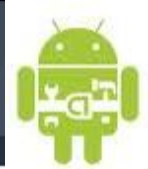
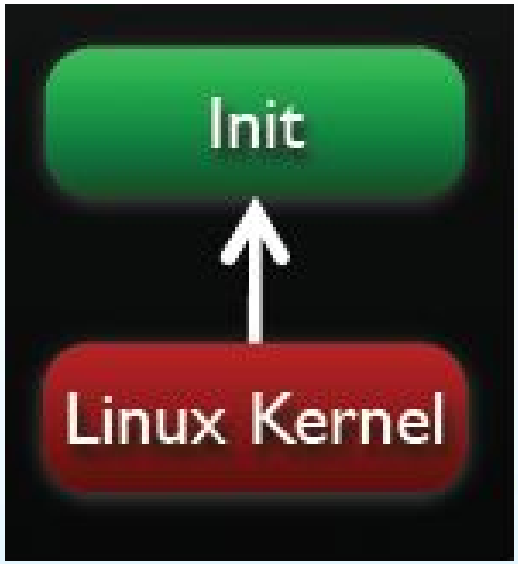
Partie II : Développement



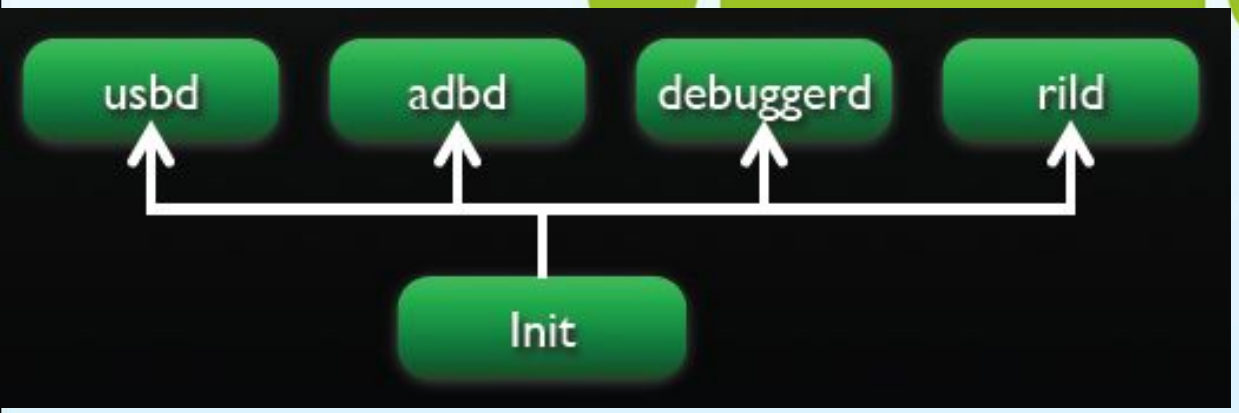
- OS ANDROID
- Android : Le challenge Android
- Structure d'une application ANDROID
- Exemple : « Hello World »
- Compilation et déploiement d'une application
- Cycle de vie d'une application
- Lancement sous activité
- Transmission d'information entre activités
- Démarrer une activité et obtenir un retour

Processus de démarrage des applications



Tout comme le système d'exploitation linux au démarrage de l'application, le boot loader charge le noyau (Kernel) et lance le processus init qui est le père de tous les processus

Processus de démarrage des applications



Comme nous pouvons le voir sur ce schéma, une fois le processus init lancé, lui a son tour lance des daemons

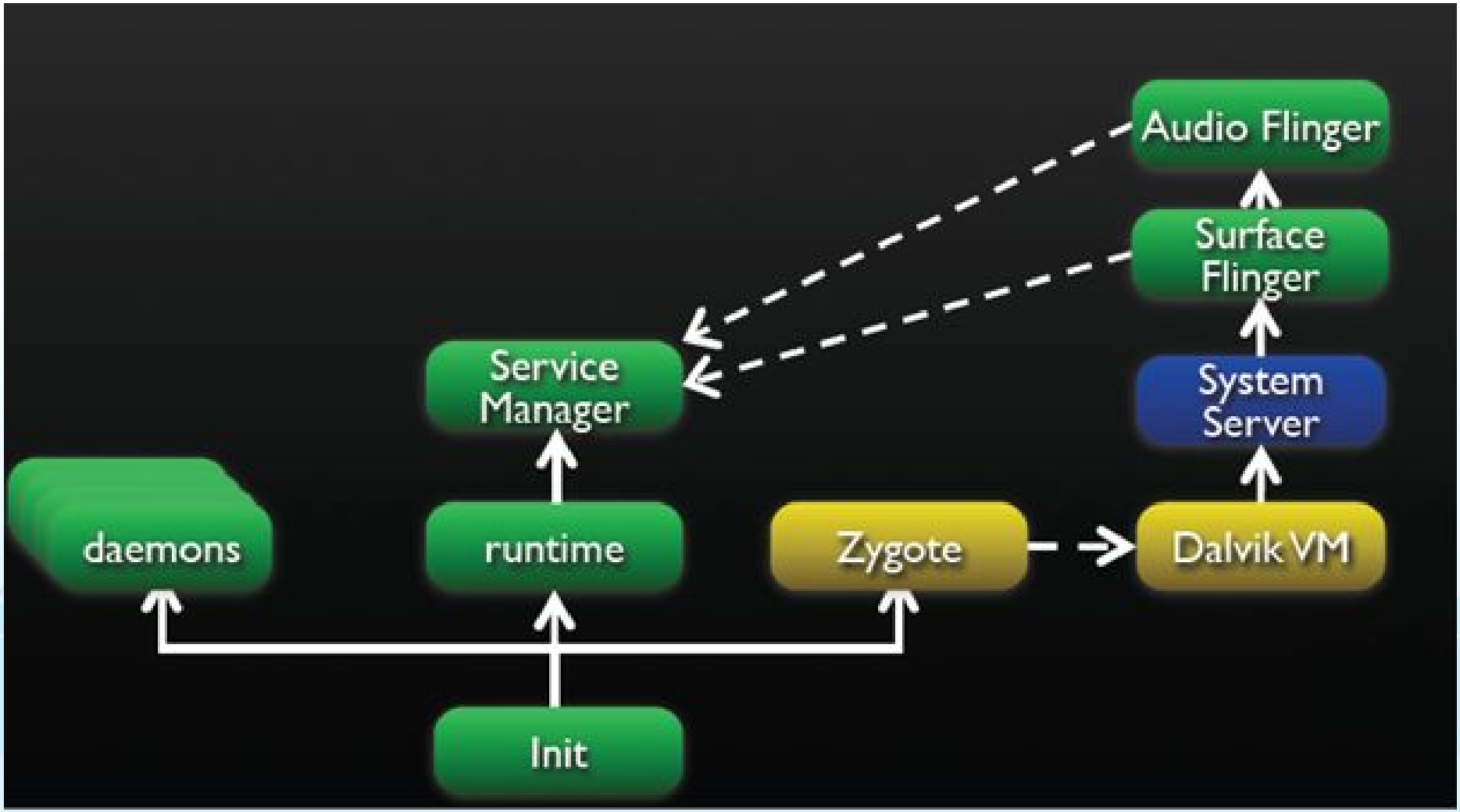
Processus de démarrage des applications

init lance le processus Zygote qui est le service le plus important car c'est à partir de lui que tous les autres services sont lancés.

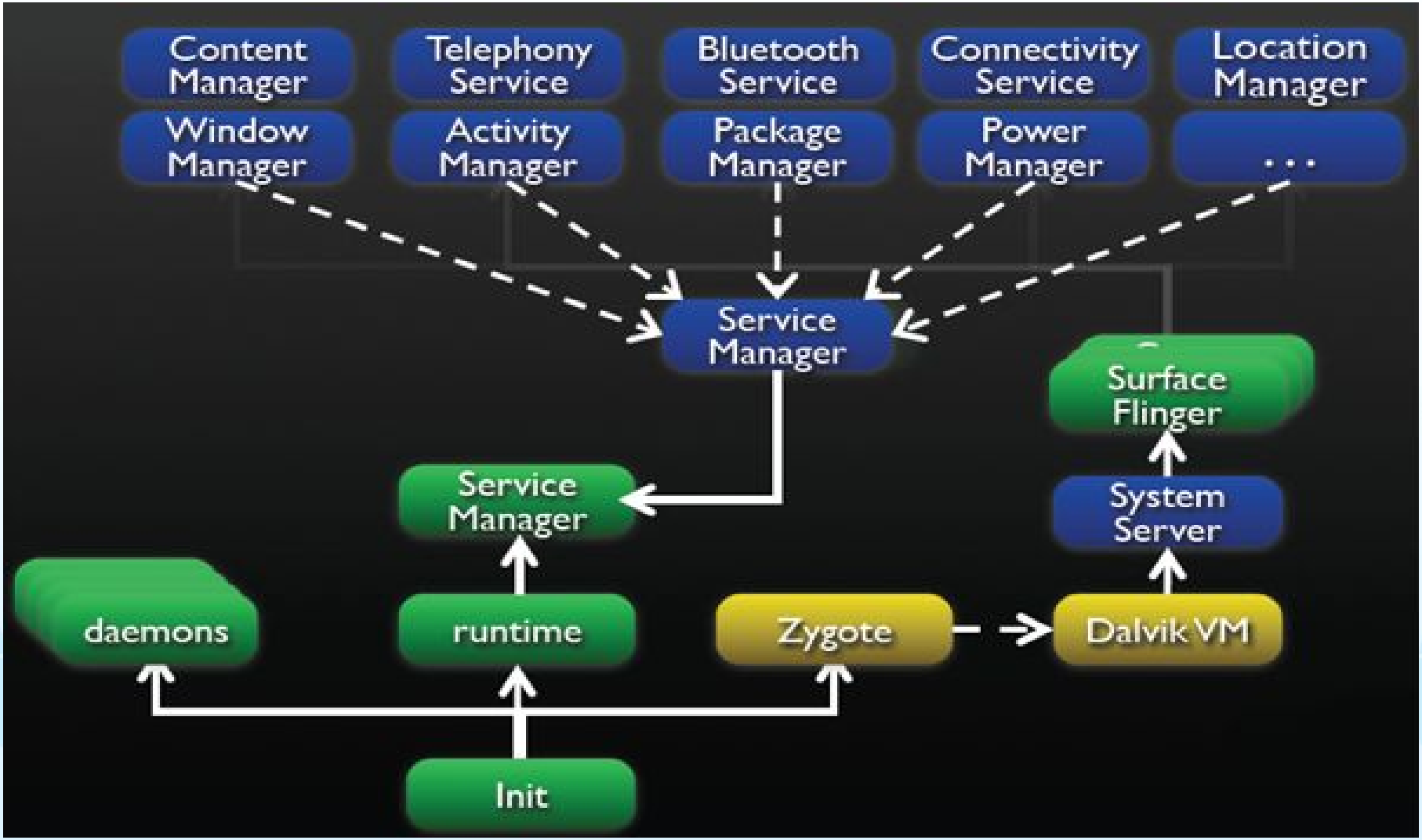
Processus de démarrage des applications

- Initialiser une instance de la Machine Virtuelle (VM) Dalvik.
- Pré charger les classes et écouter sur un socket pour créer des Dalvik VM
- Faire des Fork sur demande pour créer des instances de Dalvik VM pour chaque application. Les VM créées partagent des zones mémoire communes ce qui permet de minimiser la mémoire utilisées. Chaque VM créer par le processus zygote est un fork d'une VM "mère", ce qui permet d'accélérer le démarrage d'une application.

