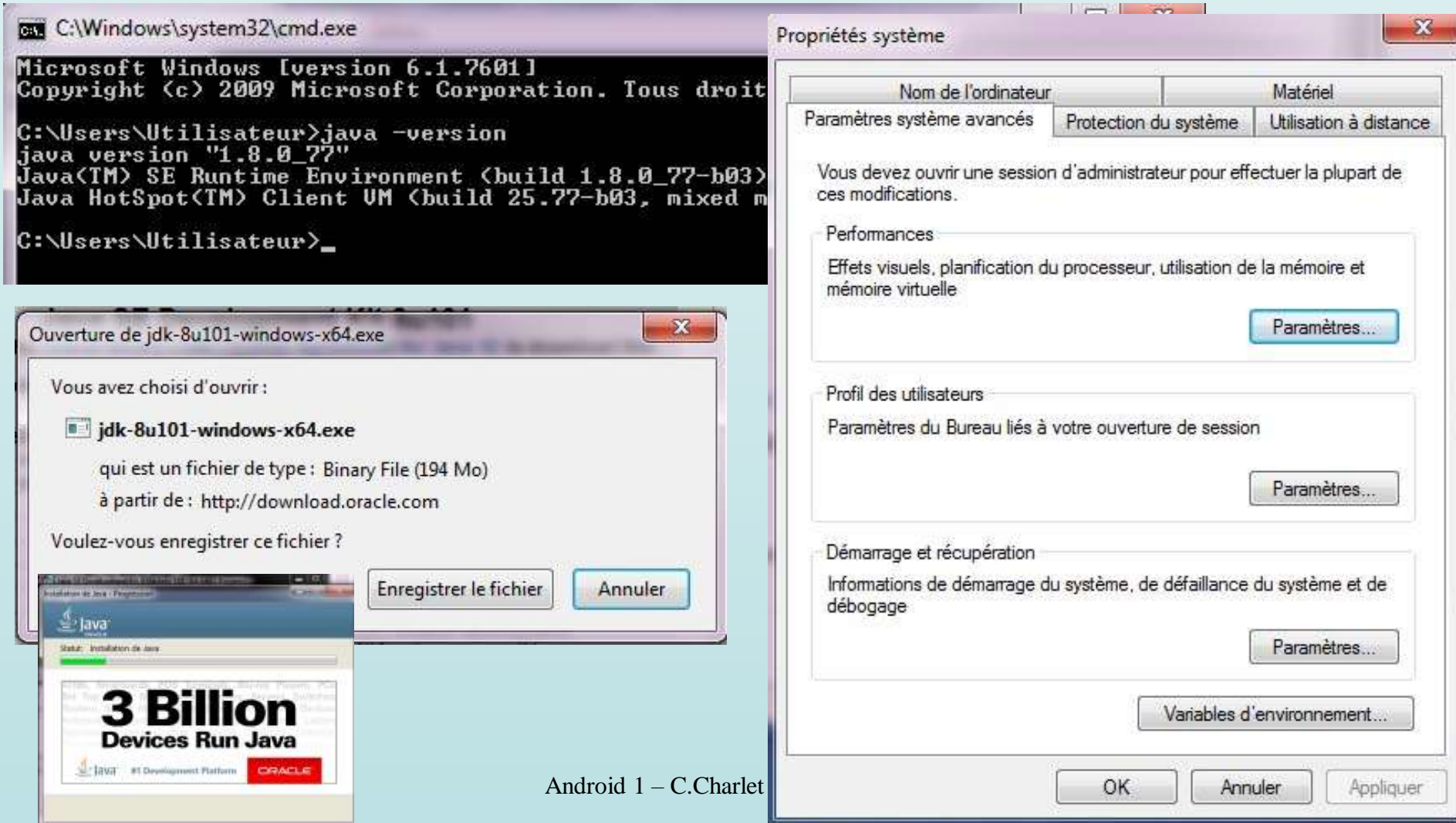
7/2023: android-studio-ide-193.6514223-windows.exe

Bref:



8. L'installation de l'environnement Android (4/18)

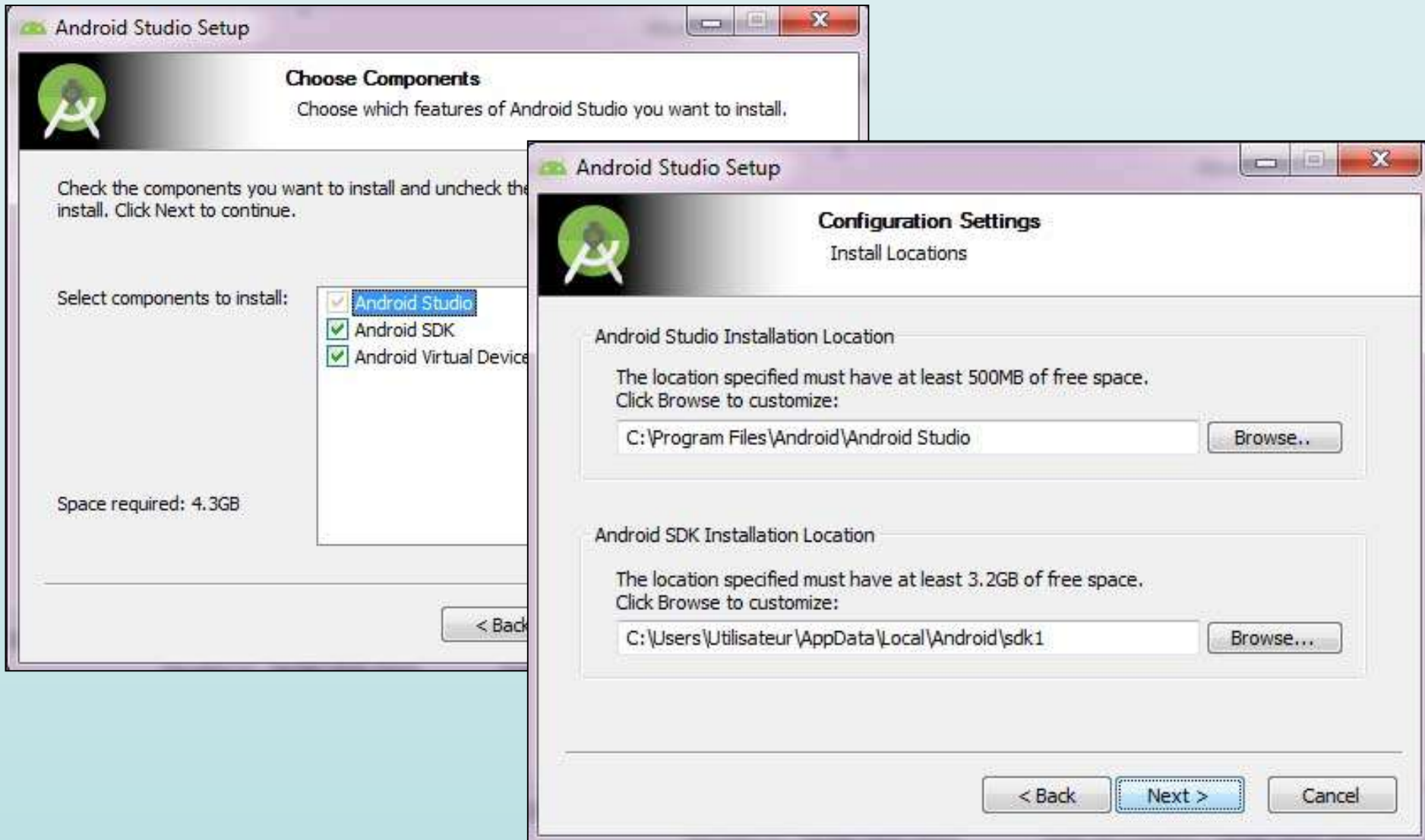
Le programme d'installation recherche le JDK de la version 1.8 :



