

Développement des applications mobiles



Dr. Mohammed BELATAR
2018/2019

But du cours

Le but de ce cours est de découvrir la programmation Mobile en se basant sur l'exemple Android, sa plate-forme de développement et les spécificités du développement embarqué sur smartphone. Le cours vise aussi à se familiariser avec l'architecture distribuée des applications mobile basée sur la consommation de services Web.

Les principales notions abordées dans ce cours sont:

● Cycle de vie d'une activité.	● Les Intent.
● Les gabarits : LinearLayout, RelativeLayout, TableLayout et ScrollView.	● Persistence des données : Sqlite.
● Les Vues de base.	● Les services.
● ListView personnalisée et Galeries	● Accès aux services web.

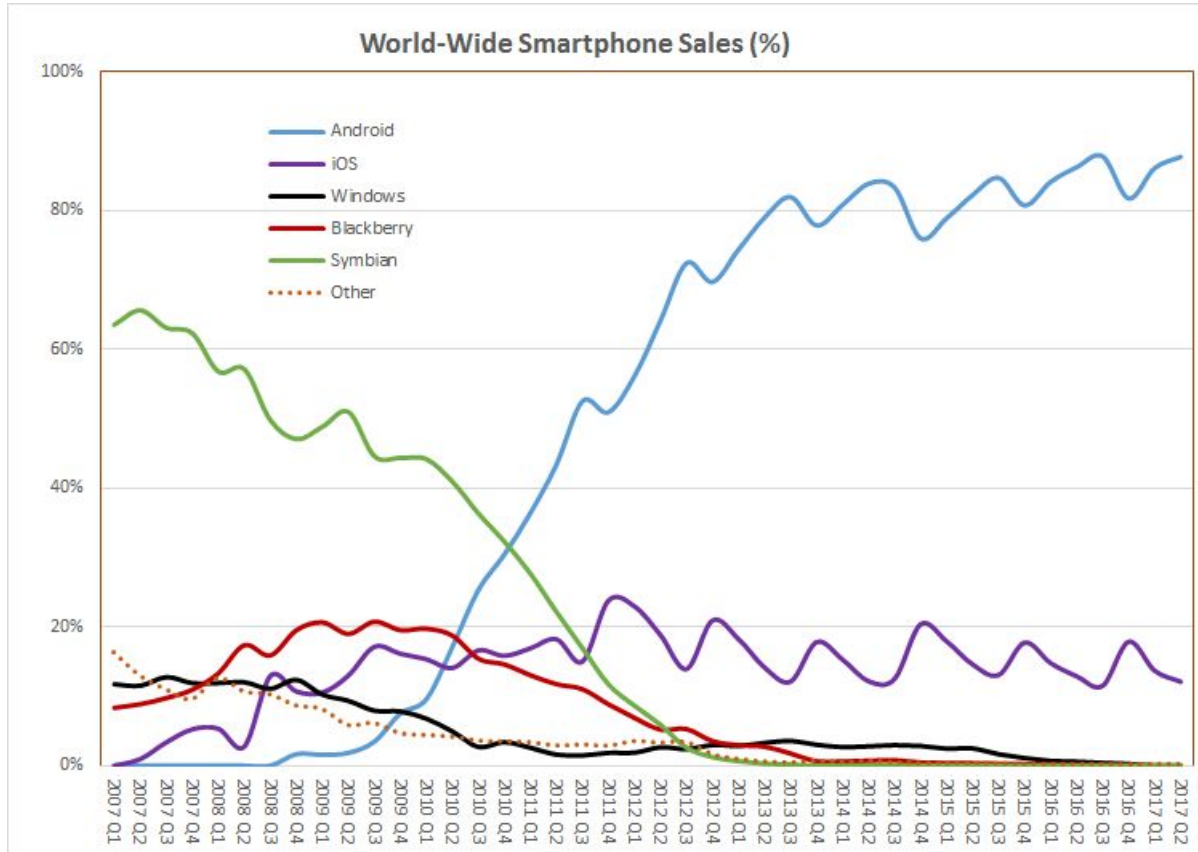


1. Plan du cours

- 1 Introduction
- 2 L'environnement Android
- 3 Interfaces graphiques
- 4 Les Intents
- 5 Persistance des données
- 6 Programmation concurrente
- 7 Connectivité
- 8 Les services Web
- 9 Divers
- 10 Annexes: outils
- 11 Annexes: codes sources
- 12 Bibliographie