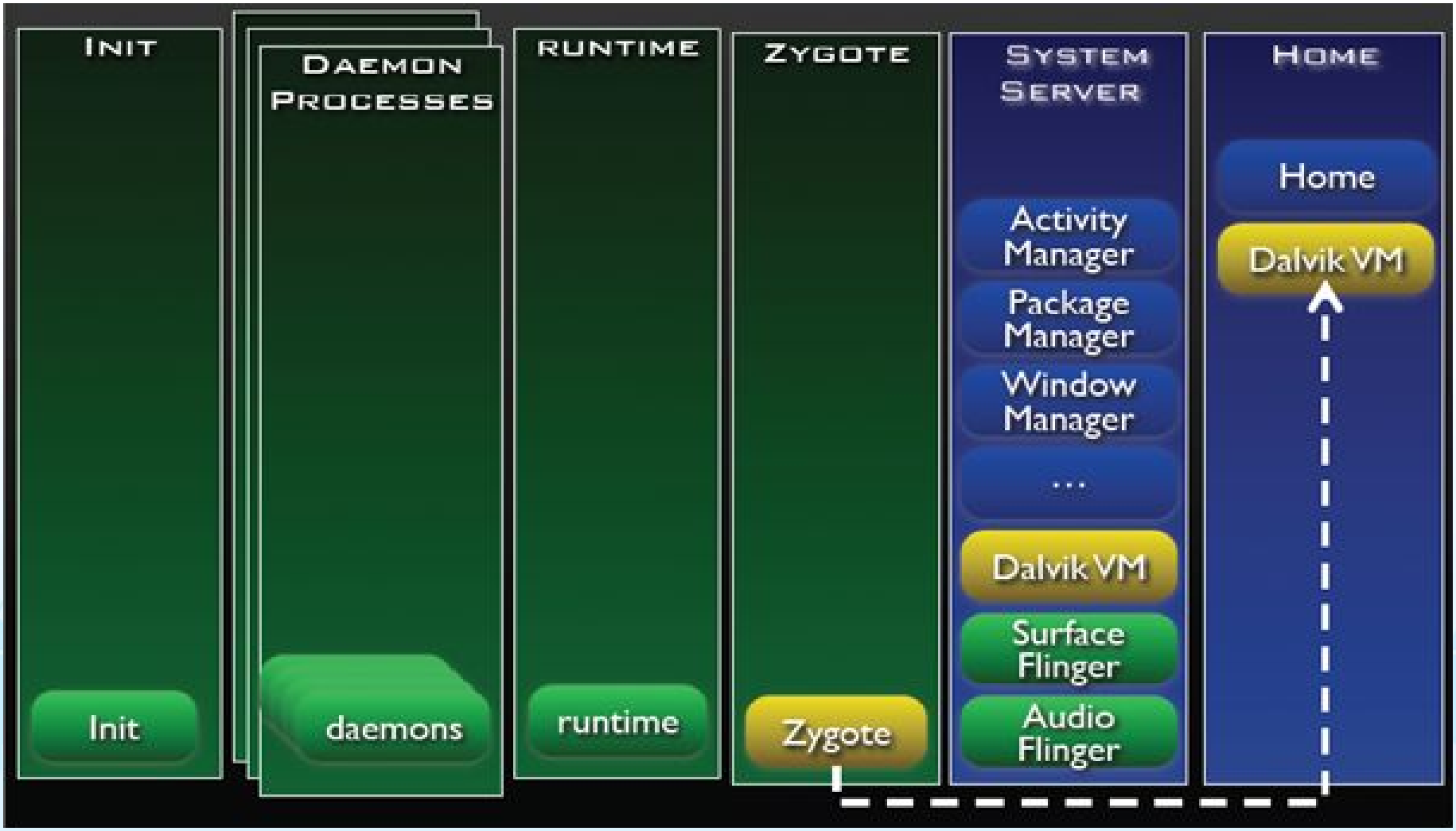
Processus de démarrage des applications



Processus de démarrage des applications

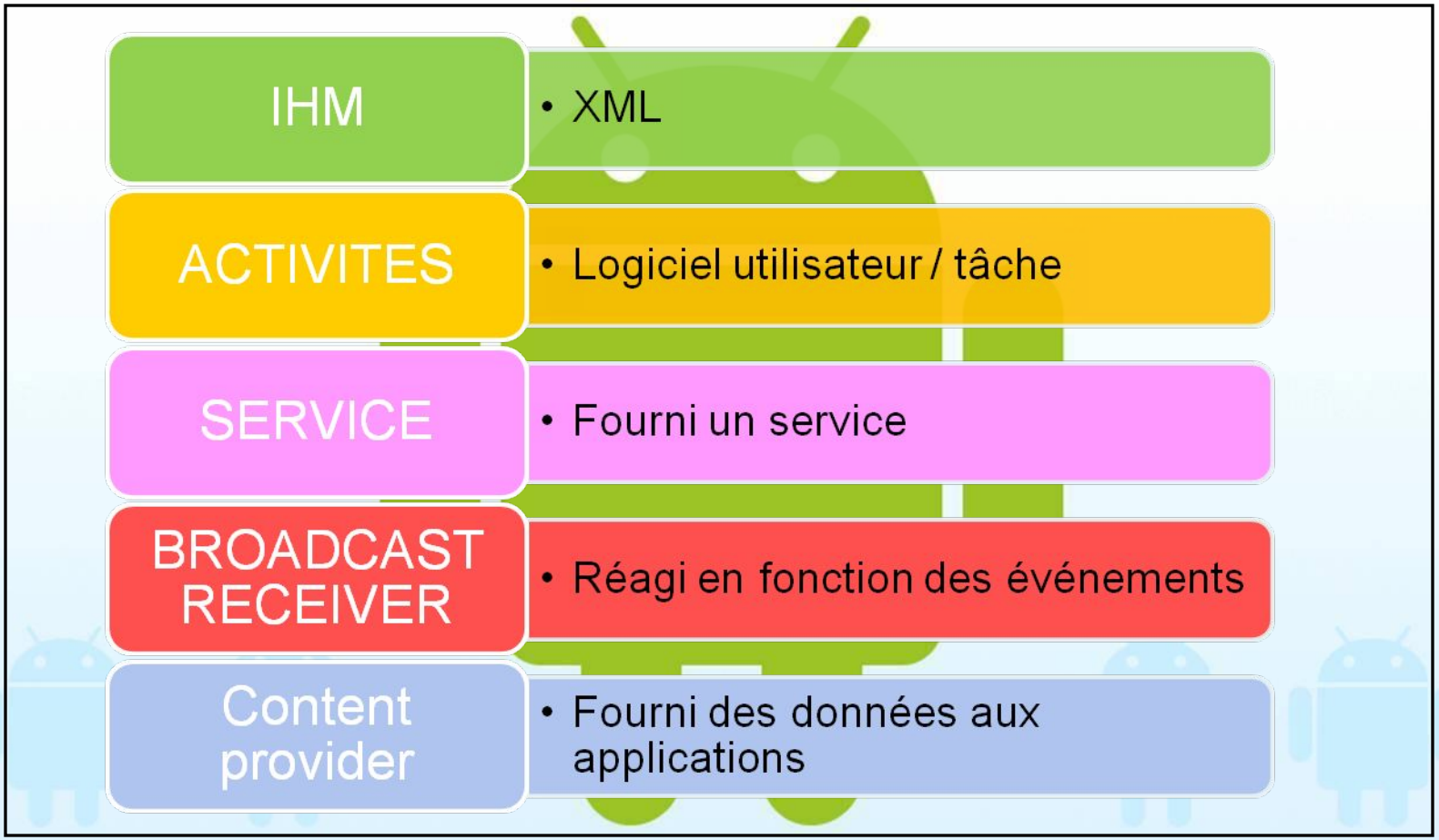


Processus de démarrage des applications



Structure d'une application ANDROID

FRAMEWORK ANDROID



Activités

- Une application peut avoir plusieurs activités par exemple pour la messagerie : on peut avoir une activité qui se charge d'afficher la liste de contact, une autre qui se charge d'afficher une zone de texte pour écrire un message à un contact, une autre pour lire les messages reçus etc.
- Dans le cas de l'application de messagerie, il existe une activité principale à partir de laquelle on démarre toutes les autres applications. Chaque activité a une fenêtre visuelle par défaut dont le contenu est fourni par l'hiérarchie des vues. Cette hiérarchie des vues est placées dans la fenêtre de l'activité grâce à la méthode setContentView (). Le Content View est l'objet racine de la hiérarchie.

Structure d'une application ANDROID

Activity

Service

- C'est un programme qui tourne en arrière plan, qui ne possède pas de GUI et n'a aucune interaction avec les utilisateurs. Chaque service hérite de la classe de base SERVICE.
- Certaines activités ont besoin de fonctionner en arrière plan au cas où on a besoin de lancer d'autres activité (l'activité est liés à un interface graphique). Pour ce faire il faudrait donc à partir d'une activité lancer un service qui tournera en arrière plan qui se chargera de continuer à jouer de la musique comme exemple écouter de la musique et lire un SMS.
- La communication avec un service se fait par des interfaces qu'elles fournissent, par exemple on peut communiquer avec un service qui multimédia qui offre une interface de commande : lecture, stop …

Broadcast Receiver

- C'est un composant qui ne fait rien mais reçoit et réagit aux annonces broadcasté. Il n'affiche rien à l'écran mais peut démarrer une activité en réponse à une information reçu ou peut utiliser le Notification Manager pour alerter l'utilisateur. Comme exemple on peut citer le changement de langue que l'on doit annoncer à toutes les applications. Une application peut également envoyer un broadcast à des applications qui aurait besoin d'utiliser des services ou des ressources qui ont été mise à jour.

- Une application peut avoir plusieurs Broadcast Receiver qui héritent de la classe BroadcastReceiver correspondant aux annonces qu'elle considère importantes.

Content Provider

- Elle rassemble l'ensemble de données d'une application disponible pour d'autres applications. Les données peuvent être sauvegardées dans les fichiers systèmes, dans la base de donnée SQLite ou autre.
- Il hérite de la classe de base Content Provider et implémente un ensemble de méthode qui permet à d'autres applications de sauvegarder et de récupérer des données. Par ailleurs cet ensemble de méthode n'est pas appelé directement par l'application, il utilise plutôt l'objet Content Resolver (pour l'interaction entre les différents processus) et fait appel à ses méthodes.

Intent

- Permet d'activer par des messages asynchrones des activités, des services et des Broadcast Receiver. Pour les activités et les services, elle décrit par une requête une action à effectuer et spécifie l'URI de la donnée sur laquelle on voudra agir comme par exemple fournir une zone de texte à l'utilisateur, ou encore afficher une image. Pour le Broadcast Receiver, l'objet Intent nomme l'action à annoncer. Il existe deux types d'Intent :

Intent explicite

- Intent explicite : désigne le composant cible par son nom (Component Name Field). Il est utilisé pour les messages internes à l'application par exemple lorsqu'une activité démarre un service ou une autre activité (de la même application).

Intent implicite

- Intent implicite qui ne désigne pas le composant cible par son nom. Dans ce cas le système doit choisir le composant qui gérera au mieux l'objet Intent (une activité ou un service pour effectuer l'action que décrit la requête ou l'ensemble de Broadcast Receiver qui correspond aux annonces de Broadcast). Pour ce faire il compare le contenu de l'objet Intent aux Intent Filter (gestionnaire des objets Intent qui permet de publier les capacités de l'application). Si un objet ne comporte pas d'Intent Filter, il ne peut recevoir que des Intent Explicite. Ils sont généralement utilisés pour activer des composants faisant partie d'autres applications.
- Afin d'informer le système de façon implicite quel Intent il doit gérer, les activités, les services, les Broadcasts Receiver peuvent avoir un ou plusieurs Intent Filter.

Environnement de développement

Environnement de développement

Pour le développement

→ https://developer.android.com/studio/install.html

- Plateforme 32 ou 64 bits pour Linux, Windows, Mac
- Bibliothèque de compatibilité 32 bits car une partie du SDK est sous 32 bits
  - sous Ubuntu 64 : `sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-1.0:i386`
  - `sous Ubuntu 14.04 :`

    sudo -i
    cd /etc/apt/sources.list.d
    echo "deb http://old-releases.ubuntu.com/ubuntu/ raring main restricted universe multiverse" > ia32-libs-raring.list
    apt-get update
    apt-get install ia32-libs

  - ou encore:
    - sudo dpkg --add-architecture i386
      sudo apt-get update
      sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
  - sous Fedora 64 : `sudo yum install zlib.i686 ncurses-libs.i686 bzip2-libs.i686`
- Java de Oracle (ne pas utiliser OpenJDK) version 1.7 ou supérieur
- Archive du SDK Android : http://developer.android.com/sdk/index.html
- Android Studio basé sur IntelliJ IDEA sur https://www.jetbrains.com/idea/

1) Installer Java et placer les binaires dans le PATH de votre environnement

2) Installer IntelliJ : http://developer.android.com/sdk/index.html

3) Installer le SDK en placant platforms, platform-tools, tools dans le PATH, car ces répertoires contiennent tous les utilitaires de compilation Android

4) Lancer IntelliJ ---> dans bin ./studio.sh

5) Utiliser le gestionnaire de SDK pour les mises à jour et le gestionnaire de machines virtuelles

Installation de l'environnement de développement

TP

- Suivre pas à pas l'installation de l'environnement
  - télécharger l'archive : IntelliJ → https://developer.android.com/sdk/index.html
  - télécharger java 1.7 → http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html
- Mettre à jour le PATH pour avoir des outils en ligne de commande
- Mettre à jour le SDK (cette opération peut être assez longue …)
- Configuration de votre projet de base avec le workspace fourni
- Parcourir le répertoire du SDK pour découvrir la structure

Installation de l'environnement de développement

Installation du matériel (smartphone, tablette)

Suivre la procédure : http://developer.android.com/tools/device.html

- Débrancher votre équipement

- Arrêter adb : "adb kill-server"

- Relancer le service udev:

  Ubuntu : "sudo service udev restart"
  Fedora, Ubuntu : "sudo udevadm control --reload-rules"
  sinon → sudo /etc/init.d/udev restart

- Démarrer adb : adb start-server

- Autoriser le mode DEBUG USB sur l'équipement (activer le mode développeur)

- Rebrancher votre équipement

- Vérifier avec : "adb devices" ou encore lsusb, ou dmesg

Installation de l'environnement de développement

Installation du matériel (smartphone, tablette)

Si la procédure précédente ne fonctionne pas :

- Débrancher votre équipement

- Arrêter adb : "adb kill-server"

- Placer un fichier adb_usb.ini dans le HOME directory qui contient :

```
nsy208@pc3-02:~/.android$ cat adb_usb.ini
# ANDROID 3RD PARTY USB VENDOR ID LIST -- DO NOT EDIT.
# USE 'android update adb' TO GENERATE.
# 1 USB VENDOR ID PER LINE.
0x2207
```