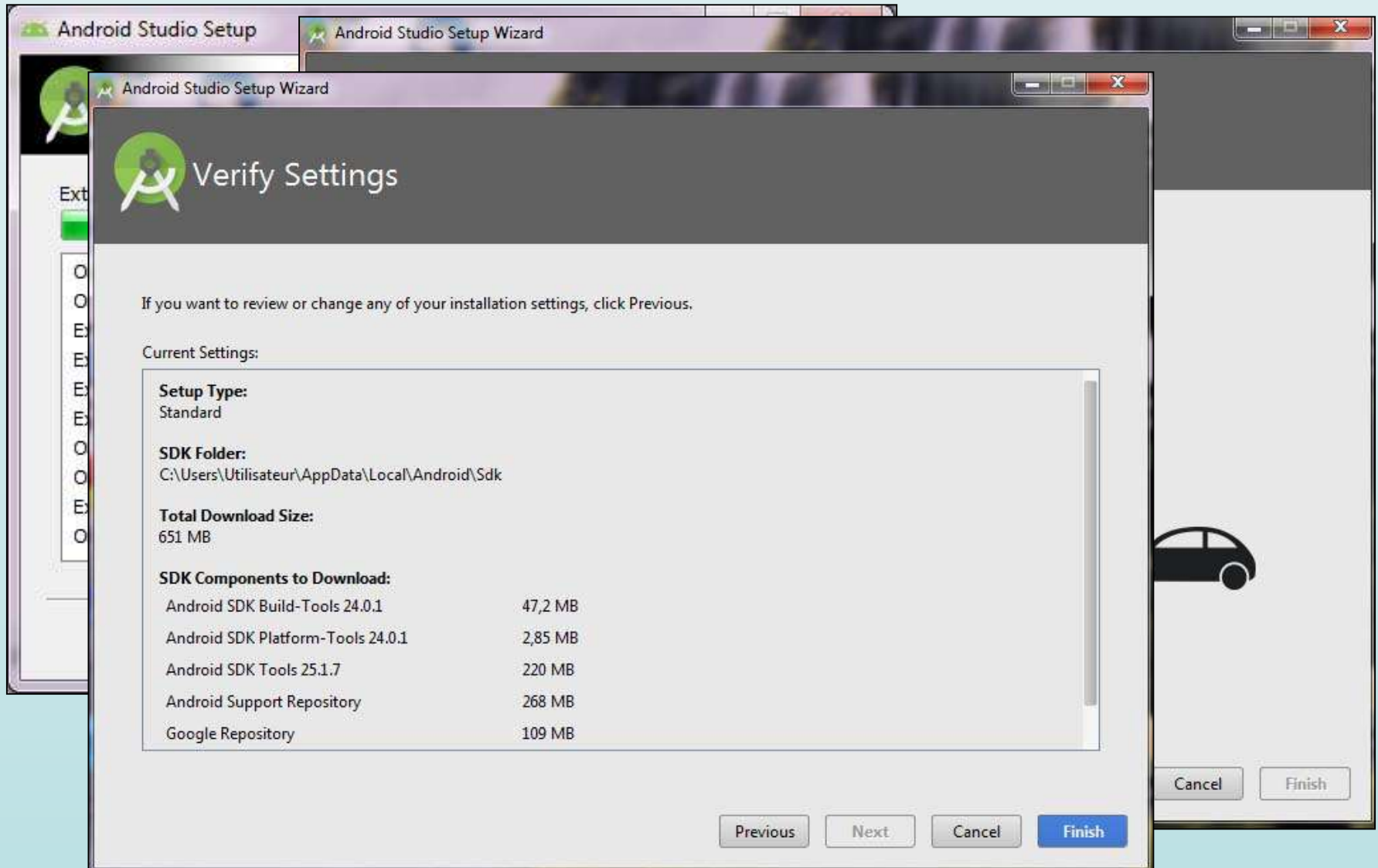
8. L'installation de l'environnement Android (5/18)



