II. Présentation Android

Ce cours vous présente l'installation et l'interface d'Android studio. Nous verrons également les bases du développement d'une application Android.

historique

Il est important de prendre la mesure des choses.

A l'heure actuelle (Novembre 2017) :

- juillet 2011 : 550 000 activations par jour
- décembre 2011 : 700 000 activations par jour
- sept. 2012 : 1.3 millions d'activations par jour (Wikipedia)
- avril 2013 : 1.5 millions d'activations par jour (Wikipedia)
- mai 2015 : 3,5 millions d'activation par jour (Wikipedia)

Nom	Version	Date
Android	1.0	09/2008
Android 1.1	1.1	02/2009
Cupcake	1.5	04/2009
Donut	1.6	09/2009
Gingerbread	2.3	12/2010
Honeycomb	3.0	02/2011
Ice Cream Sandwich	4.0.1	10/2011
Jelly Bean	4.1	07/2012
KitKat	4.4	10/2013
Lolipop	5.0	10/2014
Marshmallow	6.0	05/2015
Nougat	7.0	09/2016
Oreo	8.0	08/2017
Pie	9.0	08/2018
-	10.0	09/2019

détail et correspondance Android

Android

L'écosystème d'Android s'appuie sur deux piliers:

- le langage (Java ou Kotlin)
- le SDK qui permet d'avoir un environnement de développement facilitant la tâche du développeur

Le kit de développement donne accès à des exemples, de la documentation mais surtout à l'API de programmation du système et à un émulateur pour tester ses applications.

Stratégiquement, Google utilise la licence Apache pour Android ce qui permet la redistribution du code sous forme libre ou non et d'en faire un usage commercial.

Le SDK était:

- anciennement manipulé par un plugin d'Eclipse (obsolète)
- maintenant intégré à Android Studio (IntelliJ)

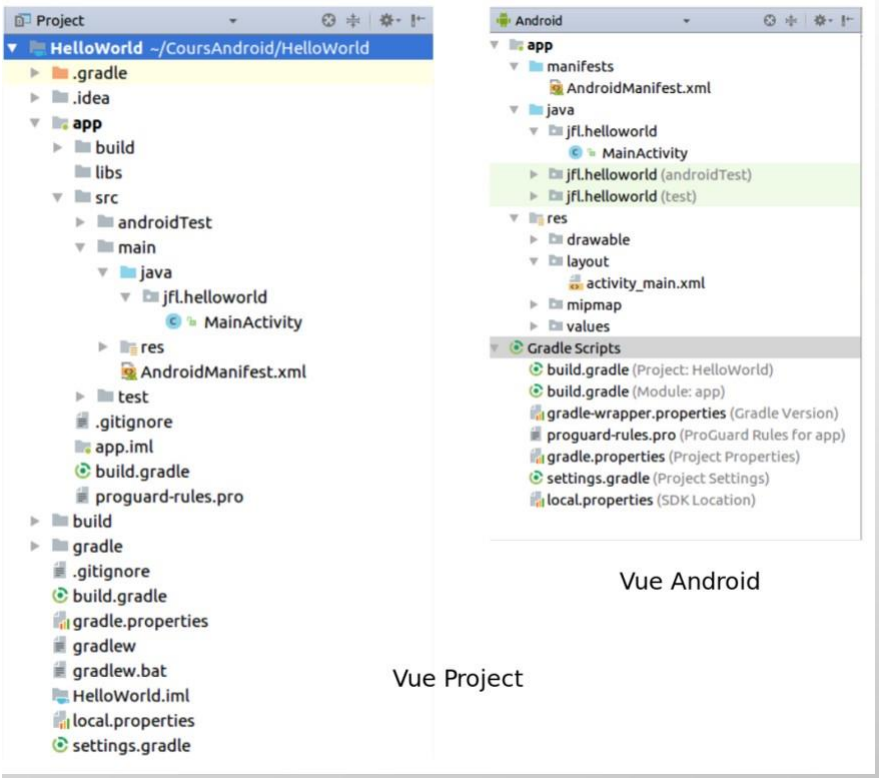
Android est en fait un système de la famille des Linux, pour une fois sans les outils GNU.