- Démarrer adb : adb start-server

- Rebrancher votre équipement

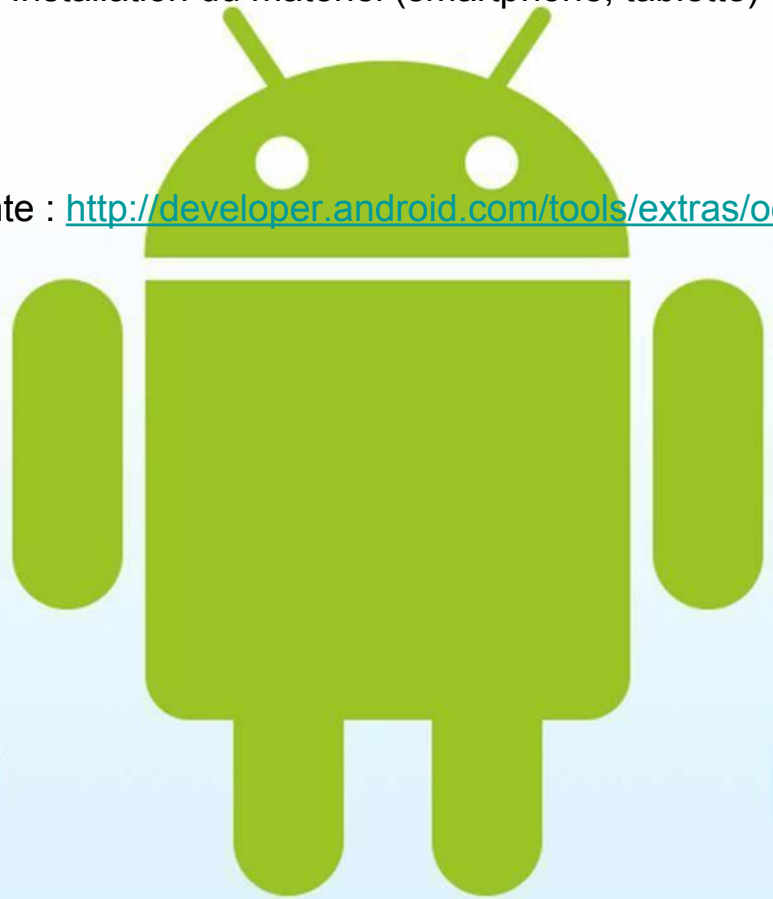
- Vérifier avec : "adb devices" ou encore lsusb, ou dmesg

Installation de l'environnement de développement

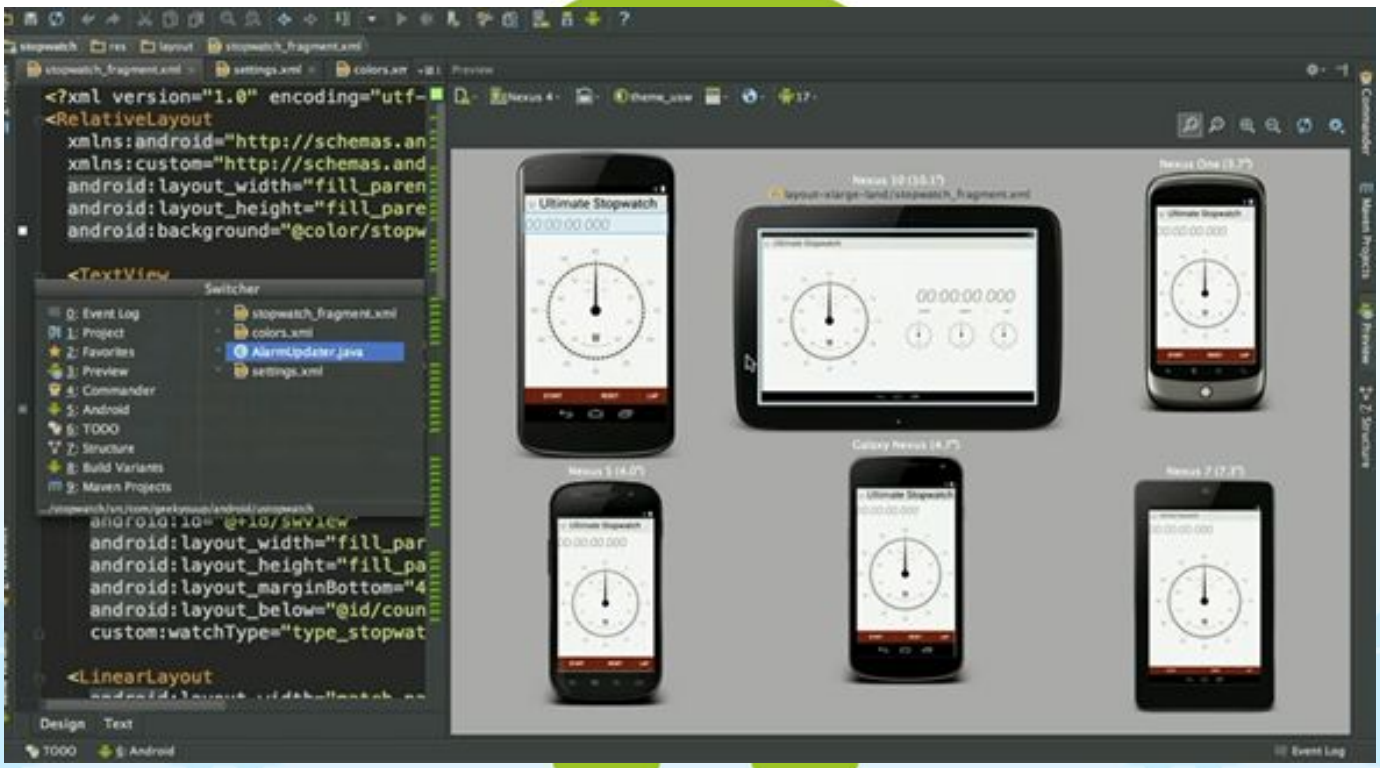
Installation du matériel (smartphone, tablette)

Sur Windows :

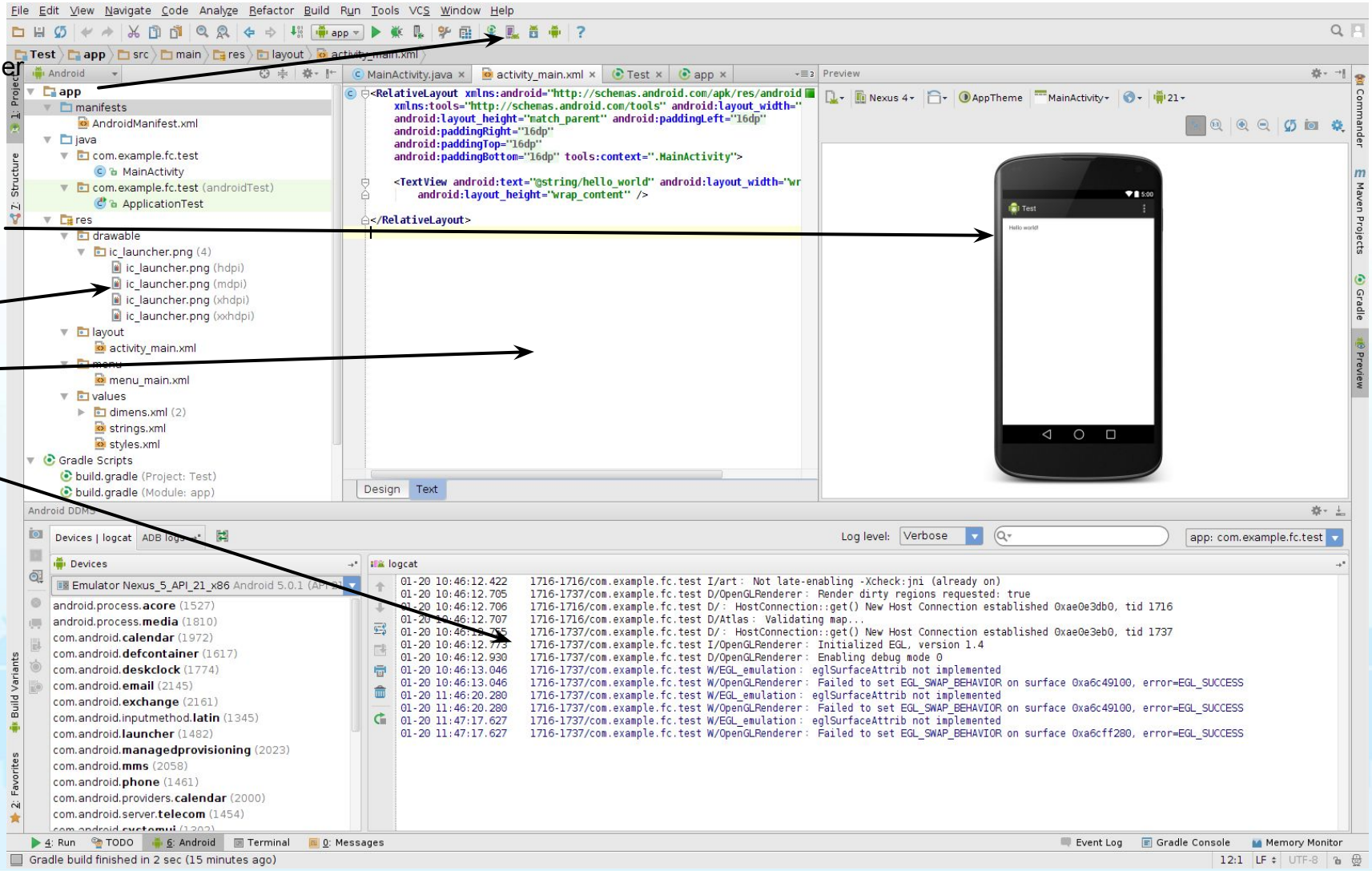
- Suivre la procédure suivante : http://developer.android.com/tools/extras/oem-usb.html