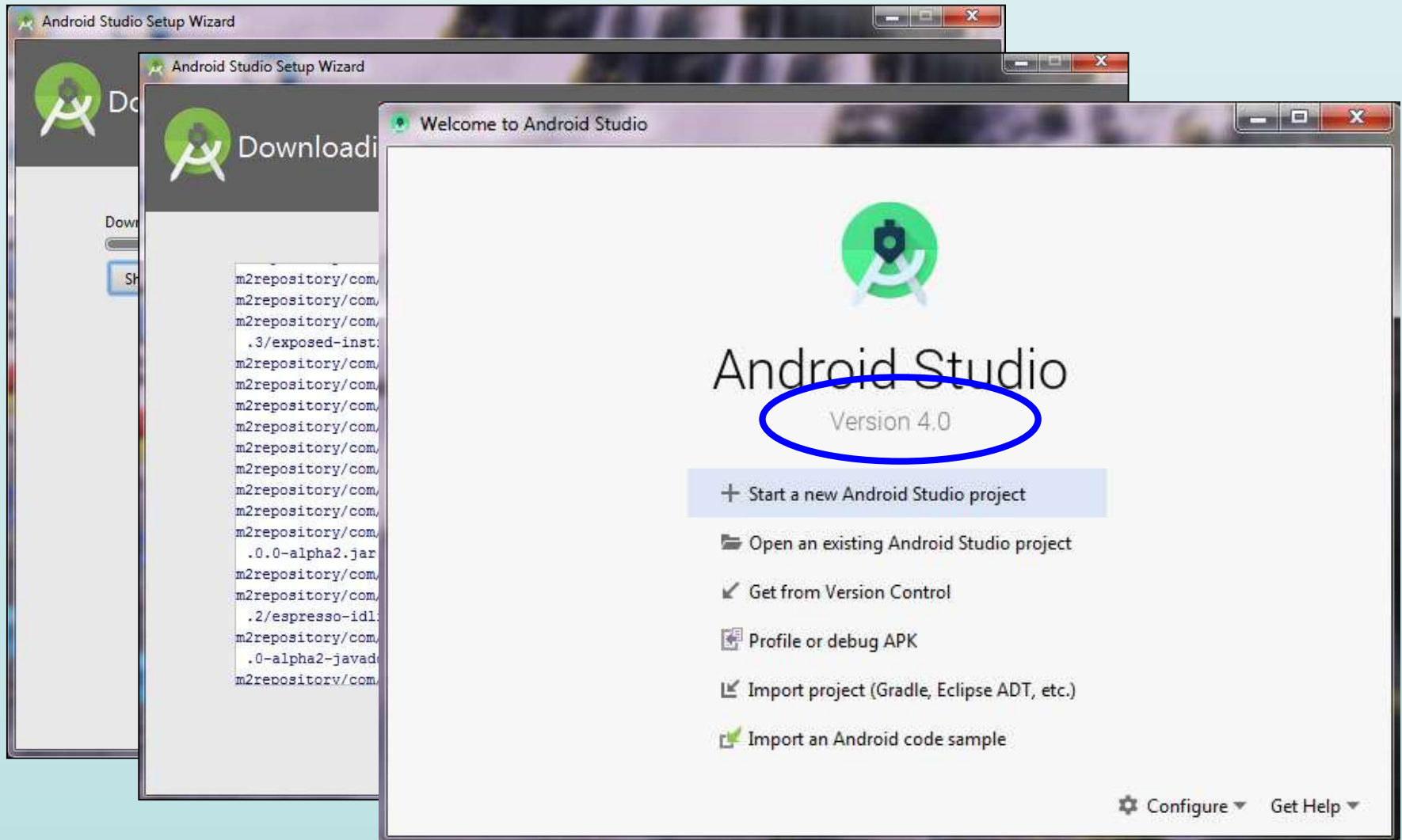
8. L'installation de l'environnement Android (6/18)



8. L'installation de l'environnement Android (7/18)



8. L'installation de l'environnement Android (8/18)



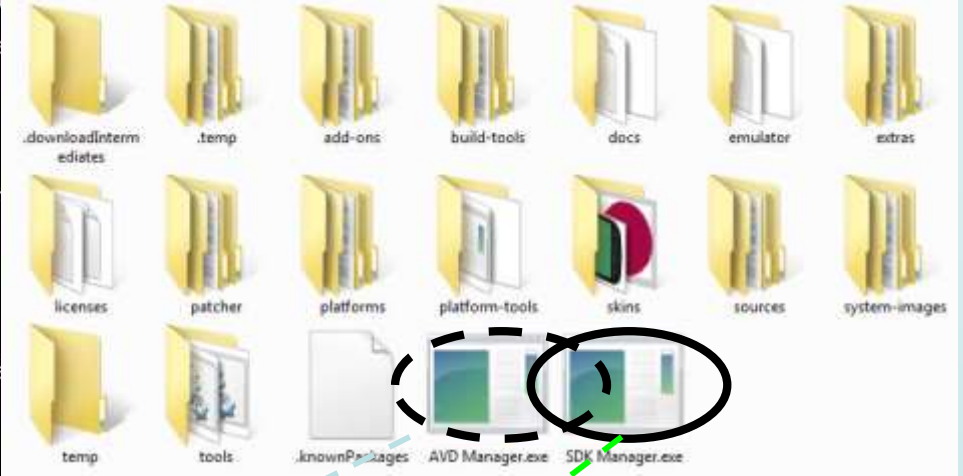
8. L'installation de l'environnement Android (9/18)