1. Introduction : les OS Mobile



2. L'environnement t Android

- 2.1 Introduction
- 2.2 Android
- 2.3 Les ressources
- 2.4 Les activités

Evolution

Il est important de prendre la mesure des choses. A l'heure actuelle (May 2015):

- 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)
- Il y aurait actuellement un parc de plus de 1 milliard d'appareils Android.

Vous pouvez visionner de la propagande Google sur Youtube [ICI](#).



Historique des versions

Le nombre de release est impressionnant [Version]:

Android	1.0	09/2008
Petit Four	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	10/2011
Jelly Bean	4.1	07/2012
Kit Kat	4.4	10/2013
Lollipop	5.0	10/2014
Marshmallow	6.0	10/2015
Nougat	7.0	08/2016
Oreo	8.0	08/2017



2.2 Qu'est ce que Android?

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

- le langage Java (depuis son apparition)
- le SDK facilitant la tâche du développeur

très récemment

- le langage Kotlin (par JetBrains)

Le kit de développement donne accès à des exemples, de la documentation mais surtout à l'API de programmation du système et à des émulateurs 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 plugin Android Development Tool permet d'intégrer les fonctionnalités du SDK à Eclipse.

Android Studio offre un environnement de développement Android complet qui remplace l'ADT.





Eclipse+ADT vs Android Studio

USER GUIDE

Search

ADT Plugin Release Notes

The Eclipse ADT plugin is no longer supported per our [announcement](#). [Android Studio](#) is now the official IDE for Android, so you should migrate your projects to Android Studio as soon as possible. For more information on transitioning to Android Studio, see [Migrate to Android Studio from Eclipse](#).

Formerly the official IDE solution for Android development, Android Developer Tools (ADT) is a plugin for Eclipse that provides GUI-based access to many of the command-line Android SDK tools.

As with ADT, support for the [Ant](#) tool for building from the command line has ended. [Gradle](#) is now the supported method of building Android apps.



Le système d'exploitation

Android est un système de la famille Linux, pour une fois sans les outils GNU. L'OS s'appuie sur:

- un noyau Linux (et ses drivers)
- une couche d'abstraction pour l'accès aux capteurs (HAL)
- une machine virtuelle: Dalvik Virtual Machine
- des applications (navigateur, gestion des contacts, application de téléphonie...)
- des bibliothèques (SSL, SQLite, OpenGL ES, etc...)

[Dalvik] est le nom de la machine virtuelle open-source utilisée sur les systèmes Android. Cette machine virtuelle exécute des fichiers .dex, plus compactes que les .class classiques. Ce format évite par exemple la duplication des String constantes. La machine virtuelle utilise elle-même moins d'espace mémoire et l'adressage des constantes se fait par un pointeur de 32 bits.

[Dalvik] n'est pas compatible avec une JVM du type Java SE ou même Java ME. La librairie d'accès est donc redéfinie entièrement par Google.



Environnement de développement

Projet :

```
<uses-sdk android:minSdkVersion="integer"
```

- minSdkVersion (obligatoire)
- targetSdkVersion (non obligatoire)
- maxSdkVersion (déconseillé)

```
android:targetSdkVersion="integer"
```

```
android:maxSdkVersion="integer" />
```

Android Studio organise le projet suivant les différentes parties les plus importantes d'une application Android :

- les “manifests”,
- les fichiers “java”,
- les fichiers “ressources”,
- les scripts “gradle”



Projet : Les Fichiers Manifest

Ils contiennent les informations importantes et essentielles de votre application pour que le système d'exploitation Android puisse l'exécuter sans problème :

- le “package name” de votre application,
- les différents composants de l'application (exemples des activités ou interfaces) toutes référencées à partir leur classes java.
- différentes autres informations comme des permissions spécifiques dont aura besoin l'application. Par exemple, pour pouvoir utiliser Internet ou la caméra, votre application doit en demander les droits.
- d'autres éléments bien sur que vous trouverez dans [la documentation](#), si vous voulez creuser davantage.