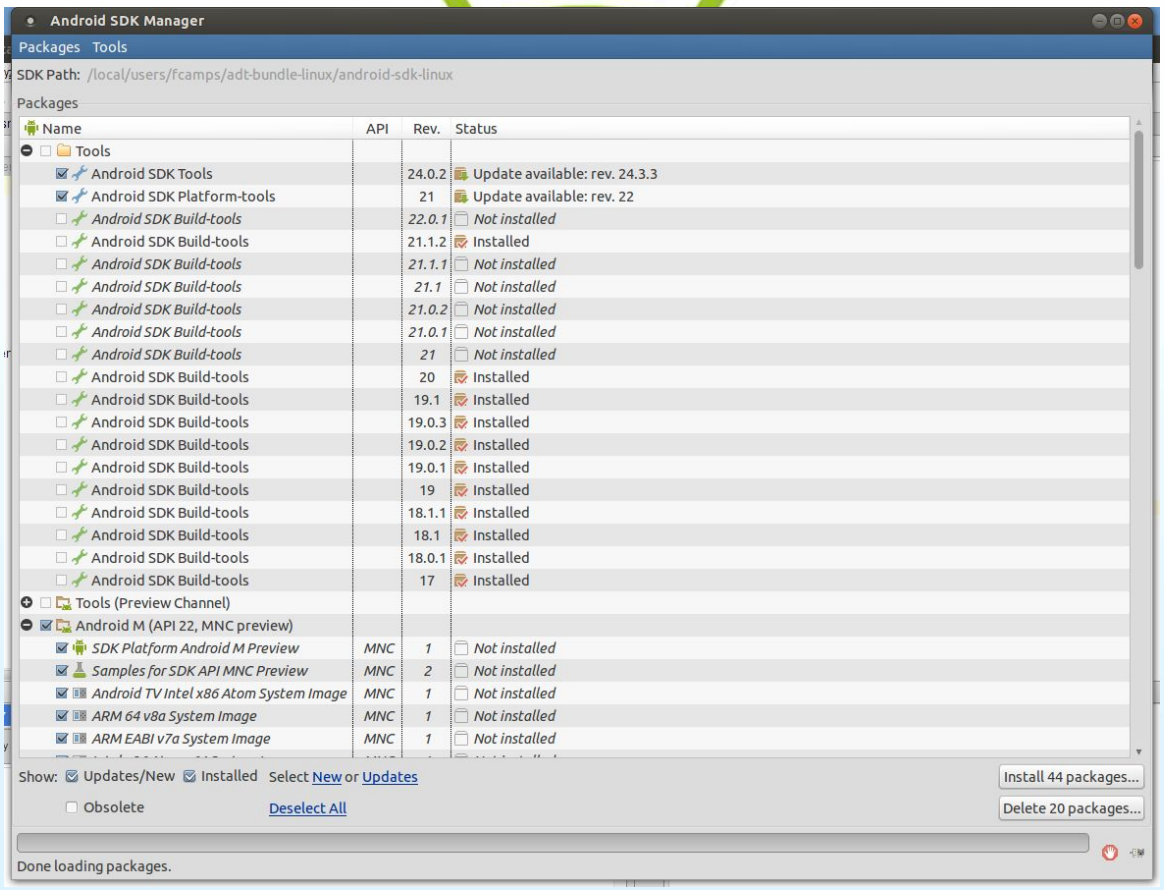
IntelliJ IDE



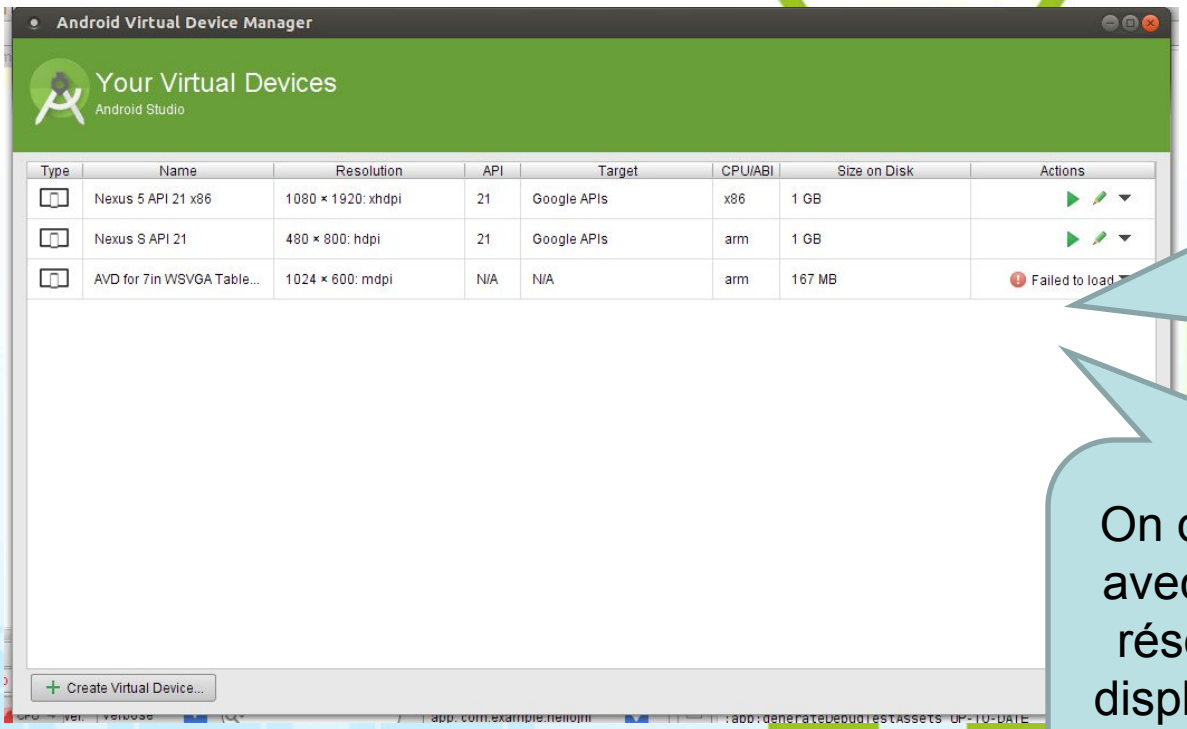
L'environnement de développement



Gestion du SDK



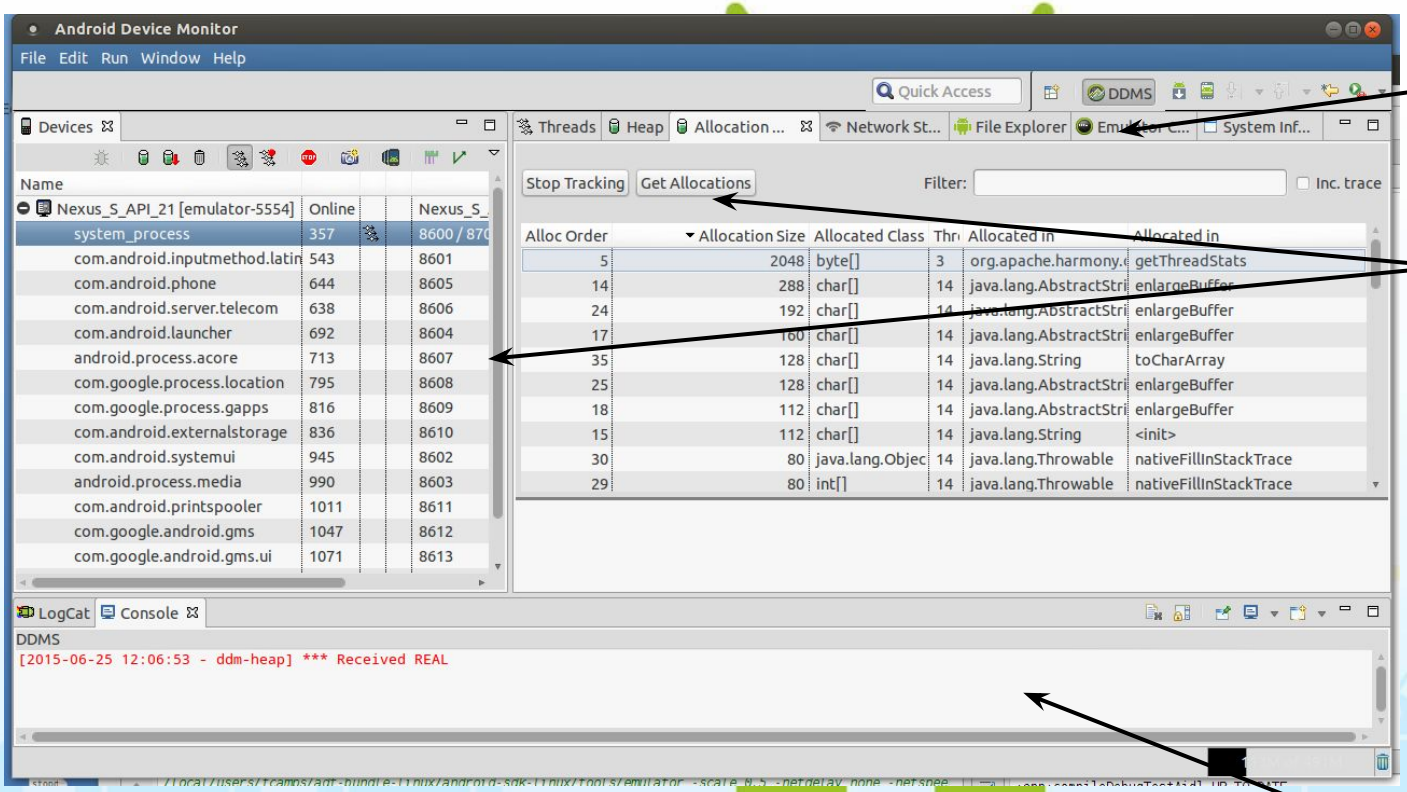
Gestion des machines virtuelles



Le gestionnaire de machine virtuelle permet de créer des téléphones virtuels dans lesquels ont peut ensuite exécuter les applications

On définir une SD card avec de la mémoire, la résolution et taille du display, la cible -> point très important

Perspective ADM (Android Device Manager)



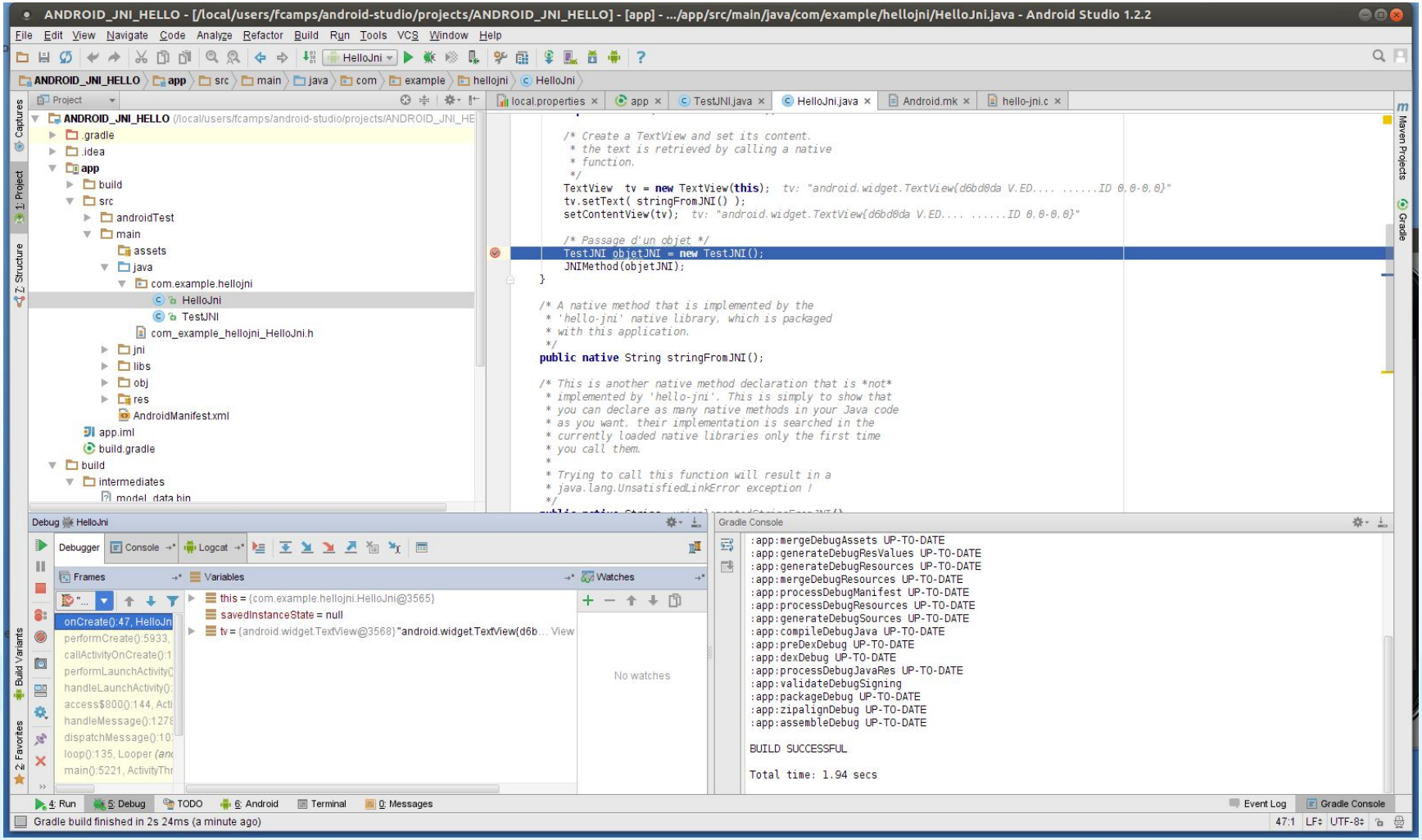
Simulating Incoming SMS Messages to the Emulator

Internal Thread

Logging

L'environnement de développement

Perspective DEBUG



Emulateur

http://developer.android.com/tools/help/emulator.html

Emulateur

http://developer.android.com/guide/developing/tools/emulator.html
http://developer.android.com/guide/developing/tools/adb.html

Liste des émulateurs :

```
[fcamps@discovery ~]$ adb devices
List of devices attached
emulator-5554   device
emulator-5556   offline
```

Shell :

```
[fcamps@discovery ~]$ adb -s emulator-5554 shell
# ps
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     268    180   c009b74c 0000875c S /init
root      2     0     0      0     c004e72c 00000000 S kthreadd
```

En ligne de commande dans le SDK :

tools/ : ensemble de scripts et commande binaire pour développer sous ANDROID, les principaux :

android — Création de propjet Android et gestion des machines virtuelles.

aapt (Android Asset Packaging Tool)— Transforme le projet en fichier APK pour l'émulateur ou les équiepements.

ddms (Dalvik Debug Monitor Service)—Equivalent au DDMS de Eclipse.

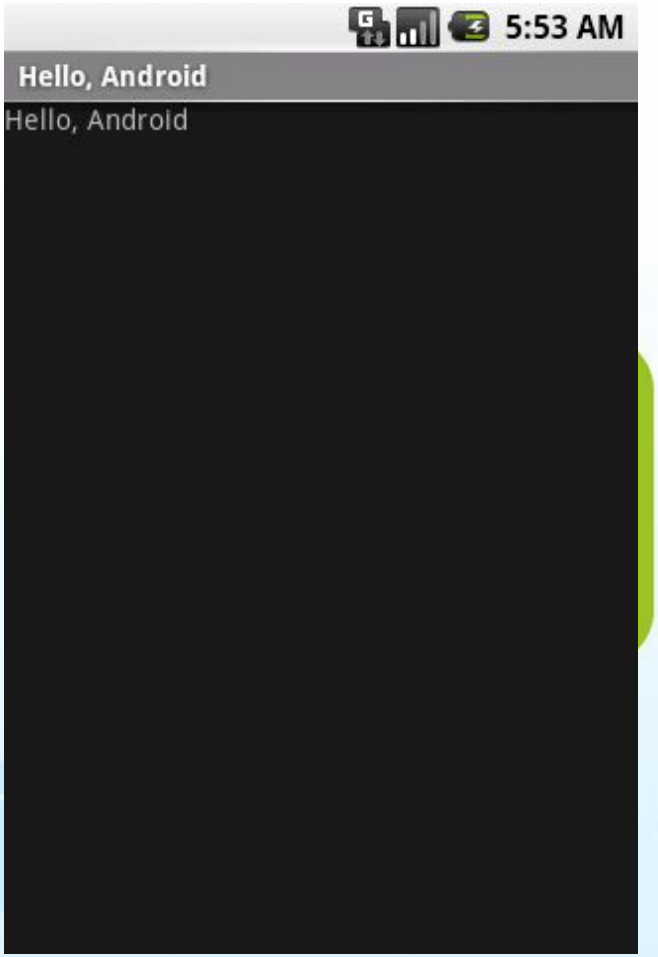
adb (Android Debug Bridge)—Utilitaire pour interagir avec l'émulateur ou l'équipement.

