
# ANDROID

## Architecture

1

## Gradle

http://gradle.org/

**Outil préconisé par Google pour construire les applications Android**

Une des nouveautés annoncée à la Google I/O 2013 concernait la mise à disposition d'un nouveau système de build.

Il était nécessaire de disposer d'un système commun facilitant la gestion de toutes les étapes de construction d'un projet.

Plusieurs initiatives existaient à base de Maven, Ant... mais au final Google a décidé de s'appuyer sur Gradle.

2

## Le choix de Gradle

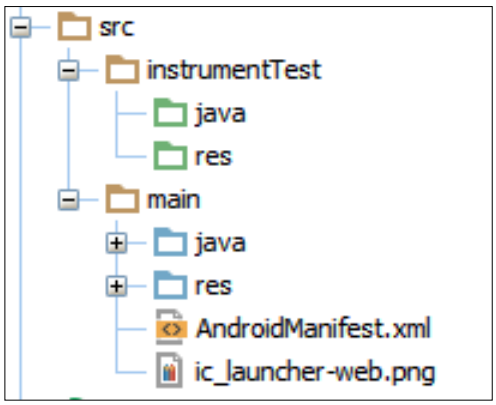
Gradle a été choisi pour plusieurs raisons

- il est plus souple dans son utilisation que Maven, qui impose une convention de développement plutôt adaptée au développement d'applications Java.
- il propose une gestion automatique des dépendances
- Un autre intérêt est d'avoir des fichiers de configuration en Groovy (voir : http://fr.wikipedia.org/wiki/Groovy_%28langage%29).
- son intégration commence à être correcte dans les IDE et notamment dans IntelliJ et AndroidStudio

3

## Mais quelle est donc la convention liée à un projet Android ?

Structuration d'un projet
Des répertoires pour le projet mais aussi pour les tests.




4

## Cycle de vie du projet et tâches

Toute application passe par différentes étapes avant de pouvoir être utilisée par les utilisateurs.

On doit récupérer les sources, vérifier, tester, créer un package pour debugger, retester, déployer, signer, releaser... Toutes ces différentes étapes constituent le cycle de vie de votre application.

Ce terme vient de Maven mais il s'applique aussi à des constructions gérées via Ant ou Gradle

Le principe de Gradle (ou de Ant) est de déclarer des tâches qui dépendent les unes des autres. Lorsque vous lancez une tâche toutes les sous tâches sont d'abord lancées

5

Exemple d'un build:

```
:compileJava
:processResources
:classes
:jar
:assemble
:compileTestJava
:processTestResources
:testClasses
:test
:check
:build

BUILD SUCCESSFUL
```

6

# INTRODUCTION

## En Java

Votre code est une suite d'instructions que l'on trouve dans un fichier « **.java** » et ce fichier sera traduit en une autre suite d'instructions dans un autre langage que l'on appelle le « **bytecode** ».

Ce code est contenu dans un fichier « **.class** ». Le bytecode est un langage spécial qu'une machine virtuelle Java peut comprendre et interpréter.

Les différents fichiers « **.class** » sont ensuite regroupés dans un « **.jar** » et c'est ce fichier qui est exécutable.

7

## Pour Android

La version de Java qui permet le développement Android est une version réduite amputée de certaines fonctionnalités qui n'ont rien à faire dans un environnement mobile.

Par exemple, la bibliothèque graphique Swing n'est pas supportée, on trouve à la place un système beaucoup plus adapté.

Mais Android n'utilise pas une machine virtuelle Java ; une machine virtuelle toute étudiée pour les systèmes embarqués a été développée, et elle s'appelle « **Dalvik** ».

Cette machine virtuelle est optimisée pour mieux gérer les ressources physiques du système.

8

Les applications Java développées pour Android doivent être compilées au format dalvik exécutable (.dex) avec l'outil dx.