Projet : Fichiers Java

C'est du Java, donc l'application repose sur des fichiers Java.

Toute logique ou fonctionnalité que vous aurez à implémenter se fera à l'intérieur d'une classe Java.

Ces classes peuvent bien entendu être regroupées en package (ou dossiers) dans une arborescence avec une racine commune.



Projet: Les fichiers ressources

Ce répertoire rassemble toutes les parties statiques de votre application :

- Le dossier drawable : qui contient soit des fichiers images ou des fichiers xml permettant de dessiner une forme quelconque.
- Le dossier layout : il contient des fichiers xml qui définissent l’affichage et la disposition des différents éléments d’une vue.
- Le dossier menu : Pour le menu contextuel de l'application.
- Le dossier mipmap : réservé spécifiquement aux différentes icônes utilisées dans l'application.
- Le dossier values : il contient des éléments statiques de l'application, réutilisables ailleurs.

Les éléments d'une application

Une application Android peut contenir (entre autres...) les 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 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.

Exemple:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.belatar.monprojet"
    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>
```



2.3 Les ressources

Les ressources de l'application sont utilisées dans le code au travers de la classe statique **R**. Le SDK génère automatiquement la classe statique **R** à chaque changement dans le projet. Toutes les ressources sont accessibles au travers de **R**, dès qu'elles sont déposées 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);
```



Les ressources

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.nom_de_la_ressource);  
texte.setText("Here we go !");
```



Les chaînes

Les chaînes constantes de l'application sont situées dans **res/values/strings.xml**. L'externalisation des chaînes permettra de réaliser l'internationalisation de l'application. Voici un exemple:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Bonjour TLM !</string>
  <string name="app_name">MonAppli</string>
</resources>
```

La récupération de la chaîne 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 chaînes traduites associées aux mêmes clés que dans **values/strings.xml**. On obtient par exemple pour les langues *ar* et *fr* l'arborescence:

```
MonProjet/  
  res/  
    values/  
      strings.xml  
    values-ar/  
      strings.xml  
    values-fr/  
      strings.xml
```

Android chargera le fichier de ressources approprié en fonction de la langue du système.