Structure d'une application ANDROID

Structure fichiers

- **src/** : Répertoire du code source *.java, *.c, *.cpp
- **manifest.xml** : Description XML du contenu de l'application, les composants (activités, services ..), les droits accordés aux applications.
- **defaultProperties.xml** : Proriétés utilisées par le script de construction Ant.
- **asset**/ : Répertoire des fichiers statiques
- **bin**/ : Répertoire qui contient l'application compilée .apk
- **gen**/ : Répertoire qui contient le ficher des références R.java.
- **res**/ : Répertoire qui contient toutes les ressources du projet, interface graphique(layouts, images, raw).

Exemple : « Hello World »



Nous allons maintenant développer une petite application que nous testerons avec l'émulateur

Exemple : « Hello World » : code source

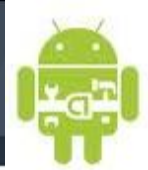
Code Java

```
package com.android.hello;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity {
 /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

        // on doit décrire un layout qui contiendra
        // dans notre cas un fond noir avec
        // l'affichage d'un String = "Hello, Android
        // puis on charge le layout :

       setContentView(R.layout.main);
    }
}
```

Exemple : « Hello World » : code source


```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
      <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Exemple : « Hello World » : fichiers XML



Fichier ressource

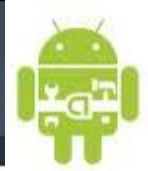
```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
      <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">hello</string>
     <string name="hello_world">Hello
world!</string>

</resources>
```

String.xml

Exemple : « Hello World » :Manifest



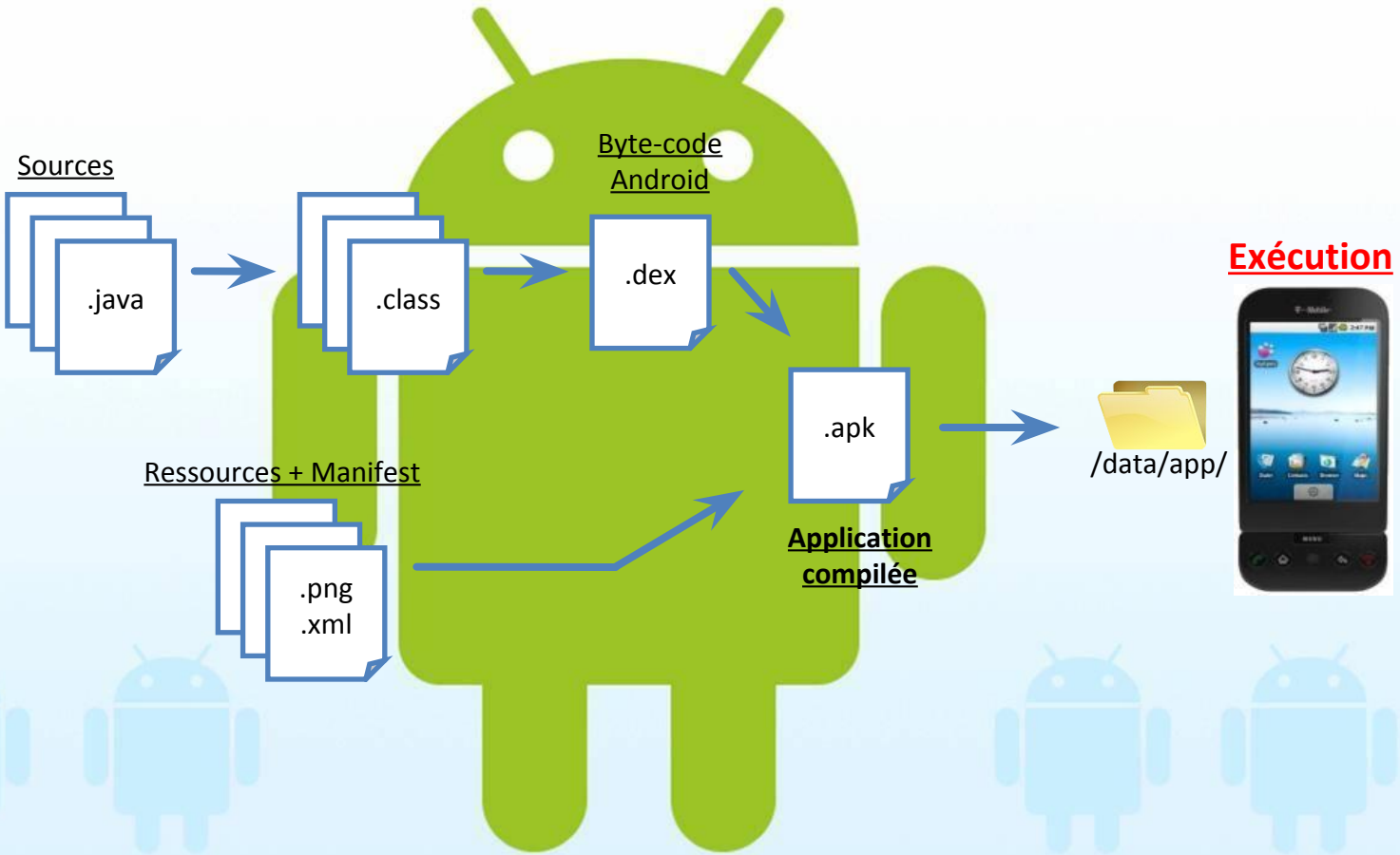
AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.android.hello"
      android:versionCode="1"
      android:versionName="1.0.0">
  <application android:icon="@drawable/icon"
android:label="@string/app_name">
    <activity android:name=".HelloAndroid"
              android:label="@string/app_name">
            <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
    </activity>
  </application>
</manifest>
```

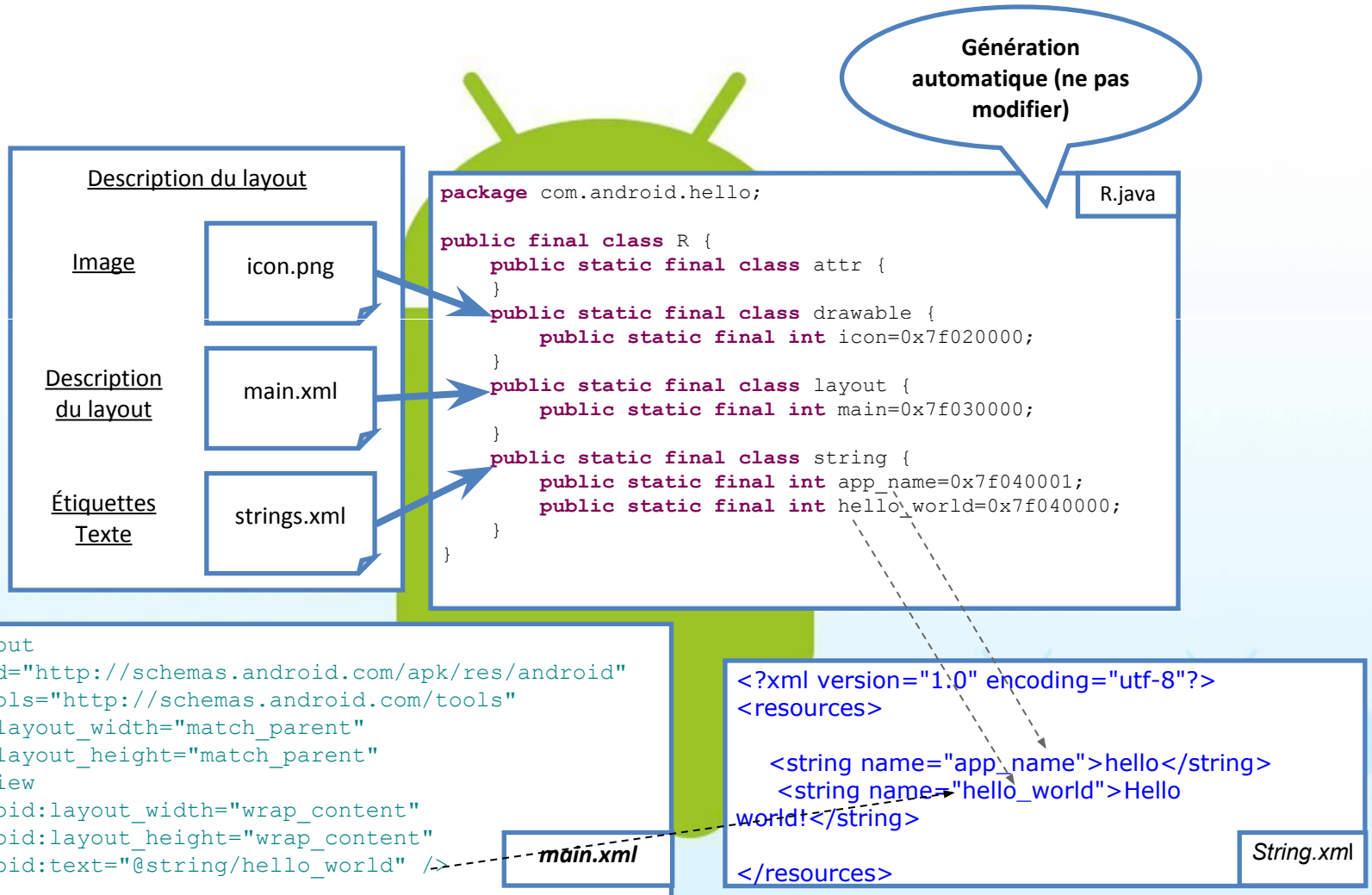
Décrire les composants de l'application :

- Les activités
- Les services
- Les permissions
- Les intents
- Les broadcasts receivers
- Etc. …

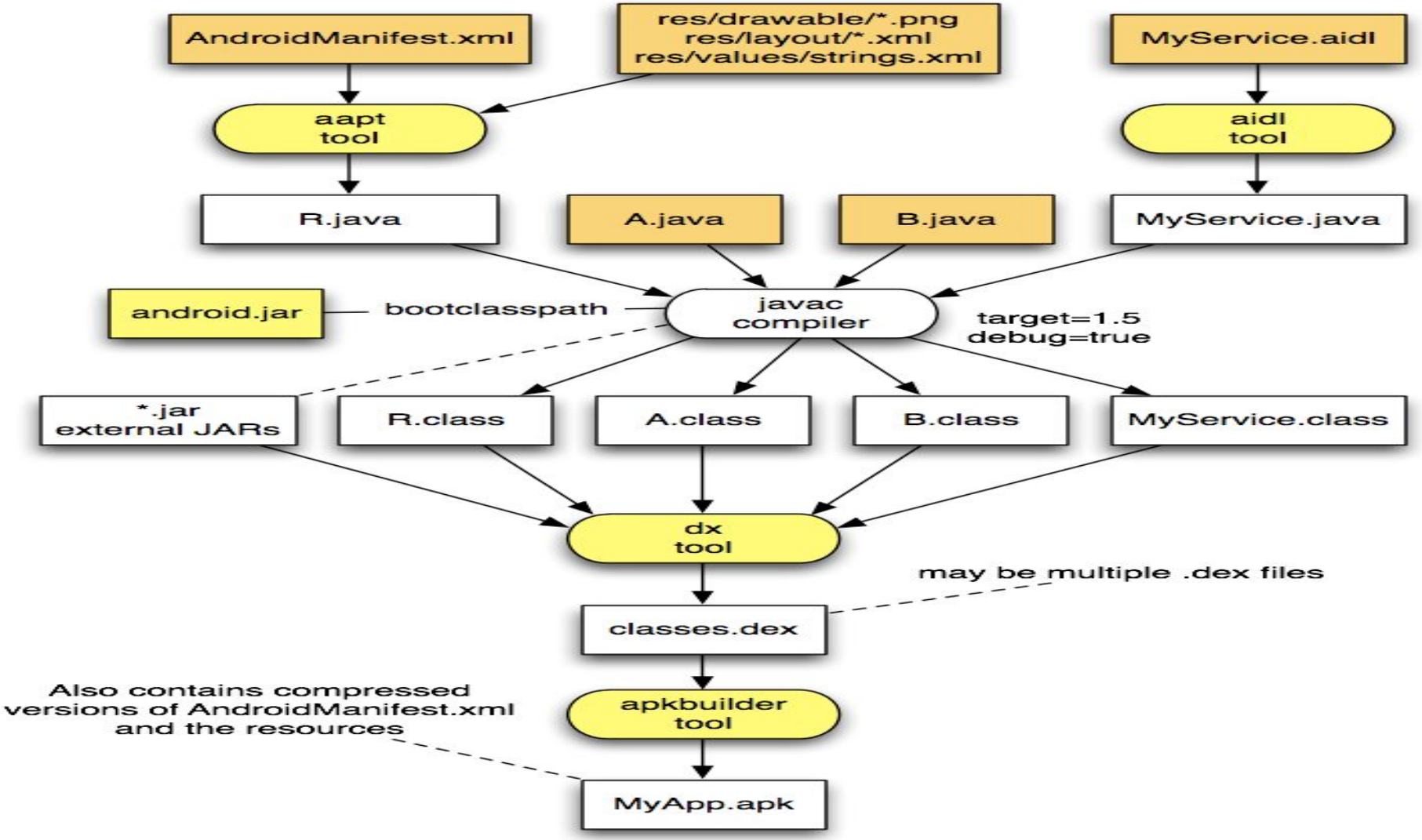
Exemple : « Hello World » - Compilation et déploiem



Exemple : « Hello World » : R.java



Compilation et déploiement d'une application



RUN TIME

DALVIK

Depuis quelques versions de Dalvik, l'application est traduite de son langage intermédiaire (le bytecode) en langage machine spécifique au processeur physique au moment de son exécution : c'est ce qu'on nomme généralement JIT, pour Just In Time, ou littéralement « juste à temps ».