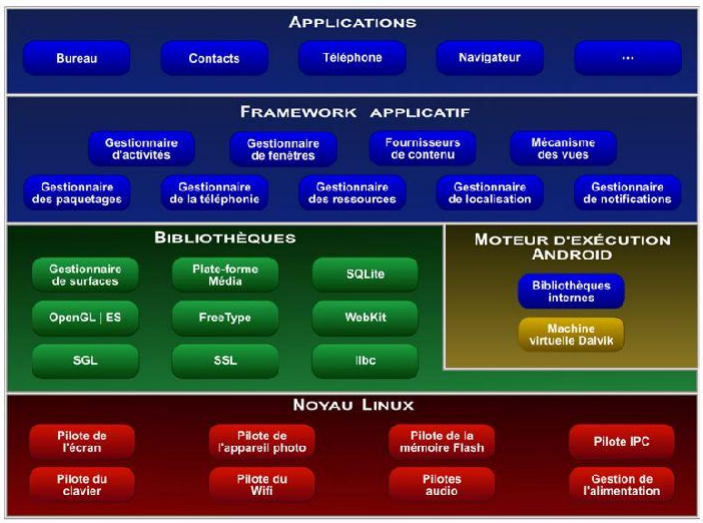
Cet outil compile les .java en .class et ensuite il convertit ces .class en .dex. Un .dex peut contenir plusieurs classes.

Le bytecode utilisé dans les .dex est le Dalvik Bytecode et non le java Bytecode.

Exécutable .apk

9

## ARCHITECTURE ANDROID




10

Le système d'exploitation d'Android se base sur Linux. C'est le noyau (« **kernel** » en anglais) de Linux qui est utilisé.

Le noyau est l'élément du système d'exploitation qui permet de faire le pont entre le matériel et le logiciel.

11

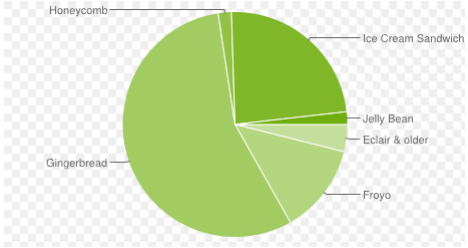
La version du noyau utilisée avec Android est une version conçue spécialement pour l'environnement mobile, avec une gestion avancée de la batterie et une gestion particulière de la mémoire.

Les bibliothèques proviennent de beaucoup de projets open-sources, écris en C/C++ pour la plupart, comme SQLite pour les bases de données, WebKit pour la navigation web ou encore OpenGL afin de produire des graphismes en 2D ou en 3D.

12

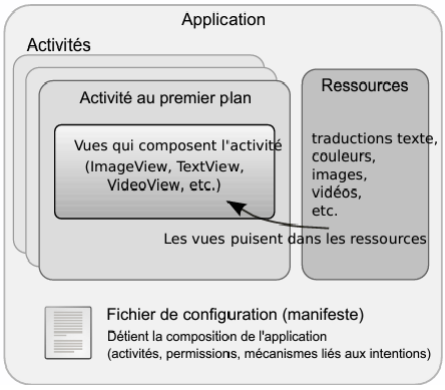
## Repartition des différentes versions au 1er Octobre 2012

| Versions | Date de sortie | API level |
|---|---|---|
| 4.1.x ; 4.2 *Jelly Bean* | 9 juillet 2012 | 16 |
| 4.0.x *Ice Cream Sandwich* | 19 octobre 2011 | 14-15 |
| 3.x.x *Honeycomb* | 22 février 2011 | 11-13 |
| 2.3.x *Gingerbread* | 6 décembre 2010 | 9-10 |
| 2.2.x *Froyo* | 20 mai 2010 | 8 |
| 2.0 ; 2.1 *Eclair* | 26 octobre 2009 | 5-6-7 |
| 1.6 *Donut* | 15 septembre 2009 | 4 |
| 1.5 *Cupcake* | 30 avril 2009 | 3 |