L'OS s'appuie sur:

- un noyau Linux (et ses drivers)
- un couche d'abstraction pour l'accès aux capteurs (HAL)
- une machine virtuelle: Dalvik Virtual Machine (avant Lollipop)
- un compilateur de bytecode vers le natif Android Runtime (pour Lollipop)
- des applications (navigateur, gestion des contacts, application de téléphonie…)
- des bibliothèques (SSL, SQLite, OpenGL ES, etc…)
- des API d'accès aux services Google

L'architecture d'un projet Android Studio

- app: le code de votre application
- build: le code compilé
- lib: les librairies natives
- src: les sources de votre applications
- main/java: vos classes
- main/res: vos ressources (XML, images, …)
- test: les tests unitaires



Des fichiers de configuration:

- build.gradle (2 instances): règles de dépendance et de compilation
- settings.gradle: liste de toutes les applications à compiler (si plusieurs)

Les éléments d'une application

Une application Android peut être composée des éléments suivants:

- des activités (android.app.Activity): il s'agit d'une partie de l'application présentant une vue à l'utilisateur
- des services (android.app.Service): il s'agit d'une activité tâche de fond sans vue associée
- des fournisseurs de contenus (android.content.ContentProvider): permet le partage d'informations au sein ou entre applications
- des widgets (android.appwidget.): une vue accrochée au Bureau d'Android
- des Intents (android.content.Intent): permet d'envoyer un message pour un composant externe sans le nommer explicitement
- des récepteurs d'Intents (android.content.BroadcastReceiver): permet de déclarer être capable de répondre à des Intents
- des notifications (android.app.Notifications): permet de notifier l'utilisateur de la survenue d'événements

Le Manifest de l'application

Le fichier AndroidManifest.xml déclare l'ensemble des éléments de l'application

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="andro.jf"
      android:versionCode="1"
      android:versionName="1.0">
  <application android:icon="@drawable/icon"
               android:label="@string/app_name">

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

    <service>...</service>
    <receiver>...</receiver>
    <provider>...</provider>

    </application>
</manifest>
```

Les ressources

Les ressources de l'applications sont utilisées dans le code au travers de la classe statique R.

ADT re-génère automatiquement la classe statique R à chaque changement dans le projet. Toutes les ressources sont accessibles au travers de R, dés qu'elles sont déclarées dans le fichier XML ou que le fichier associé est déposé dans le répertoire adéquat.

Les ressources sont utilisées de la manière suivante:

```
android.R.type_ressource.nom_ressource
```

qui est de type int. Il s'agit en fait de l'identifiant de la ressource.

On peut alors utiliser cet identifiant ou récupérer l'instance de la ressource en utilisant la classe Resources:

```
Resources res = getResources();
String hw = res.getString(R.string.hello);
XXX o = res.getXXX(id);
```

Une méthode spécifique pour les objets graphiques permet de les récupérer à partir de leur id, ce qui permet d'agir sur ces instances même si elles ont été créées via leur définition XML:

```
TextView texte = (TextView)findViewById(R.id.le_texte);
texte.setText("Here we go !");
```

Autres

Les chaines constantes de l'application sont situées dans res/values/strings.xml. L'externalisation des chaines permettra de réaliser l'internationalisation de l'application.

Voici un exemple :

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello Hello JFL !</string>
    <string name="app_name">AndroJF</string>
</resources>
```

La récupération de la chaine se fait via le code:

```
Resources res = getResources();
String hw = res.getString(R.string.hello);
```

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 même clefs que dans values/strings.xml.