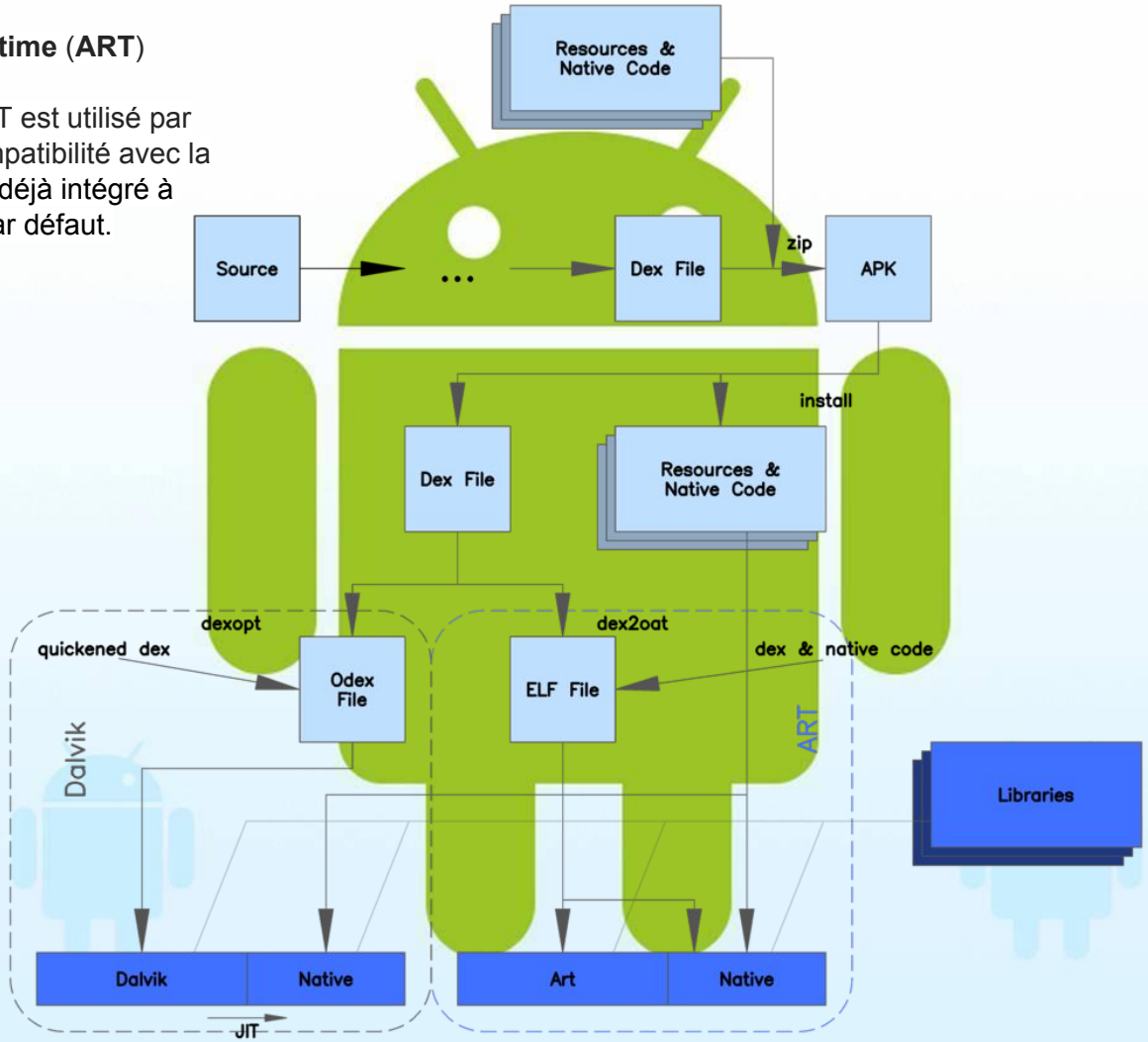
Android Runtime (ART)

La machine virtuelle Dalvik, comme Java, permet de faire un code fonctionnel sur tous les environnements qui l'exécutent, sans tenir compte des instructions requises de chaque matériel, pourquoi s'embêter à recompiler à chaque fois ? C'est justement ce que va résoudre l'ART qui implémente dès sa conception le principe AOT, Ahead-Of-time. L'application sera donc compilée dès lors qu'elle se verra installée sur le matériel évitant la conversion à chaque exécution.

Compilation et déploiement d'une application

DALVIK et Android Runtime (ART)

A partir de Android 5, ART est utilisé par défaut et maintient la compatibilité avec la machine Dalvik. ART est déjà intégré à KitKat, mais pas activé par défaut.



TP: "hello World!"

- Création d'une application:
  - Wizard,
  - Codage,
  - Déploiement sur émulateur, smartphone.

- Découverte de l'environnement:
  - IDE,
  - Emulateur,
  - DDMS,
  - Gradle,
  - SDK manager,
  - Composition du SDK,
  - Ligne de commande en dehors de IntelliJ.

Logging dans le DDMS

Commonly Used Log Methods

Log.e() Logs errors

Log.w() Logs warnings

Log.i() Logs informational messages

Log.d() Logs debug messages

Log.v() Logs verbose messages

Excessive use of the Log utility can result in decreased application performance. Debug and verbose logging should be used only for development purposes and removed before application publication.

Logs are visible in DDMS/Logcat

Commonly Used Log Methods

- The first parameter of each Log method is a string called a tag :

private static final String *TAG = "MyApp";*

Log.*i(TAG, "In onCreate() callback method");*

- You can use the LogCat utility from within Eclipse to filter your log messages to the tag string. See Appendix B, "Eclipse IDE Tips and Tricks," for details.

Cycle de vie d'une activité

Cycle de vie d'une activité

```
public  class ExampleActivity extends Activity {
@Override public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
       // The activity is being created.
 }
@Override protected void onStart() {
super.onStart();
// The activity is about to become visible.
 }
@Override protected void onResume() {
 super.onResume();
// The activity has become visible (it is now "resumed"). }
@Override protected void onPause() {
super.onPause();
// Another activity is taking focus (this activity is about to be
"paused"). }
@Override protected void onStop() {
super.onStop();
// The activity is no longer visible (it is now "stopped") }
@Override protected void onDestroy() {
super.onDestroy();
// The activity is about to be destroyed.
} }}
```

C'est la structure de base pour une activité. Toutes les méthodes peuvent être « écrasées »

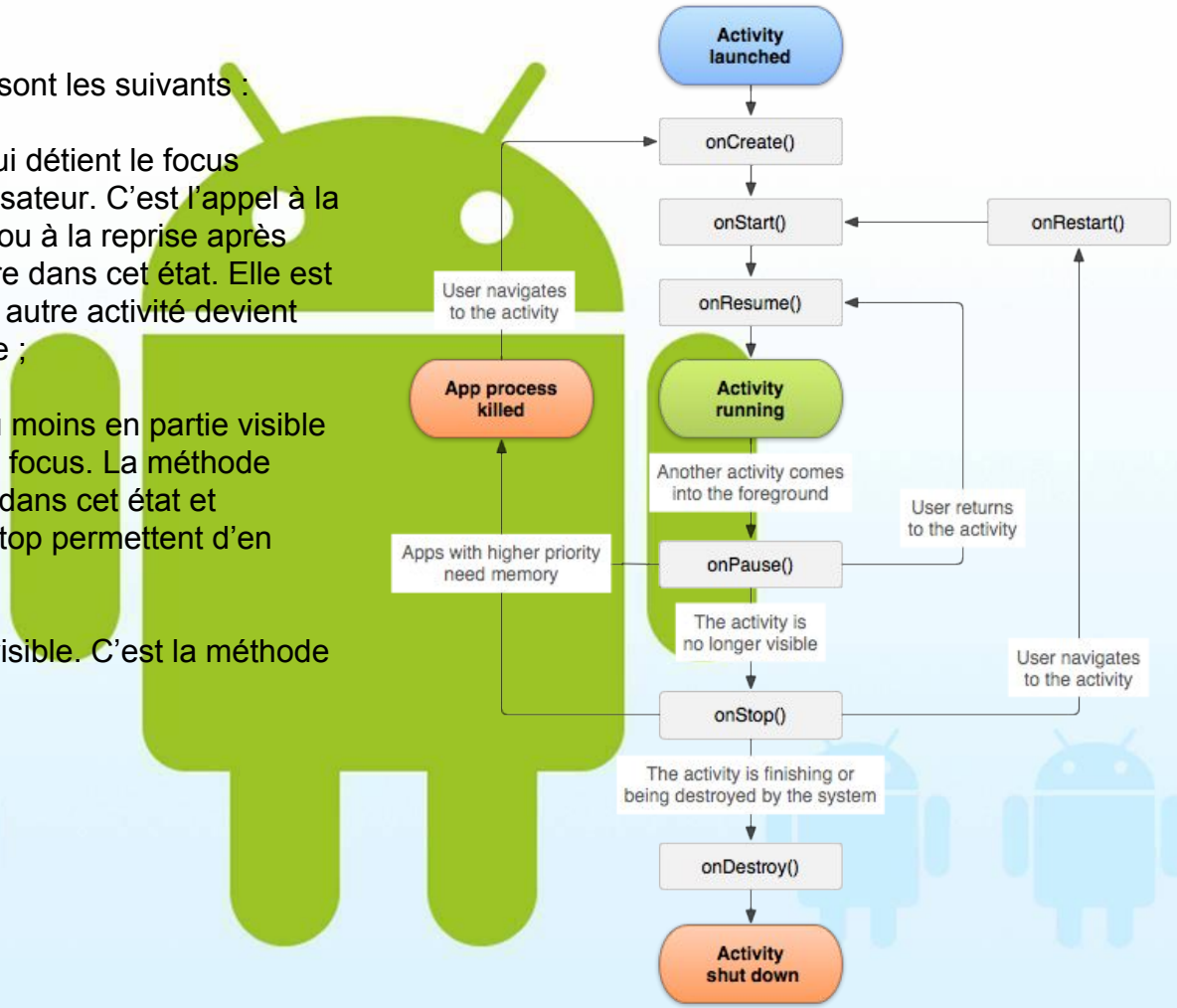
Cycle de vie d'une activité : Diagramme d'état

Les états principaux d'une activité sont les suivants :

• **active** (active) : activité visible qui détient le focus utilisateur et attend les entrées utilisateur. C'est l'appel à la méthode onResume, à la création ou à la reprise après pause qui permet à l'activité d'être dans cet état. Elle est ensuite mise en pause quand une autre activité devient active grâce à la méthode onPause ;

• **suspendue** (paused) : activité au moins en partie visible à l'écran mais qui ne détient pas le focus. La méthode onPause est invoquée pour entrer dans cet état et les méthodes onResume ou onStop permettent d'en sortir ;

• **arrêtée** (stopped) : activité non visible. C'est la méthode onStop qui conduit à cet état.