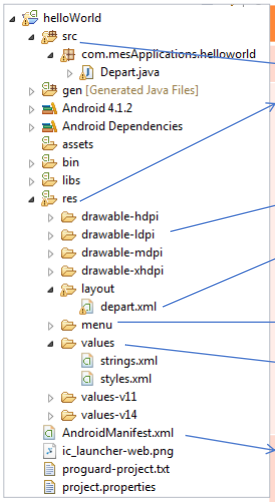



13

## Une application Android est une application objet




14




## Contenu

L'ensemble des sources du projet

res : répertoire contenant toutes les ressources telles que les images, les vues de l'interface graphique, etc. nécessaires à l'application. Ce répertoire est structuré par défaut de la manière suivante :

- res/drawable : contient les ressources de type image sur différentes dimensions;
- res/layout : contient les descriptions des interfaces graphiques au format XML (les vues) ;
- res/xml : contient les fichiers XML supplémentaires (non présents par défaut) ;
- res/menu : contient la description des menus, composants très courants d'une vue ;
- res/values : contient diverses ressources, tels que les textes, qui sont empaquetées sans aucun traitement.

androidManifest.xml : fichier XML décrivant l'application et ses composants, tels que les activités, les services etc. Le manifeste est en quelque sorte la carte d'identité de l'application, et permet d'autoriser l'exécution des activités et autres actions de l'application.

15

## EXEMPLE SIMPLE DU FICHIER ANDROIDMANIFEST.XML

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.mesApplications.helloworld"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="15" />

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

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

</manifest>
```

### Explications

La version de l'application est précisée par les attributs **versionCode** (numéro de la version) et **versionName** (nom de la version : visible dans le Market) de la balise manifeste.

La version minimum du SDK pour pouvoir utiliser l'application est renseignée dans l'attribut **minSdkVersion** de la balise **uses-sdk** ainsi que la version du SDK utilisée pour développer l'application (balise **targetSDKVersion**)

L'icône et le nom de votre application sont précisées par les attributs **icon** et **label** de la balise application ainsi que le thème utilisé par l'application . Répertoire res.

La description de l'activité principale (balise **activity**) :
Le nom de la classe qui implémente Activity.
Le titre pour l'activité.
Des filtres sur l'activité :
- **MAIN** : indique qu'il s'agit de l'activité principale de l'application.
- **LAUNCHER** : indique que cette activité est présente dans le lanceur d'application.

## FICHIER STRINGS.XML

```
<resources>

    <string name="app_name">Application_helloWorld</string>
    <string name="hello_world">Que le monde est beau !</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_depart">Mon hello world</string>

</resources>
```

Ce fichier contient quatre chaînes de caractères :

- Le nom de l'application.
- Le message « Que le monde est beau !» à afficher.
- La chaîne de caractères utilisée dans la barre d'action/menu.
- Le titre de l'activité principale.

17

## DOSSIER VALUES

Vous pouvez remarquer les quatre dossiers values :

- **values** : dossier utilisé par défaut pour stocker les différentes valeurs utiles dans une application (utilisé dans la majorité des cas).

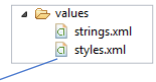
- **values-large** : tous les fichiers contenus dans ce dossier remplacent les fichiers values par défaut lorsque l'appareil cible possède un grand écran (tablette par exemple).

- **values-v11** : tous les fichiers contenus dans ce dossier remplacent les fichiers values par défaut lorsque l'appareil possède la version 3.x d'Android (Honeycomb).

- **values-v14** : tous les fichiers contenus dans ce dossier remplacent les fichiers values par défaut lorsque l'appareil possède la version 4.x d'Android (Ice Cream Sandwich ou JellyBean).

18

## FICHIER STYLES.XML

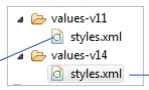