Administrateur : C:\Windows\System32\cmd.exe

```
C:\Users\Utilisateur\AppData\Local>cd \users\utilisat
C:\Users\Utilisateur\AppData\Local\Android>dir
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est 8EB4-D609

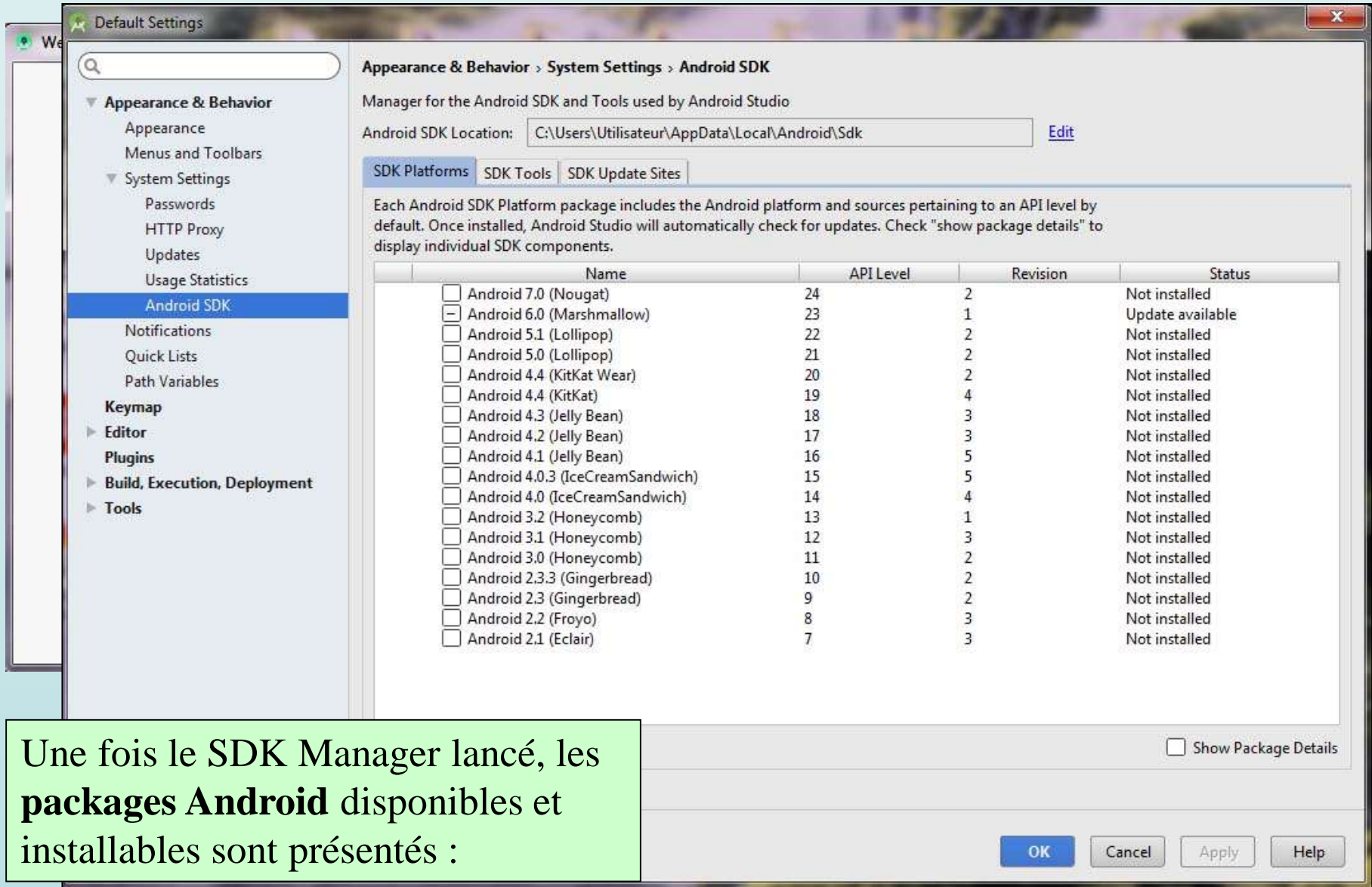
Répertoire de C:\Users\Utilisateur\AppData\Local\An
18/08/2016  15:43    <REP>          .
18/08/2016  15:43    <REP>          ..
07/07/2020  14:11    <REP>          sdk
18/08/2016  15:49    <REP>          sdk1
                0 fichier(s)                0 octets
                4 Rép(s) 18.940.764.160 octets libre
C:\Users\Utilisateur\AppData\Local\Android>cd sdk
C:\Users\Utilisateur\AppData\Local\Android\sdk>dir
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est 8EB4-D609

Répertoire de C:\Users\Utilisateur\AppData\Local\Android\sdk
07/07/2020  14:11    <REP>          .
07/07/2020  14:11    <REP>          ..
10/07/2020  11:44    <REP>          .downloadIntermediates
17/07/2020  09:26    <REP>          16 .knownPackages
10/07/2020  11:47    <REP>          .temp
13/11/2015  01:17    <REP>          add-ons
13/11/2015  01:18    <REP>          220.209 AVD Manager.exe
07/07/2020  14:06    <REP>          build-tools
18/08/2016  15:51    <REP>          docs
07/07/2020  14:06    <REP>          emulator
18/08/2016  16:02    <REP>          extras
10/07/2020  11:47    <REP>          licenses
07/07/2020  14:05    <REP>          patcher
07/07/2020  14:06    <REP>          platform-tools
10/07/2020  10:10    <REP>          platforms
13/11/2015  01:18    <REP>          220.721 SDK Manager.exe
10/07/2020  10:42    <REP>          skins
07/07/2020  14:39    <REP>          sources
10/07/2020  11:42    <REP>          system-images
21/08/2016  09:29    <REP>          temp
18/08/2016  15:59    <REP>          tools
                3 fichier(s)                440.946 octets
                18 Rép(s) 18.936.516.608 octets libres
```



2 sdk ? parce que réinstallation !

8. L'installation de l'environnement Android (10/18)



Default Settings

Appearance & Behavior > System Settings > Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: [Edit](#)

SDK Platforms | SDK Tools | SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

	Name	API Level	Revision	Status
<input type="checkbox"/>	Android 7.0 (Nougat)	24	2	Not installed
<input checked="" type="checkbox"/>	Android 6.0 (Marshmallow)	23	1	Update available
<input type="checkbox"/>	Android 5.1 (Lollipop)	22	2	Not installed
<input type="checkbox"/>	Android 5.0 (Lollipop)	21	2	Not installed
<input type="checkbox"/>	Android 4.4 (KitKat Wear)	20	2	Not installed
<input type="checkbox"/>	Android 4.4 (KitKat)	19	4	Not installed
<input type="checkbox"/>	Android 4.3 (Jelly Bean)	18	3	Not installed
<input type="checkbox"/>	Android 4.2 (Jelly Bean)	17	3	Not installed
<input type="checkbox"/>	Android 4.1 (Jelly Bean)	16	5	Not installed
<input type="checkbox"/>	Android 4.0.3 (IceCreamSandwich)	15	5	Not installed
<input type="checkbox"/>	Android 4.0 (IceCreamSandwich)	14	4	Not installed
<input type="checkbox"/>	Android 3.2 (Honeycomb)	13	1	Not installed
<input type="checkbox"/>	Android 3.1 (Honeycomb)	12	3	Not installed
<input type="checkbox"/>	Android 3.0 (Honeycomb)	11	2	Not installed
<input type="checkbox"/>	Android 2.3.3 (Gingerbread)	10	2	Not installed
<input type="checkbox"/>	Android 2.3 (Gingerbread)	9	2	Not installed
<input type="checkbox"/>	Android 2.2 (Froyo)	8	3	Not installed
<input type="checkbox"/>	Android 2.1 (Eclair)	7	3	Not installed

☐ Show Package Details

OK Cancel Apply Help

Une fois le SDK Manager lancé, les **packages Android** disponibles et installables sont présentés :

8. L'installation de l'environnement Android (11/18)

The screenshot displays the 'Appearance & Behavior' settings in Android Studio, specifically the 'System Settings' section for the 'Android SDK'. The 'Manager for the Android SDK and Tools used by Android Studio' is shown with the 'Android SDK Location' set to 'C:\Users\Utilisateur\AppData\Local\Android\Sdk'. The 'SDK Platforms' tab is active, showing a list of Android versions from 7.0 (Nougat) down to 2.1 (Eclair). A 'Confirm Change' dialog box is open, indicating that several components will be installed, including sources for various SDK revisions and specific platform versions (e.g., Android SDK Platform 18 revision 3, Android SDK Platform 19 revision 4). In the background, the 'Component Installer' window is visible, showing the progress of downloading and installing these requested components.