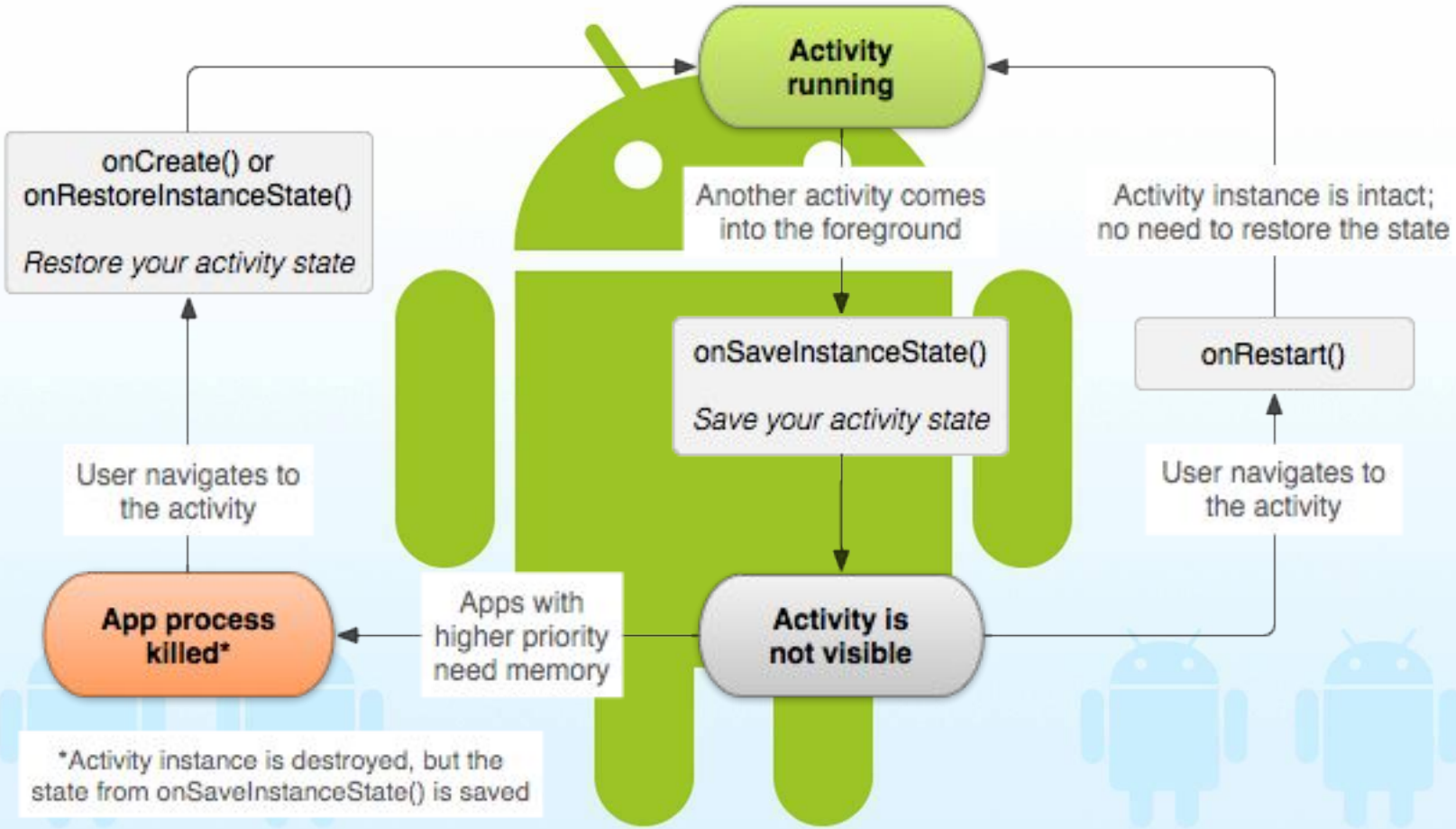
Cycle de vie d'une activité

Method	Description	Killable?	Next
onCreate()	Called when the activity is first created. This is where you should do all of your normal static set up: create views, bind data to lists, etc. This method also provides you with a Bundle containing the activity's previously frozen state, if there was one. Always followed by onStart().	No	onStart()
onRestart()	Called after your activity has been stopped, prior to it being started again. Always followed by onStart()	No	onStart()
onStart()	Called when the activity is becoming visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.	No	onResume() or onStop()
onResume()	Called when the activity will start interacting with the user. At this point your activity is at the top of the activity stack, with user input going to it. Always followed by onPause().	No	onPause()
onPause()	Called when the system is about to start resuming a previous activity. This is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, etc. Implementations of this method must be very quick because the next activity will not be resumed until this method returns. Followed by either onResume() if the activity returns back to the front, or onStop() if it becomes invisible to the user.	Yes	onResume() or onStop()

Cycle de vie d'une activité

Method	Description	Killable?	Next
onStop()	Called when the activity is no longer visible to the user, because another activity has been resumed and is covering this one. This may happen either because a new activity is being started, an existing one is being brought in front of this one, or this one is being destroyed. Followed by either onRestart() if this activity is coming back to interact with the user, or onDestroy() if this activity is going away.	**Yes**	onRestart() or onDestroy(
onDestroy()	The final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called finish()The final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called finish() on it, or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the isFinishing() method.	**Yes**	*nothing*

Cycle de vie d'une activité : Diagramme d'état



http://developer.android.com/guide/components/activities.html

Cycle de vie d'une activité : Diagramme d'état

Destruction d'une activité par le système

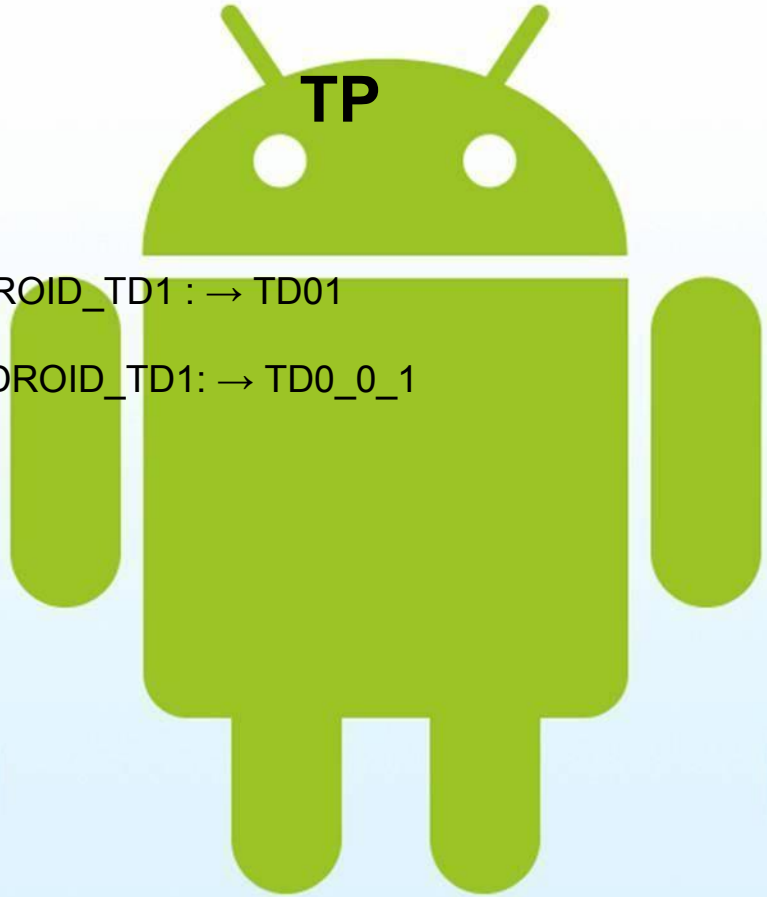
Lorsque le système détruit une activité pour récupérer de la mémoire, l'objet activité est détruit :

- Le système est obligé de recréer un objet lorsque l'utilisateur utilise back dans la navigation. L'utilisateur ne sait pas que l'activité a été détruite et recrée. L'utilisateur s'attend donc à retrouver son application dans le même état.
- Pour retrouver l'état de l'application il faut surcharger la méthode onSaveInstanceState()
- Pour récupérer l'état il faut utiliser un Bundle qui est un conteneur qui enregistre les états des variables, ensuite dans OnCreate() on récupére l'état précédent puis on initialise toutes les structures. (Voir TD1)

- **Sauvegarde des IHM, par défaut** : si l'on ne surcharge pas onSaveInstanceState(), alors l'implementation par défaut de onSaveInstanceState() permet de sauvegarder l'état des interfaces graphiques, car chaque chaque widget possède une onSaveInstanceState() qui est appelé. Par exemple il est possible de sauvegarder l'état d'un checkBox (checked ou non), un EditTest sauvegarde le texte saisi. Le seul travail à fournir est de donner un id unique à chaque widget avec l'attribut android:id, par exemple <TextView android:id="@+id/nameTextbox"/> dans le fichier manifest.xml. Si l'ID n'est pas défini alors le contexte du widget n'est pas sauvegardé par défaut.

- **Test** : Pour tester la restauration des paramètres de l'application, il suffit de faire une rotation de l'équipement. Car au moment de la rotation, le systeme détruit l'application pour l'ouvrir dans un nouveau mode. Dans ce cas il est très important que l'application sauvegarde son état.

Cycle de vie d'une activité : Diagramme d'état



Etat d'une activité: ANDROID_TD1 : → TD01

Etat + restauration : ANDROID_TD1: → TD0_0_1

La classe Intent

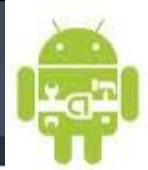
Au coeur du système Android, les Intents forment un mécanisme sophistiqué et complet permettant aux activités et aux applications d'interagir entre elles.

Les objets Intent ont essentiellement trois utilisations, ils permettent :

1. Démarrer une activité au sein de l'application courante
2. Solliciter d'autres applications
3. Envoyer des informations entre application

http://developer.android.com/reference/android/content/Intent.html
http://developer.android.com/guide/components/intents-filters.html

La classe Intent



1 Démarrer une activité

```
Intent intent = new Intent(this,ActiviteADemarrer.class); // ici explicite
startActivity(intent);
```

Le constructeur de la classe Intent prend les paramètres suivants :

• Context PackageContext : le contexte à partir duquel l'Intent est créé et sera envoyé. Ce paramètre fait référence la plupart du temps à l'activité en cours pointée par le mot clef this ;

• Class<?> cls *: un type de classe Java héritant de la classe Activity et pointé ici* par le mot clef ActiviteADemarrer.class.

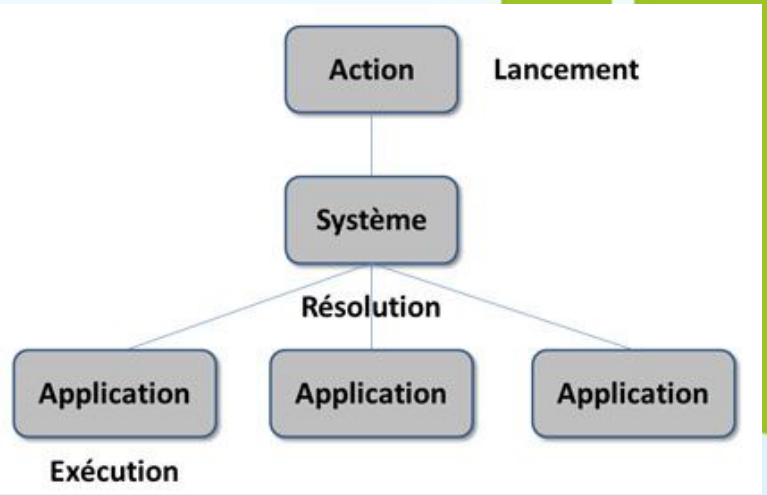
Déclarer une activité dans AndroidManifest.xml :

```
<application ...>
<activity android:name=".MonActivite" />
</application>
```

La classe Intent

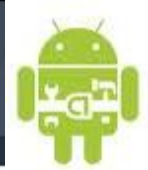
2 Solliciter d'autres applications

- En effet, l'envoi d'un Intent permet également de demander à un composant d'une autre application que la vôtre de traiter l'action que vous souhaiteriez réaliser
- Le système se base sur les informations que vous spécifiez dans votre objet Intent : action, données, catégorie
- Ce mécanisme permet d'éviter les dépendances vers des applications




```
Uri uri = Uri.parse("tel:0612345678");
Intent intent = new Intent(Intent.ACTION_DIAL, uri);
startActivity(intent)
```

La classe Intent



Les actions natives (http://developer.android.com/reference/android/content/Intent.html)

Action Définition	Action Définition
ACTION_ANSWER.	Prendre en charge un appel entrant.
ACTION_CALL	Appeler un numéro de téléphone. Cette action lance une activité affichant l'interface pour composer un numéro puis appelle le numéro contenu dans l'URI spécifiée en paramètre.
ACTION_DELETE	Démarrer une activité permettant de supprimer une donnée identifiée par l'URI spécifiée en paramètre.

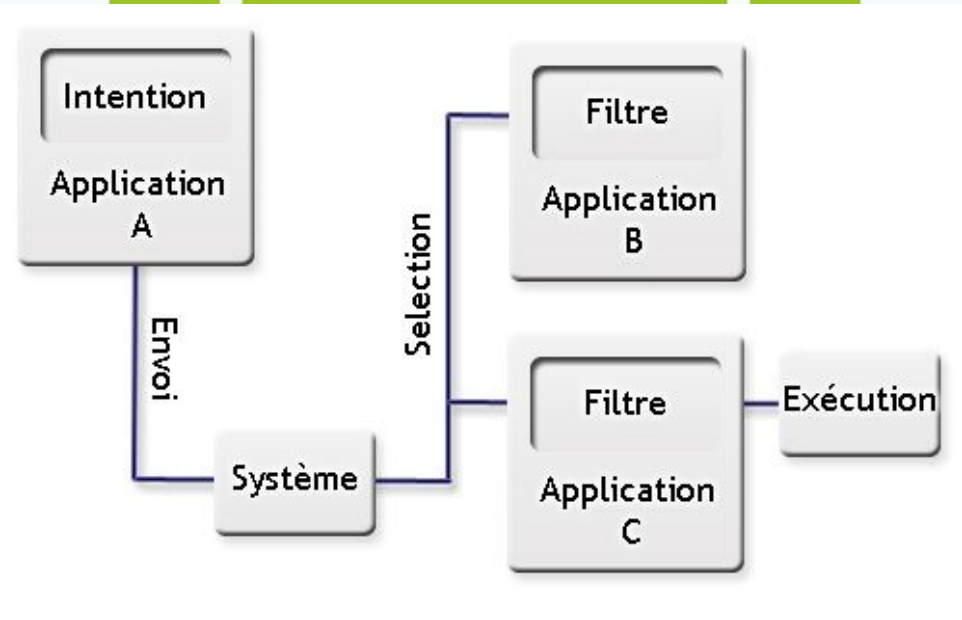
Par exemple, démarrer un navigateur internet

```
private void doWebBrowser(){
Uri uri = Uri.parse("http://www.google.fr/");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
}
```

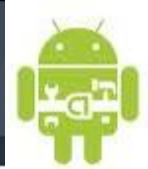
La classe Intent

Filtrer les actions

Android permet aux applications de spécifier quelles sont les actions qu'elles gèrent : le système peut ainsi choisir le composant le mieux adapté au traitement d'une action (véhiculée dans un objet Intent).