```
<resources>

    <style name="AppTheme" parent="android:Theme.Light" />

</ resources>
```

19




```
<resources>

    <style name="AppTheme" parent="android:Theme.Holo.Light" />

</resources>
```

```
<resources>

    <style name="AppTheme" parent="android:Theme.Holo.Light.DarkActionBar"/>

</resources>
```

20

## Internationalisation

Le système de ressources permet de gérer très facilement l'internationalisation d'une application. Il suffit de créer des répertoires **values-XX** où **XX** est le code de la langue que l'on souhaite implanter.

On place alors dans ce sous-répertoire le fichier xml **strings.xml** contenant les chaines traduites associées aux mêmes clefs que dans **values/strings.xml**.

21

On obtient par exemple pour les langues es et fr l'arborescence :

```
MyProject/
  res/
    values/
      strings.xml
    values-es/
      strings.xml
    values-fr/
      strings.xml
```

22

Les applications Android sont composées de 4 types de composants :

- *Activities: Une Activity représente un écran de l'application.*

*Une application peut avoir une ou plusieurs activities (par exemple pour une application de messagerie on pourrait avoir une Activity pour la liste des contacts et une autre pour l'éditeur de texte).*

*Chaque Activity est implémentée sous la forme d'une classe qui hérite de la classe Activity.*

23

- *Services: Les services n'ont pas d'interface graphique et tournent en tâche de fond. Il est possible de s'inscrire à un service et de communiquer avec celui-ci en utilisant l'API Android.*

- *Broadcast receivers: Il se contente d'écouter et de réagir aux annonces broadcast (par exemple changement de fuseau horaire, appel entrant...)*

- *Content providers : Il permet de partager une partie des données d'une application avec d'autres applications.*

24

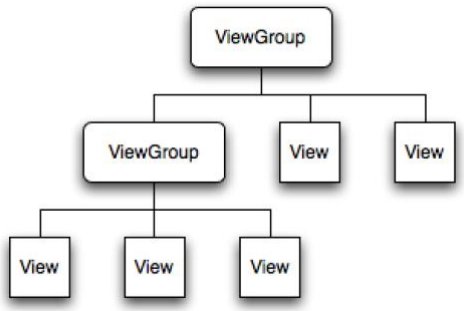
## Interfaces graphiques:

Les Interfaces graphiques dans Android sont construites en utilisant les classes View et Viewgroup.

La classe View sert de base à un ensemble de sous-classes appelées Widget qui permettent l'interface avec l'utilisateur (boutons, zone de texte...).

25

La classe ViewGroup sert de base à un ensemble de sous-classes appelées layout qui servent à organiser les View dans l'espace. Ainsi, une interface graphique peut être représentée sous forme d'un arbre de Views:




26

Pour faire apparaître cette interface graphique, la classe principale (Activity) doit appeler la méthode setContentView() en passant en référence le noeud racine.

Il y a 2 méthodes pour implémenter une interface graphique dans l'API Android:

 L'arbre peut être décrit selon un langage XML, qui est converti à la compilation du projet en une ressource "View" instanciable depuis le programme principal. L'interface peut alors être chargée avec la méthode setContentView().

27

## accueil.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/and
roid" android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:gravity="center"
android:id="@+id/accueilid"
</LinearLayout>
```

```
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.accueil); }
```

28

On peut accéder à cet élément par son id et agir dessus au travers du code Java :

```
LinearLayout l = (LinearLayout)findViewById(R.id.accueilid);

l.setBackgroundColor(Color.BLACK);
```

29

- On peut instancier directement les éléments dans le code de l'application et en manipulant leurs méthodes pour aboutir au résultat voulu.

```
LinearLayout gabarit = new LinearLayout(this);

// Centrer les éléments graphiques
 gabarit.setGravity(Gravity.CENTER);
```

30

L'avantage de décrire l'interface graphique en XML est que cela permet de mieux séparer la partie présentation de la partie exécution.

Cependant, la description XML doit être faite avant la compilation du projet.