8. L'installation de l'environnement Android (12/18)

Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

☒ Phone and Tablet

Minimum SDK: API 18: Android 4.3 (Jelly Bean)

- API 15: Android 4.0.3 (IceCreamSandwich)
- API 16: Android 4.1 (Jelly Bean)
- API 17: Android 4.2 (Jelly Bean)
- API 18: Android 4.3 (Jelly Bean)
- API 19: Android 4.4 (KitKat)
- API 21: Android 5.0 (Lollipop)
- API 22: Android 5.1 (Lollipop)
- API 23: Android 6.0 (Marshmallow)

☐ Wear

Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ TV

Minimum SDK: API 21: Android 5.0 (Lollipop)

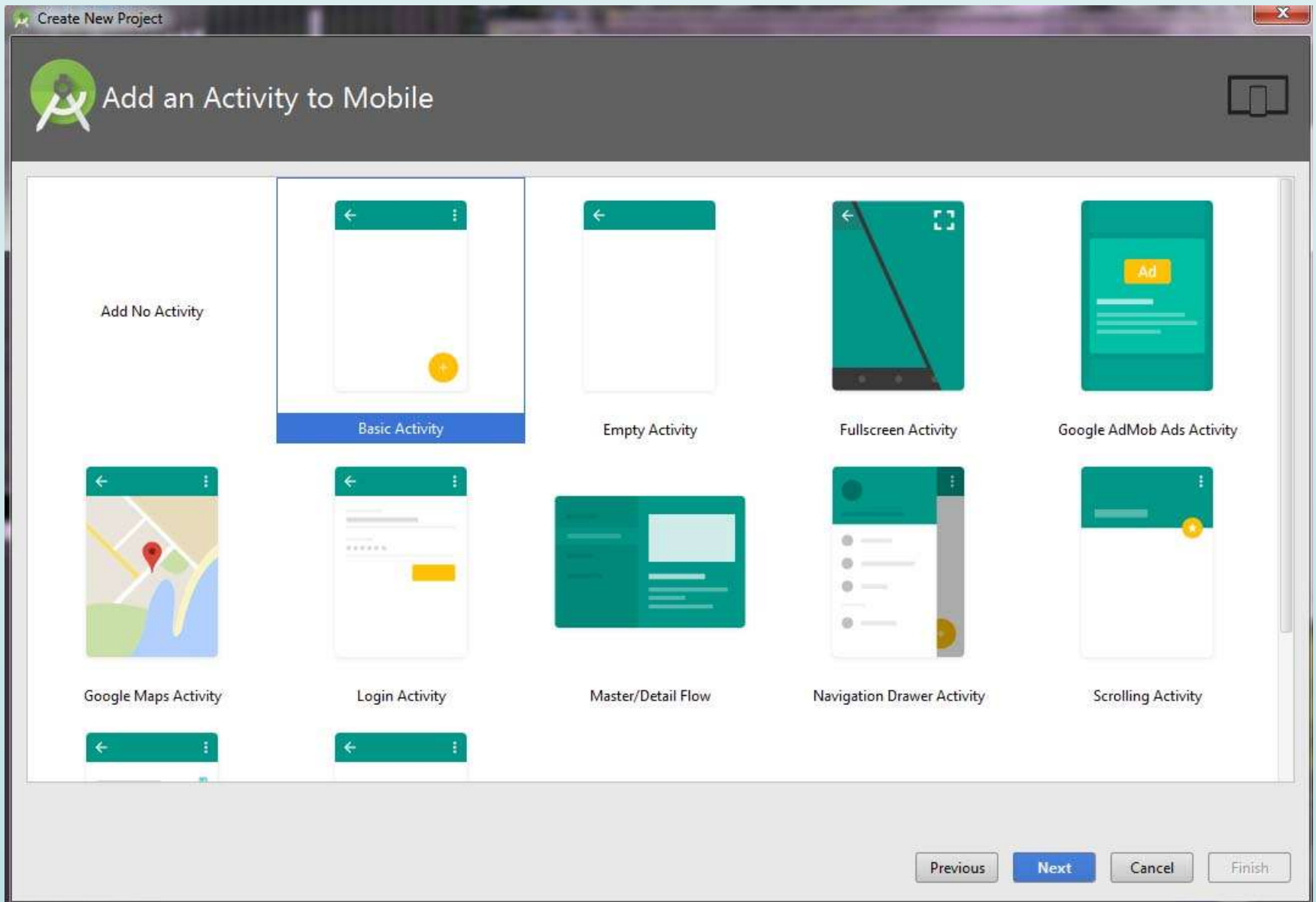
☐ Android Auto

☐ Glass

Minimum SDK: Glass Development Kit Preview (API 19)

Previous Next Cancel Finish

8. L'installation de l'environnement Android (13/18)



8. L'installation de l'environnement Android (14/18)

The image shows two overlapping windows from the Android Studio application. The background window is titled 'Create New Project' and 'Customize the Activity'. It features a preview of a 'Basic Activity' on the left, which has a green header bar with a back arrow and a yellow floating action button. To the right of the preview are input fields for 'Activity Name' (HelloActivity), 'Layout Name' (activity_hello), 'Title' (HelloActivity), and 'Menu Resource Name' (menu_hello). There is also an unchecked checkbox for 'Use a Fragment'. Below these fields is a note: 'Creates a new basic activity with an app bar.' At the bottom of this window, there is a text box containing the instruction: 'The application name for most apps begins with an uppercase letter'. The foreground window is titled 'Welcome to Android Studio' and shows the Android Studio logo and the text 'Creating project...'. It lists several options: 'Open an existing Android Studio project', 'Check out project from Version Control', 'Import project (Eclipse ADT, Gradle, etc.)', and 'Import an Android code sample'. At the bottom right of this window are buttons for 'Configure', 'Get Help', 'Previous', 'Next', 'Cancel', and 'Finish'.

Create New Project

Customize the Activity

Creates a new basic activity with an app bar.

Activity Name: HelloActivity

Layout Name: activity_hello

Title: HelloActivity

Menu Resource Name: menu_hello

☐ Use a Fragment

Basic Activity

The name of the activity class to create

The application name for most apps begins with an uppercase letter

Welcome to Android Studio

Creating project...

Initializing module (Phone and Tablet)