Autres valeurs simples

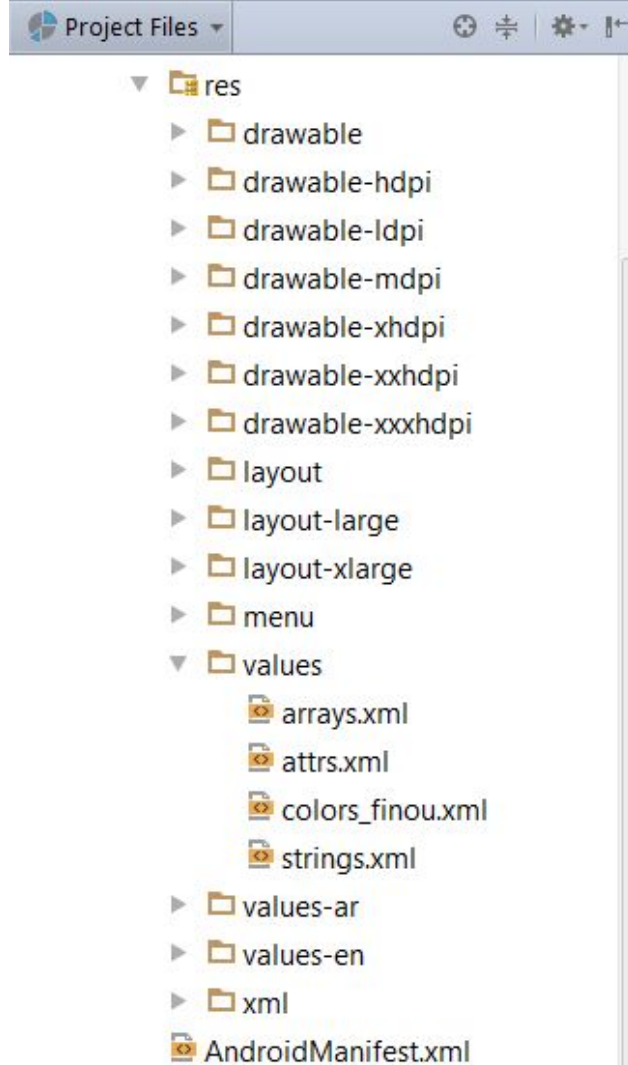
Plusieurs fichiers xml peuvent être placés dans **res/values**. Cela permet de définir des chaînes, des couleurs, des tableaux... On peut créer de nouveaux fichiers de ressources contenant des valeurs simples, comme par exemple un tableau de chaînes:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="monTableau">
    <item>item1</item>
    <item>item2</item>
  </string-array>
</resources>
```

Autres ressources

D'autres ressources peuvent être définies dans **res**:

- les menus
- les images (**R.drawable**)
- des dimensions (**R.dimen**)
- des couleurs (**R.color**)



2.4 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 dans plusieurs 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

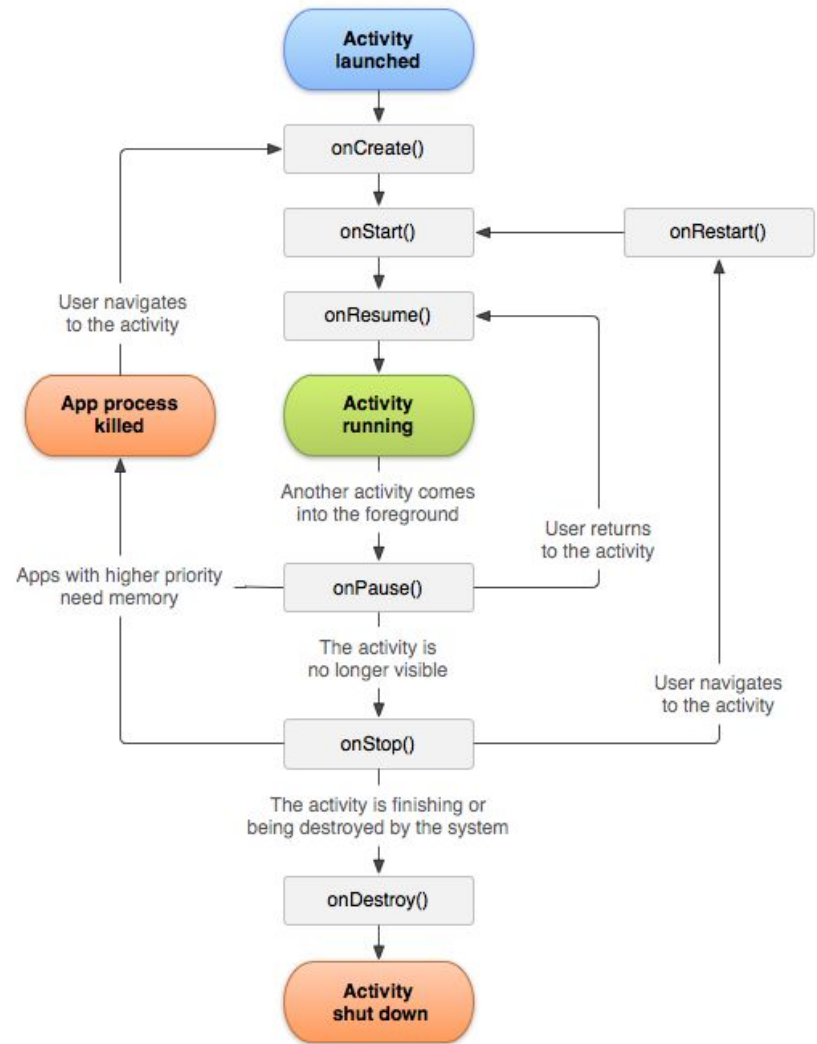
onCreate() / **onDestroy()**: permet de gérer les opérations à faire avant l'affichage de l'activité, et lorsqu'on détruit complètement l'activité de la mémoire. On met en général peu de code dans **onCreate()** afin d'afficher l'activité le plus rapidement possible.

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. **onPause()** ne doit pas être trop long, c'est un état transitoir, pas un état stationnaire.

onRestart(): cette méthode supplémentaire est appelée quand on relance une activité qui est passée par **onStop()**. Ensuite, **onStart()** est aussi appelée. Cela permet de différencier le premier lancement d'un re-lancement.

Cycle de vie d'une Activité



Cycle de vie d'une activité

A quoi bon cela peut servir ?

- S'assurer que notre appli ne va pas se crasher si on reçoit un appel ou si on ouvre une autre appli lors de l'utilisation de notre application.
- Eviter de consommer inutilement des ressources
- Ne pas perdre l'état des interactions si l'utilisateur est obligé d'interrompre sa tâche pour plus tard
- Ne pas se cracher ou perdre l'état en cas de rotation de l'écran.

Cycle de vie : Pause/Resume

Voici les principales situations où une Activité peut passer en statuts Pause/Resume :

- Activité partiellement visible
- Activité en train d'être stoppée
- Activité visible en mode multi-activités (à partir de 7.0)