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
```

D'autres ressources sont spécifiables dans res :

- les menus (R.menu)
- les images (R.drawable)
- des couleurs (R.color)

Les activités

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
- actif -> suspendue : ne détient plus le focus
- suspendue -> actif
- suspendue -> détruit

Le nombre de méthodes à surcharger et même plus important que ces états :

```
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(); } }
```

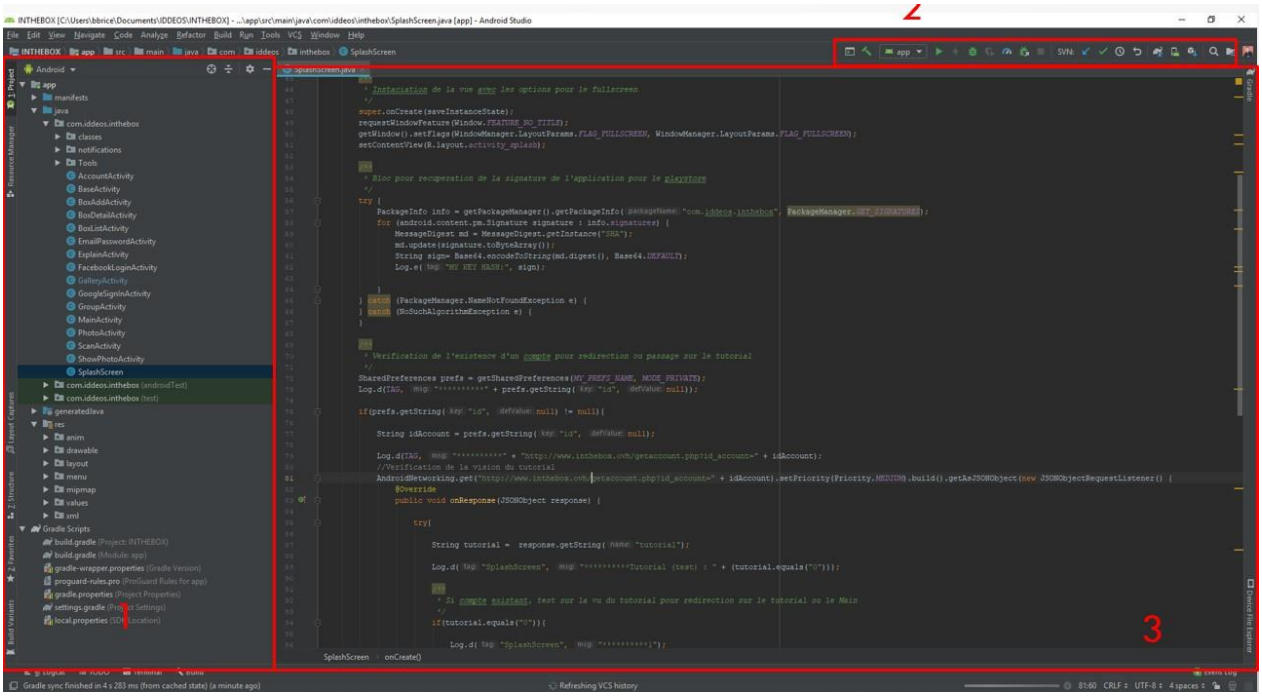
Cycle de vie d'une activité

- `onCreate()` / `onDestroy()` : permet de gérer les opérations à faire avant l'affichage de l'activité, et lorsqu'on détruit complètement l'activité de la mémoire. On met en général peu de code dans `onCreate()` afin d'afficher l'activité le plus rapidement possible.
- `onStart()` / `onStop()` : ces méthodes sont appelées quand l'activité devient visible/invisible pour l'utilisateur.
- `onPause()` / `onResume()` : une activité peut rester visible mais être mise en pause par le fait qu'une autre activité est en train de démarrer, par exemple B.