Open an existing Android Studio project

Check out project from Version Control

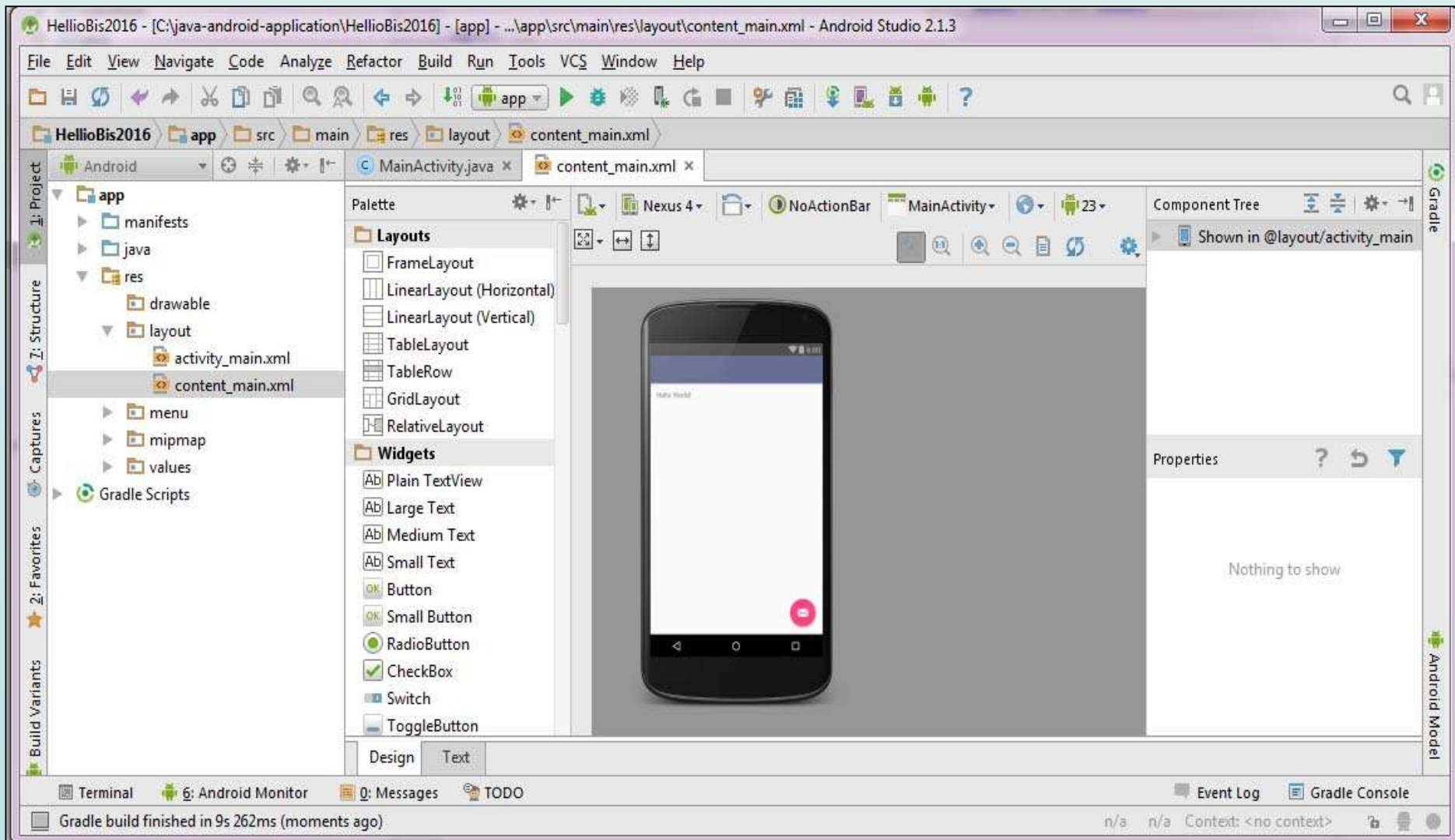
Import project (Eclipse ADT, Gradle, etc.)

Import an Android code sample

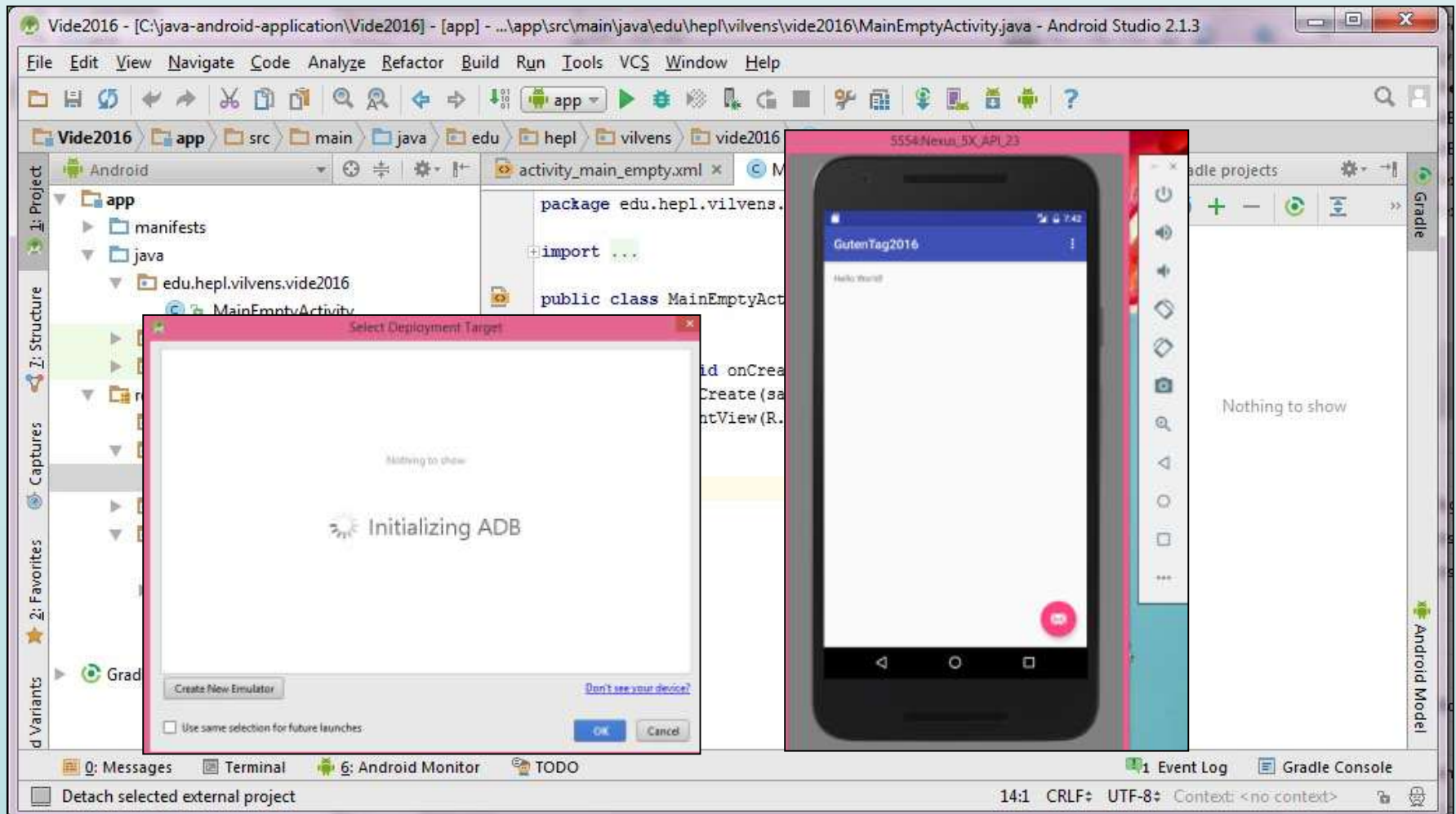
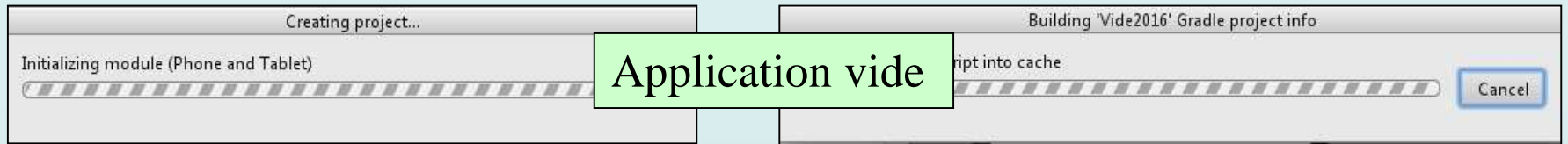
Configure Get Help