Opérations à faire en mode Pause :

- Vérifier la visibilité, si l'activité n'est pas visible, arrêter les animations, lecture vidéo...ou toute autre chose qui peut consommer la CPU
- Sauvegarder des états, des données en vue d'un potentiel Stop
- Libérer les ressources système (GPS Par Exemple)



Cycle de vie : Stop/Start/Restart

Voici des exemples de situations où une Activité peut basculer entre statuts Stop/Start:

- L'utilisateur lance une autre application
- Lancement d'une nouvelle activité depuis la même application (stopped=>restart)
- Réception d'un appel entrant

Principalement pour faire des sauvegardes plus lourdes que celles habituellement effectuées lors d'un Pause

- Une activité peut être détruite automatiquement par le système alors qu'elle est en mode Stopped.

Sauvegarde de l'état de l'Activité

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. En effet, lorsque l'on appuie par exemple sur la touche *Home*, en revenant sur le “bureau”, Android peut-être amené à décharger les éléments graphiques de la mémoire pour économiser les ressources. Si l'on re-bascule sur l'application (appui long sur *Home*), l'application peut avoir perdu les valeurs saisies dans les zones de texte...

Si une zone de texte n'a pas d'identifiant, Android ne pourra pas la sauver et elle ne pourra pas être restaurée à partir de l'objet **Bundle**.

Si l'application est complètement détruite (tuée), rien ne peut être restauré.


Sauvegarde de l'état de l'Activité

Le code suivant permet d'ajouter plus d'informations à la sauvegarde:

```
protected void onSaveInstanceState(Bundle savedInstanceState) {  
    savedInstanceState.putInt("playerScore", mCurrentScore);  
    savedInstanceState.putInt("playerLevel", mCurrentLevel);  
    super.onSaveInstanceState(savedInstanceState);  
}
```

Ces valeurs peuvent être récupérées par la suite dans le Bundle de onCreate(..)

Exercice

- Créez une Activity, 
- Redéfinissez les différentes méthodes du cycle de vie,
- Afficher dans chacune un message de log à l'aide de la classe Log,
- Observez l'ordre de l'affichage de ces messages dans la console dans les différents cas de figures. N'oubliez pas de tester la rotation de l'écran.