- `onPause()` ne doit pas être trop long, car B ne sera pas créé tant que `onPause()` n'a pas fini son exécution.
- `onRestart()`: cette méthode supplémentaire est appelée quand on relance une activité qui est passée par `onStrop()`. Puis `onStart()` est aussi appelée. Cela permet de différencier le premier lancement d'un relancement.

Le cycle de vie des applications est très bien décrit sur la page qui concerne les Activity.

Interface developpement

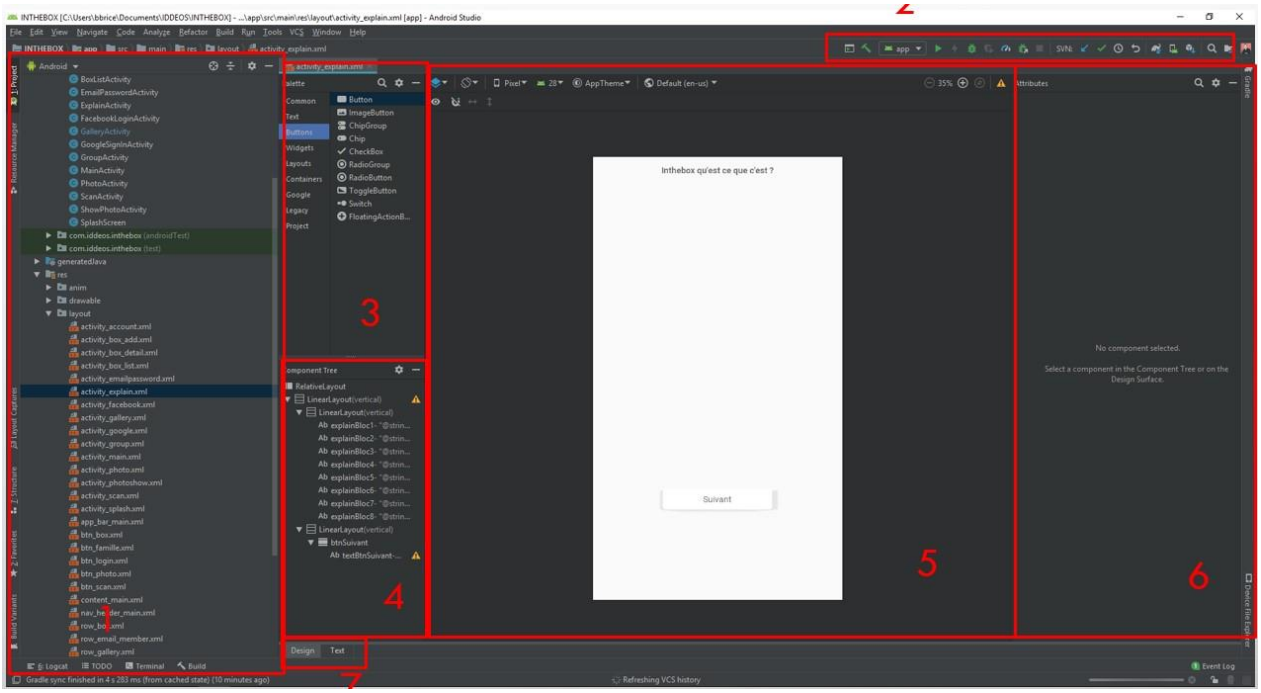
Interface disponible lors de l'édition d'un fichier .java par exemple :



1. Gestionnaire de fichier
2. outils (gestionnaire de machine virtuel, gestionnaire de SDK, build, …)
3. Éditeur de code

Interface composition graphique

Interface disponible lors de l'édition d'un fichier layout par exemple :



1. Gestionnaire de fichier
2. outils (gestionnaire de machine virtuel, gestionnaire de SDK, build, …
3. Bibliothèque de composants (s'utilise en glisser déposer vers le preview

4. Composition du layout
5. preview
6. gestion des paramètres d'un composant
7. passage en xml