Previous Next Cancel Finish

8. L'installation de l'environnement Android (15/18)



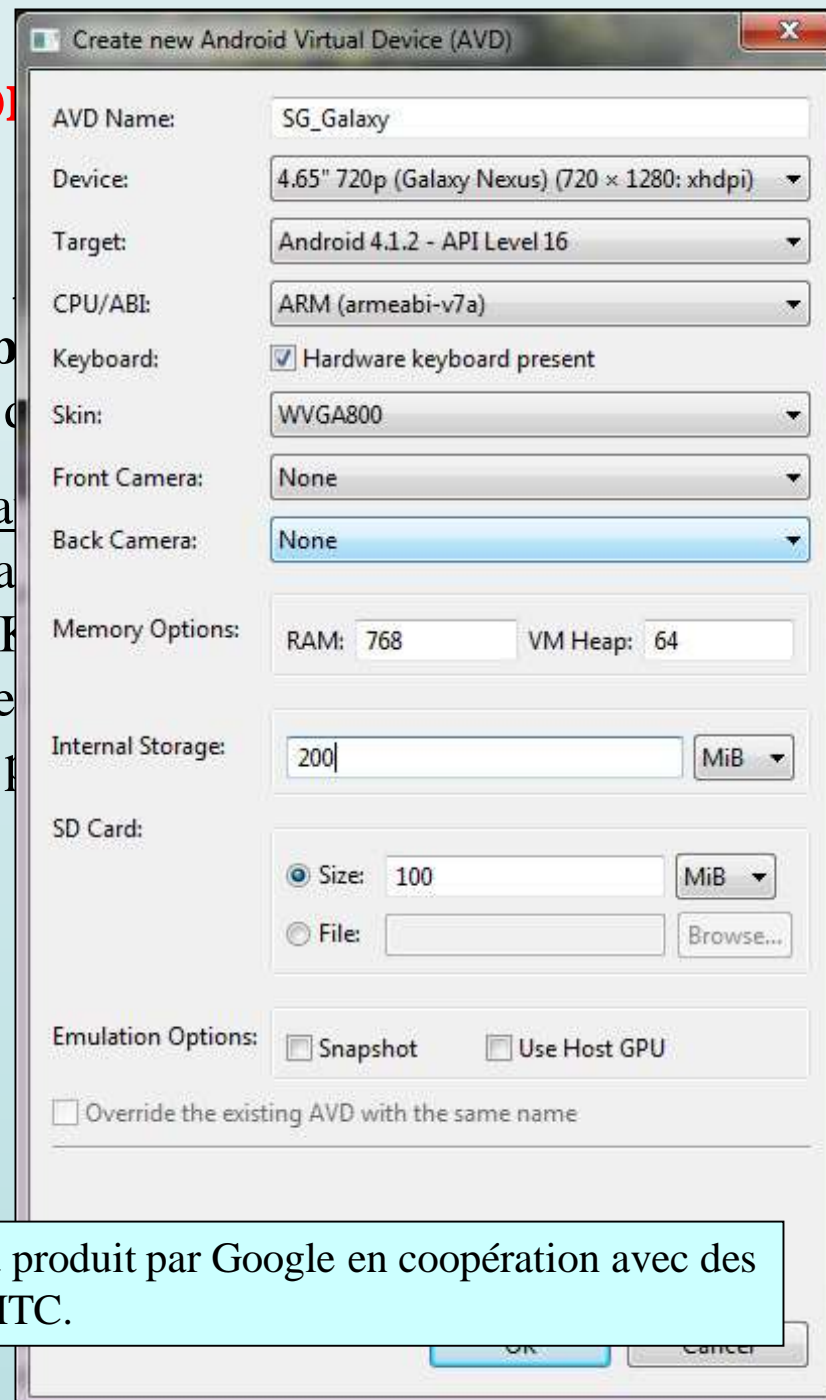
8. L'installation de l'environnement Android (16/18)



8. L'installation de l'environnement

Mais il faudra probablement encore installer un émulateur, c'est-à-dire **une version de l'émulateur de mobile** pour représenter au mieux un mobile qui existe.