31

Le fichier R.java est un fichier généré par le SDK Android. Ce fichier se génère automatiquement une fois que tout le code de votre projet peut être compilé (pas d'erreur sur votre projet).
Ce qui veut dire que si ce fichier n'est pas présent, c'est qu'il y a soit :

des erreurs dans votre projet :

XML : problèmes d'accents, de majuscules, de ressources mal créées, de balises mal écrites, etc.

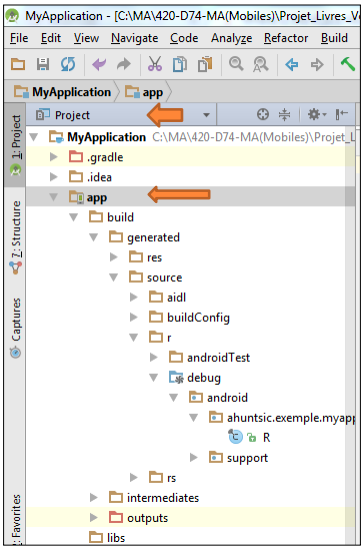
Java : Une erreur java empêche la compilation de votre projet.

Manifest : Des éléments / attributs présents dans le manifest sont incorrects.

- qu'il ne peut pas le générer automatiquement, car l'action "Build automatique" n'est pas sélectionnée sur le projet.

32

## ANDROID STUDIO IL EST :




33

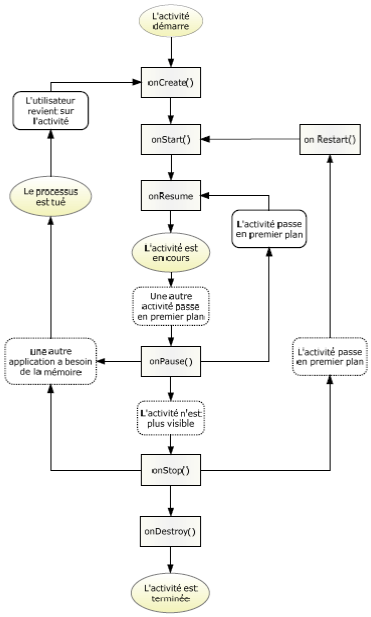
Contrairement aux applications classiques, les applications Android n'ont pas un contrôle complet sur leur cycle de vie.

- Les composants doivent être à l'écoute des changements d'états de l'application pour être près à une fin intempestive

- La mémoire et la gestion des processus est prise en charge exclusivement par le moteur d'exécution

34

## CYCLE DE VIE D'UNE ACTIVITÉ




35

Une application Android étant hébergée sur un système embarqué, le cycle de vie d'une application ressemble à celle d'une application Java ME. L'activité peut passer des états:

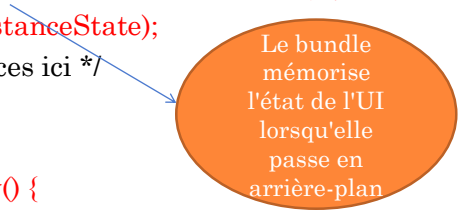
- démarrage -> actif: détient le focus et est démarré (**onStart** invoqué)
- actif -> suspendue: ne détient plus le focus (**onPause** invoqué)
- suspendue -> actif: **onResume** invoqué
- suspendue -> détruit: **onDestroy** invoqué

36

```
public class Main extends Activity {
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.acceuil); }
    protected void onDestroy() {
    super.onDestroy(); }
    protected void onPause() {
    super.onPause(); }
    protected void onResume() {
    super.onResume(); }
    protected void onStart() {
    super.onStart(); }
    protected void onStop() {
    super.onStop(); }
    }
```

37

```
public class MonActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    /* Allocation des ressources ici */
    }
    @Override
    protected void onDestroy() {
    super.onDestroy();
    /* Désallocation des ressources ici */
    }
}
```




L'objet **Bundle** passé en paramètre de la méthode **onCreate** permet de restaurer les valeurs des interfaces d'une activité qui a été déchargée de la mémoire.

38