La classe Intent



Filtrer les actions

```
<activity ...
<intent-filter>
<action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <category android:name="android.intent.category.BROWSABLE”/>
      <data android:scheme="demo" />
</intent-filter>
</activity>
```

• ***action : identifiant unique sous forme de chaîne de caractères.*** Il est d'usage d'utiliser la convention de nom Java ;

• ***category : premier niveau de filtrage de l'action.*** Cette balise indique dans quelle circonstance l'action va s'effectuer ou non. Il est possible d'ajouter plusieurs balises de catégorie ;

• ***data : filtre l'objet Intent au niveau des données elles-mêmes.*** Par exemple en jouant avec l'attribut android:host on peut répondre à une action comportant un nom de domaine particulier, comme *www.monsite.com.*

La classe Intent

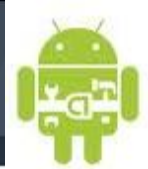
3 Envoyer des informations entre applications

```
Intent intent = new Intent(this,MonActivite.class);
intent.putExtra("maclé", "hello !");
startActivity(intent);
```

Récuperer une donnée d'un Bundle :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
Bundle extra = this.getIntent().getExtras();
if(extra != null)
String data = extra.getString("maclé"); }
```

La classe Intent

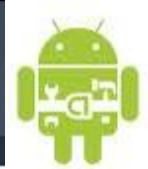


Transférer un Intent

Si votre application réceptionne un Intent comportant une action que vous ne traitez pas ou que vous ne souhaitez pas traiter

```
Intent intent = getIntent();
Uri data = intent.getData();
if(...)
startNextMatchingActivity(getIntent());
```

La classe Intent



Diffuser des Intents à but informatif

- La première étape pour comprendre cette mécanique de diffusion consiste à envoyer un Intent au système grâce à la méthode sendBroadcast d'une activité

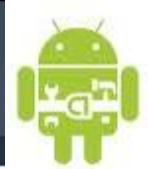
```
private void doBroadcast(){
Intent intent = new Intent(MyBroadcastReceiver.VIEW);
intent.putExtra( "extra", "hello !");.
```

Recevoir et traiter des Intents diffusés

```
<manifest ...>
<application ...>
...
<receiver android:name="MyBroadcastReceiver">
<intent-filter>
<action
android:name="cnrs.intent.action.VIEW" />
<category
android:name="android.intent.category.DEFAULT" />
```

```
public final class MyBroadcastReceiver extends
BroadcastReceiver {
public static final String VIEW =
"cnrs.intent.action.VIEW";
@Override
public void onReceive(Context context, Intent
intent) {
... // Insérer le code de traitement de l'Intent ici.
}
}
```

La classe Intent



Créer un récepteur d'Intents dynamiquement

- Pour créer un récepteur d'Intents pendant l'exécution d'une application, créez tout d'abord une classe dérivant de BroadcastReceiver

```
MyBroadcastReceiver receiver;
private void doAddReceiver(){
receiver = new MyBroadcastReceiver();
IntentFilter filter =
new IntentFilter(MyBroadcastReceiver.CALL);
registerReceiver(receiver, filter);
}
```

Libération des ressources prises par un récepteur d'Intents

```
MyBroadcastReceiver receiver;
...
unregisterReceiver(receiver);
```

Activités et interactions

Développement avancé : Lancement d'une sous acti

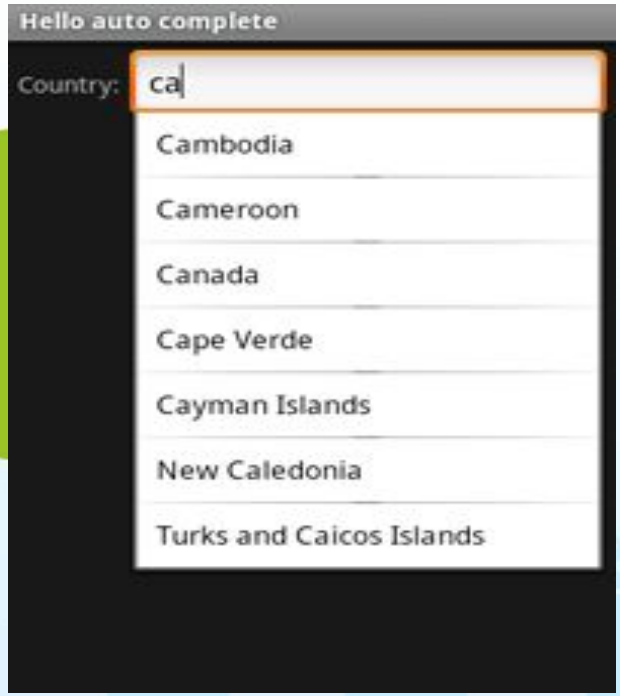
Chaque programme (visible) est programmé en activity

Activite1

Lancement

Retour

Activite2



Lancement sous activity

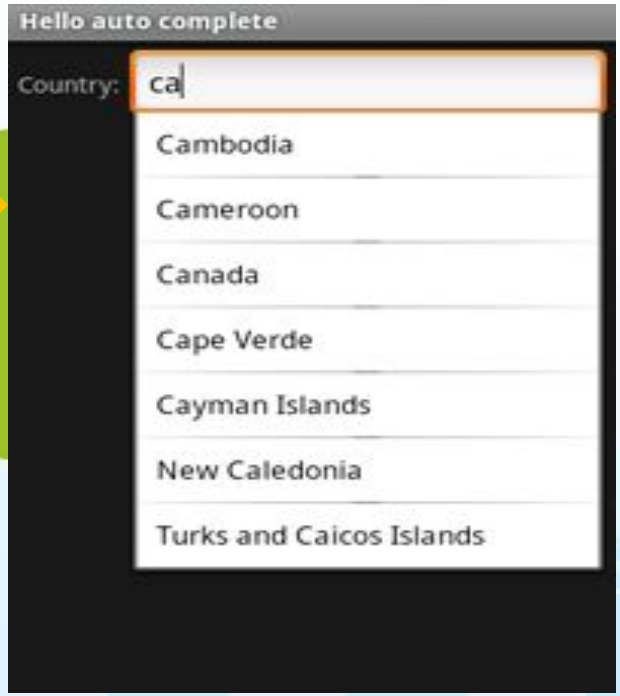
Activite1

```
public void onClick(View v)
 {
if (v == mButton)
  {
Intent intent = new Intent(Activity1.this, Activity2.class);

// démarage de l'activite 2
startActivity(intent);
  }
}
```

Lancement

Retour / menu back

Activite2



Lancement sous activity (manifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="example.android"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:label="@string/app_name" android:name="Activity1">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    <activity android:name="Activity2"></activity>
</application>
    <uses-sdk android:minSdkVersion="8" />
</manifest>
```

Activity principale

Activity autre

Développement avancé : Passage d'information entre activité

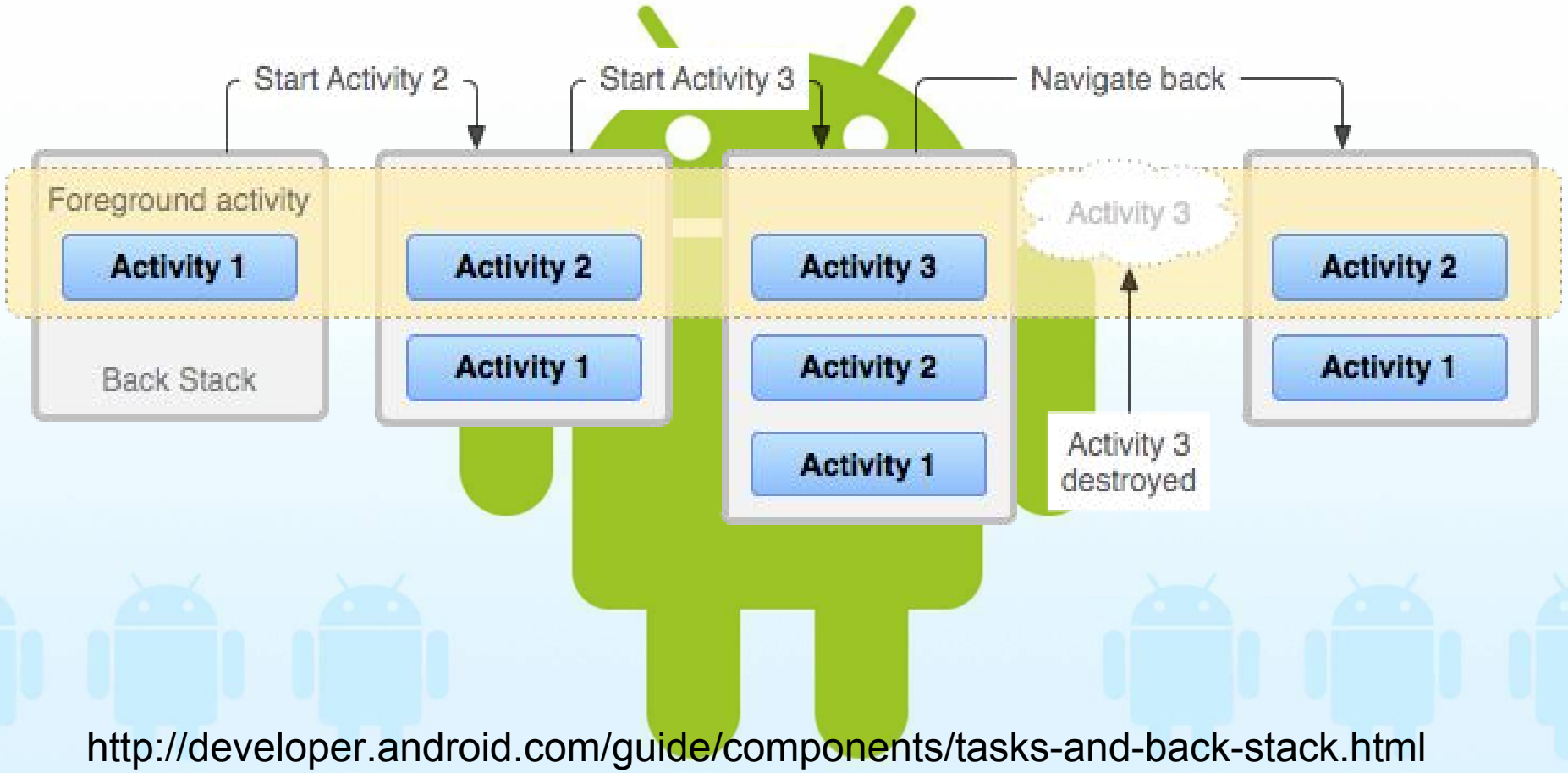
Lors du lancement d'une activité il est parfois nécessaire de transmettre des informations, ces informations sont transmises lors du lancement :

activity1

```
Intent intent = new Intent(Activity1.this,
Activity2.class);

// envoie d'une info a l'activite 2
intent.putExtra("example.android.Activity1",
"info");

// démarage de l'activite 2
startActivity(intent);
```

activity2

```
Bundle extras = getIntent().getExtras();
String info =
extras.getString("example.android.Activity1");
```



http://developer.android.com/guide/components/tasks-and-back-stack.html

Développement avancé : Activity obtenir un retour

La méthode startActivity() permet de démarrer une autre activité. Cette méthode est utile mais ne propose pas un mécanisme de retour d'information vers l'activité "parent". Pour gérer le retour d'information vers l'activité parent (données de formulaire, état ...), l'API offre une possibilité avec startActivityForResult(). Avec cette méthode, lorsque l'activité aura terminée sa tâche, elle avertira l'activité parent.

```
public class MyActivity extends Activity {
     ...

static final int PICK_CONTACT_REQUEST = 0;

protected boolean onKeyDown(int keyCode, KeyEvent event)
{

if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
// When the user center presses, let them pick a
contact.
             startActivityForResult(
                 new Intent(Intent.ACTION_PICK,
                 new Uri("content://contacts")),
                 PICK_CONTACT_REQUEST);
            return true;
        }
        return false;
    }
```

```
protected void onActivityResult(int requestCode, int
resultCode, Intent data)
 {
         if (requestCode == PICK_CONTACT_REQUEST) {
             if (resultCode == RESULT_OK) {
                 // A contact was picked.  Here we will
just display it
                 // to the user.
                 startActivity(new
Intent(Intent.ACTION_VIEW, data));
             }
         }
     }
 }
```

http://developer.android.com/reference/android/app/Activity.html#StartingActivities

La classe Intent

TP

Lancer une activité avec passage de paramètres : ANDROID TD2 → Activity1

Lancer une activité avec un retour : ANDROID TD2 → startScreen

Lancer la géolocalisation : ANDROID TD2 → LaunchDemo

Lancer une activité du système : SimpleSMS → SMS + MAP