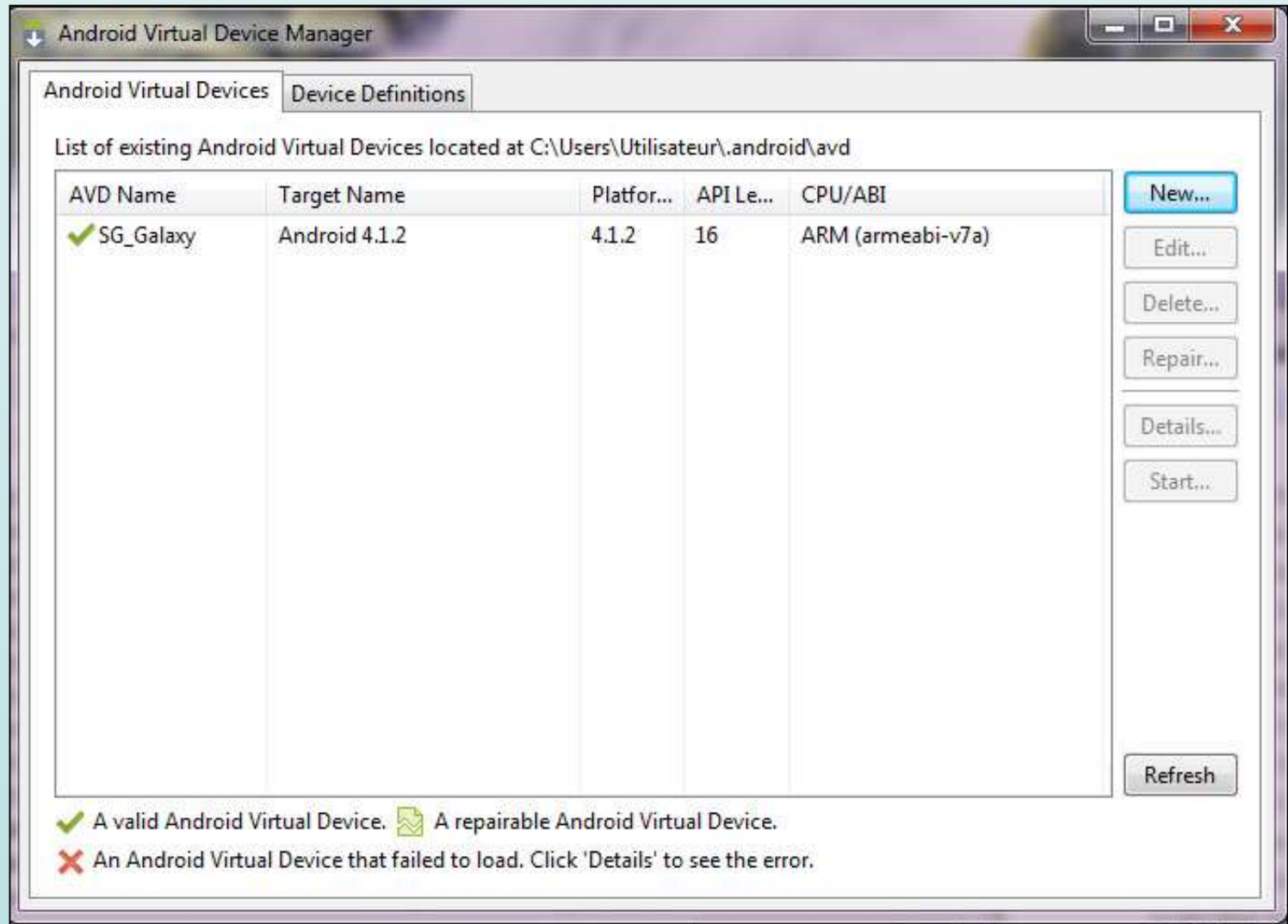
L'étape est quasi-obligatoire, car l'AVD par défaut (SDK manuellement) est un **Google Nexus**. Cela se fait via **"Android Studio Manager"**, appelé directement ou depuis le SDK Manager pour définir un appareil virtuel. Un tableau initial est fourni des AVD; un appui sur le bouton New donne le



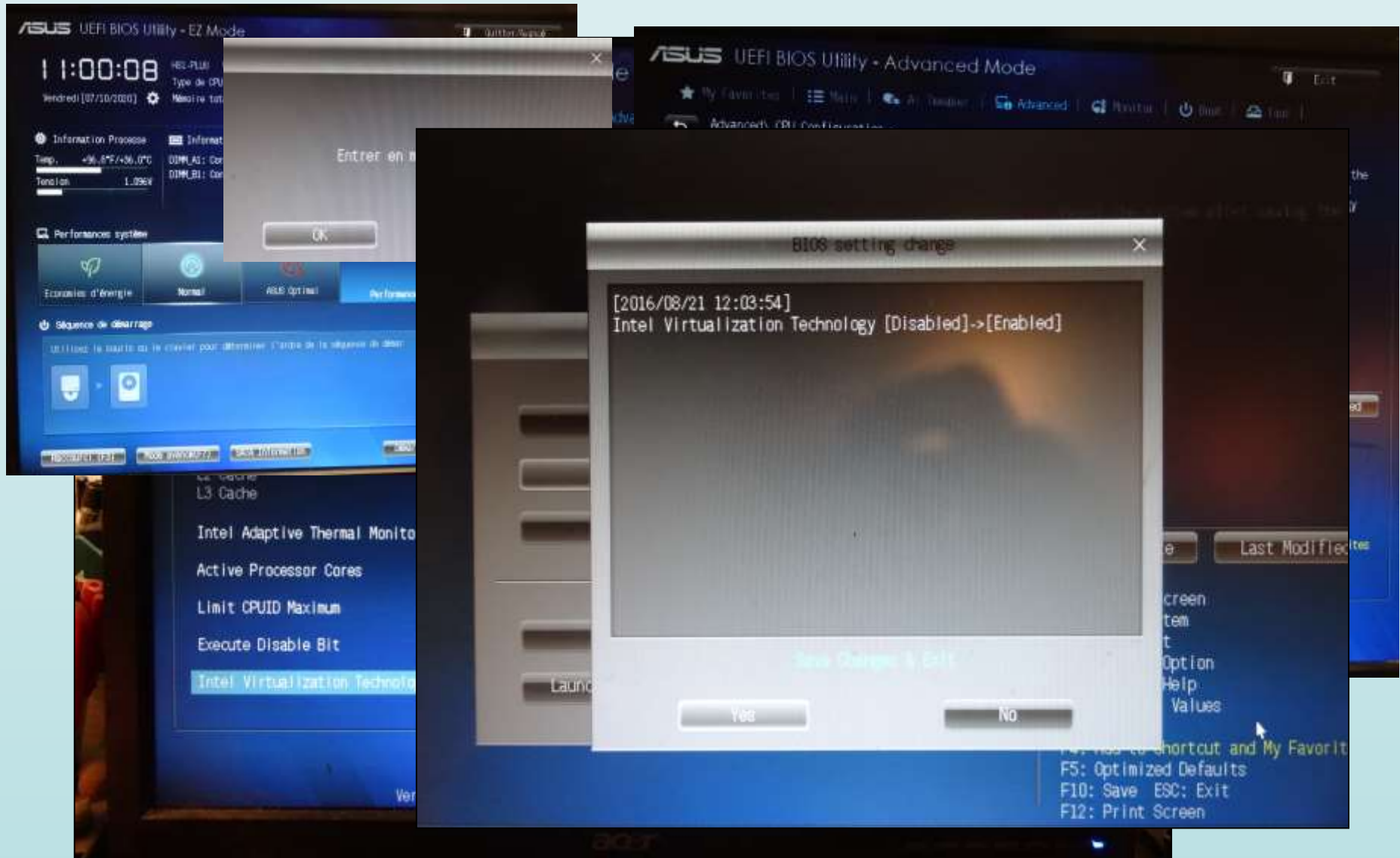
Google Nexus : appareils tournant sous le système Android produit par Google en coopération avec des constructeurs comme Motorola, Lenovo, Asus, Samsung, HTC.

8. L'installation de l'environnement Android (18/18)

Résultat :



grr ...





By now ...
... that's all folks ;-) !
Mais il y a encore à faire ...
➔ *développement proprement dit*